Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Návrh kombinačních logických obvodů pomocí genetického algoritmu

Jiří Podivín

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jiří Podivín

Informatika

Název práce

Návrh kombinačních logických obvodů pomocí genetického algoritmu

Název anglicky

Design of Boolean logic circuits using genetic algorithm

Cíle práce

Bakalářská práce je zaměřena na genetické algoritmy a jejich využití při návrhu kombinačních logických obvodů. Hlavním cílem práce je nalézt, popsat a implementovat vhodnou fitness funkci pro evoluční návrh jednoduchých kombinačních logických obvodů.
Další cíle práce jsou:
-optimalizace výsledného řešení
-analýza vlivu komplexity zadání na čas nutný pro vyřešení úlohy
-analýza vlivu velikosti populace na čas nutný pro vyřešení úlohy
-implementace jednoduchého grafického zobrazení výsledného obvodu

Metodika

Metodika řešené problematiky je založena na studiu odborných zdrojů.

Řešení je poté implementováno pomocí aplikace napsané v jazyce C# na platformě .NET. Na základě analýzy fungování této aplikace budou formulovány závěry této práce. V praktické části budou využity vybrané standardní nástroje softwarového inženýrství.

Doporučený rozsah práce

35-40 stran

Klíčová slova

genetický alogritmus, logické obvody, C#, .NET, syntéza

Doporučené zdroje informací

HYNEK, Josef. Genetické algoritmy a genetické programování. Praha: Grada, 2008. Průvodce (Grada). ISBN 978-80-247-2695-3.

.NET Framework Class Library. Microsoft Developer Network [online]. USA: Microsoft, 2016 [cit. 2016-06-07]. Dostupné z: https://msdn.microsoft.com/en-us/library/mt472912(v=vs.110).aspx

SEKANINA, Lukáš. Evoluční hardware: od automatického generování patentovatelných invencí k sebemodifikujícím se strojům. Praha: Academia, 2009. Gerstner. ISBN 978-80-200-1729-1.

Předběžný termín obhajoby

2017/18 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 02. 03. 2018

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Návrh kombinačních logických obvodů pomocí genetického algoritmu" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 2.3.2018 Jiří Podivín

Poděkování

Rád bych touto cestou poděkoval Ing. Jiřímu Brožkovi, Ph.D. za jeho odborné konzultace a svým rodičům za podporu.

Návrh kombinačních logických obvodů pomocí genetického algoritmu

Abstrakt

Cílem práce je zhodnotit proveditelnost a praktičnost návrhu kombinačních logických obvodů genetickým algoritmem za předpokladu simulace kandidátských obvodů na počítači. V teoretické části práce je popsán princip genetických algoritmů, popis algoritmu zvoleného pro práci samotnou a specifika problematiky evolučního návrhu hardwaru. Teoretická část se také věnuje předchozím pracím v odvětví evolučního návrhu hardwaru a jejich implikacím vzhledem k výsledkům, a praktické stránce práce.

Ve vlastní práci je popis implementace algoritmu, hardwaru užitého pro simulaci obvodů a selekci kandidátů a způsob sběru a analýzy dat. Dále jsou zde představena a analyzována kandidátská a optimální řešení daných úloh.

Bylo zjištěno, že ačkoliv je možné navrhovat logické obvody genetickým algoritmem, je tento přístup značně náročný na výpočetní prostředky, a to především kvůli simulaci samotných obvodů.

Klíčová slova: genetický algoritmus, logické obvody, C#, .NET, syntéza

Design of Boolean logic circuits using genetic algorithm

Abstract

The goal of the work is to evaluate feasibility and practicality of designing combinatory logic circuits by genetic algorithm, with the simulation of candidate circuits in a computer. The theoretical part of the work describes principles of genetic algorithms, description of algorithm chosen for the work itself and specifics of evolutionary hardware design.

The theoretical part also summarizes previous work in the field of evolutionary hardware design and its implications regarding results described in the practical part of the work.

The practical part describes the implementation of the chosen algorithm, used hardware, selection of candidates, collection and analysis of collected data, along with the description of the specific candidate and optimal solutions of given problems.

The results show that while the design of logic circuits using genetic algorithms is feasible, it requires substantial computational resources, especially due to a simulation of individual circuits.

Keywords: genetic algorithm, logic circuits, C#, .NET, synthesis

Obsah

Úvod

V posledních třech dekádách došlo k mohutnému rozvoji informačních technologií spojených s využitím umělé inteligence a genetických algoritmů, a to především díky snadnější dostupnosti výkonného hardwaru.
Jedním z možných využití těchto postupů je i návrh hardwarových řešení, a to včetně elektronických obvodů.
Princip genetických algoritmů se inspiruje v evoluci a spočívá především ve využití zjednodušené verze procesu přirozeného výběru. Během průběhu algoritmu jsou jednotlivá kandidátská řešení ohodnocena podle daných kritérií, jejich ohodnocení poté kladně ovlivňuje pravděpodobnost jejich vstupu do další generace.

Před vznikem nové generace jsou kandidátská řešení upravena pomocí operátorů mutace a křížení. Tyto operátory jsou aplikovány s pravděpodobností určenou tvůrcem algoritmu nebo v některých případech algoritmem samým.

Nejedná se přitom o zcela novou myšlenku. Již v padesátých letech Alan Turing navrhl využití principů evoluce ve výpočetní technice a během následujících dekád došlo v tomto odvětví k mohutnému rozmachu, který kulminoval právě během posledních let převedením genetických algoritmů z laboratoří a teorie do průmyslového procesu.

Od té doby již byly mnohokrát aplikovány, a to s velkým úspěchem.
V odvětvích tak různorodých jako aviatika, návrhy softwaru nebo elektrotechnika bylo s pomocí genetických algoritmů dosaženo značných úspor a inovací.

Je nesporné, že v následujících dekádách dozná toto odvětví značného vývoje, ať už z důvodu zlepšování výpočetních schopností počítačů samotných, díky lepšímu využití stávajících prostředků a nástrojů; jako například masivního paralelismu a pokroku v oblasti FPGA obvodů nebo díky moderním výrobním technikám, které mohou značně uspíšit testování navržených řešení.

Využití v praxi se ale setkává s řadou problémů: od náročného procesu tvorby a selekce kandidátských řešení, přes cenu a omezenou znovu použitelnost FPGA obvodů, až k příliš úzce specializovaným řešením, která jsou závislá na místních podmínkách.

Genetické algoritmy byly také kritizovány pro údajně nízkou efektivitu využití výpočetních zdrojů a obtížnost nastavení počátečních podmínek.
Další překážkou širšího využití genetických algoritmů je jejich samotná podstata odvozená z přirozeného procesu evoluce.

Pokud nejsou požadavky nastaveny s dostatečnou přesností nebo v případě, že fitness řešení není správně ohodnocena, může algoritmus poskytnout řešení, které, ač správné podle zadaných podmínek, není implementovatelné, případně splňuje zadání způsobem, který je v přímém rozporu se zájmy jeho poskytovatele.
Příkladem mohou být právě již zmíněná zařízení, jejichž chod závisí na unikátních místních podmínkách.

Jedním z možných východisek, je využití kandidátských řešení, simulovaných na počítači namísto fyzických zařízení.
Náklady na návrh se tak mohou značně snížit, neboť není nutné mnohokrát vyrobit a poté testovat fyzický stroj, a stejně tak je možné omezit riziko úzce specializovaných řešení, která jsou aplikovatelná pouze ve specifických podmínkách, kterými se evolučně navržený hardware nechvalně proslavil.

I zde se však vyskytují překážky, především výpočetní náročnost simulace a otázka její přesnosti. Tyto dva problémy jsou ze zřejmých příčin úzce spojeny a budou dále popsány v této práci. Zvláště druhý z nich přitom může zcela odstranit výhody simulovaných řešení oproti jejich fyzickým protějškům.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je především prozkoumat vhodnost genetických algoritmů k vývoji a výzkumu kombinačních digitálních obvodů, a to za předpokladu, že jsou kandidátská řešení testována ve virtuální podobě. Popřípadě, za jakých okolností a v jakých mezích by byly genetické algoritmy k tomuto účelu vhodné.
Dále je cílem určit závislost mezi parametry algoritmu a dobou zpracování úlohy.

Zvláštní pozornost je věnována vztahům mezi délkou doby zpracování a velikostí populace, pravděpodobností mutace a pravděpodobností křížení.

2.2 Metodika

K simulaci kandidátských řešení byla použita aplikace vyvinutá autorem této práce v jazyku C# s pomocí Visual Studio 2015. Optimální řešení byla pro ověření přesnosti a validity ručně přenesena do programu LogiSim, verze 2.1.7 a programu CedarLogic, verze 1.5, a pak otestována všemi validními vstupy daného obvodu.
Stejná aplikace, vyvinutá autorem práce, byla využita při tvorbě, selekci a ohodnocení kandidátských řešení.
Aplikace využívá platformu .NET verze 4 a pouze základní knihovny dodané s touto platformou. Při její tvorbě nebyly použity knihovny třetích stran, ani kód vyvinutý třetí stranou. Pro zadávání parametrů jednotlivých testů byl vytvořen jednoduchý interpreter příkazů, bez použití kódu nebo knihoven třetích stran.

2.2.1 Simulace obvodu a popis funkce virtuálních hradel

Obvod byl simulován pomocí virtuálních hradel typu NAND, která určují svůj stav a výstupní hodnoty podle výstupů hradel k nim přímo připojených.
Proces rekurzivně pokračuje až k samotným vstupům obvodu, kde hradla načtou hodnoty předem určené aplikací.

Ke splnění zadání měl algoritmus k dispozici virtuální hradla NAND. Hradlo NAND bylo vybráno, neboť tvoří kompletní logiku, a je s ním možno zkonstruovat všechna ostatní hradla, a potažmo i všechny běžně používané logické obvody.

Každé virtuální hradlo disponuje předem definovaným množstvím vstupů připojených na ostatní hradla v kandidátském řešení.
Počet vstupů hradel, i počet hradel samotných, je dán během zadání problému a během průběhu algoritmu se nemění.
Určení množství hradel a jejich vstupů před spuštěním samotného algoritmu, umožňuje značně zmenšit prohledávací prostor řešení a urychlit nalezení řešení optimálního. Omezení také brání selhání samotného programu v důsledku vyčerpání systémových zdrojů.
Hradlo může využít jako vstup hodnoty poskytnuté jinými hradly či hodnoty dodané programem samotným.

Program umožňuje také to, aby hradlo přijímalo i hodnoty ze sebe samého, a tak vytvářelo sekvenční obvody, nicméně pro potřeby této práce byla simulace omezena na obvody kombinační.

V důsledku toho je existence zpětnovazebních smyček v obvodu, a to libovolné délky, považována algoritmem za chybu a negativně se projevuje na ohodnocení kandidátského řešení. V případě existence smyčky se hodnota funkce fitness kandidátského řešení snižuje za každé hradlo zapojené do smyčky, ať už je hradlo zapojeno přímo nebo přes jiné hradlo.

Hradla jsou simulována jako objekty s metodou Signal(), odpovídající logické funkci NAND a proměnnými obsahujícími informace o svých vstupech, stavech a výstupech.

Jednotlivá hradla jsou organizována v datové struktuře typu pole, kde indexy hradel slouží jako jednoznačné identifikátory a během průběhu algoritmu jsou tyto indexy použity k získání informací o stavu hradel a k odvození jejich polohy ve výsledném obvodu. Posledně jmenovaná informace je poté použita k vykreslení struktury obvodu do souboru typu SVG. Po celou dobu existence pole si hradla zachovávají svou pozici a mění se pouze jejich logický výstup.

Každé hradlo má uložené indexy hradel, od kterých získává své vstupy.
V průběhu simulace metoda Signal() přepne hradlo do stavu výpočtu a využije indexy k určení hradel, jejichž výstup má vyžít jako vstupní hodnoty funkce NAND. Následně zkontroluje stavy těchto hradel. Pokud u nich nalezne existenci smyčky nebo jiné chyby, přepne své hradlo do stavu chyba, v opačném případě zavolá jejich funkci Signal(). Celý proces se poté opakuje, dokud není dosaženo vstupů samotného obvodu nebo pokud není v obvodu objevena smyčka.

Vliv smyčky se projevuje na všech hradlech, která se jí účastní i na všech hradlech, která získávají své vstupy z hradel do smyčky zapojených.

Existence smyčky má proto na hodnocení řešení vliv přímo úměrný množství hradel, která jsou součástí smyčky i hradel, která získávají alespoň jednu vstupní hodnotu od hradel, jež jsou součástí smyčky.

2.2.2 Vybraná zadání

Jako testovací zadání pro algoritmus bylo vybráno toto: jednobitová sčítačka, generátor parity a multiplexor se čtyřmi signálovými vstupy.

Uvedená zadání byla vybrána s přihlédnutím k jejich jednoduchosti, snadnému ověření validity řešení, malému množství validních vstupů a obvyklosti v praxi i výuce.

Pro potřeby potvrzení fungování aplikace samotné a jejího debuggování byla použita i zadání složitější, jako například jednoduchá ALU a násobička, spolu se zadáními vytvořenými pseudonáhodným generátorem posloupností.
Zmíněná zadání ale nebyla, v důsledku své náhodné podstaty nebo v případě ALU délky doby zpracování, použita ke sběru statistických dat a pro účely této práce dále nemají význam.

Algoritmus byl u všech zadání omezen počtem hradel, počtem možných vstupů jednoho hradla a také typem hradel samotných.

Lepších výsledků, přinejmenším z hlediska nutného času a výpočetních zdrojů, by bylo možné dosáhnout poskytnutím více druhů hradel nebo zavedením volnějších parametrů pro jejich vstupy, nicméně v takovém případě by se aplikace stala náchylnější k chybám a její debuggování by zabralo neúměrně dlouhý čas vzhledem k jejímu konečnému užití.

3 Teoretická východiska

3.1 Genetické algoritmy

3.1.1 Úvod

Genetický algoritmus je heuristickým algoritmem spadajícím do širší kategorie evolučních algoritmů, která obsahuje mimo jiné genetické programování, evoluční strategie a evoluční programování.
Všechny tyto techniky se inspirují Darwinovou evoluční teorií, popřípadě teoriemi z ní odvozenými, a využívají idealizovaný model přirozeného výběru. (Lažaňský, 2001)

Stejně jako u ostatních optimalizačních algoritmů je možné i algoritmy evoluční popsat jako procesy, hledající globální extrém funkce

$$f: A \to \mathbb{R},$$

tato funkce se nazývá funkcí účelovou, a v případě evolučních algoritmů funkcí fitness. (Sekanina, 2009)
Hodnota fitness funkce se dá považovat za analogii míry přizpůsobivosti organismů jejich prostředí, a v případě evolučních algoritmů slouží ke srovnání jednotlivých členů populace mezi sebou, a ke zjištění vhodnosti stávajících řešení poskytnutých algoritmem.

Je nutné poznamenat, že genetický algoritmus není vhodný pro všechny optimalizační úlohy – fakt, který již byl matematicky dokázán. (Lažaňský, 2001)

Teorém NFL neboli No free lunch teorém, dokazuje, že pro dostatečně velkou množinu optimalizačních problémů nemůže žádný prohledávací algoritmus v průměru překonat prosté systematické prohledáváni všech možných řešení. (Wolper & Macready, 1997)

Efektivita genetických algoritmů, stejně jako ostatních optimalizačních algoritmů, je tedy fundamentálně omezena.

Funkce f nemusí být nutně zcela definována, v mnohých případech to řešené zadání dokonce znemožňuje. Pro potřeby evolučních algoritmů je především důležité, aby výpočet hodnoty fitness funkce trval co nejkratší dobu, v závislosti na řešené problematice mohou ale existovat i další omezení. (Lažaňský, 2001)

Požadavek na co nejkratší dobu výpočtu hodnoty fitness funkce vyplývá z faktu, že pro ukončení algoritmu je zpravidla třeba velkého množství generací, přičemž všechna kandidátní řešení dané generace musejí být ohodnocena již zmíněnou funkcí.

Výpočetní náročnost fitness funkce se tedy může drasticky projevit na délce běhu algoritmu, zvláště pokud je množina kandidátských řešení značně velká.

Evoluční algoritmy používají fitness funkci k ohodnocení množiny kandidátských řešení, tato množina se nazývá populace a díky ní může algoritmus prohledávat prostor možných řešení paralelně. (Lažaňský, 2001)

Kandidátská řešení jsou pak dále upravována a mezi sebou kombinována operátory, odvozenými z biologických procesů. Mezi nejznámější patří mutace a křížení. (Sekanina, 2009)

Rysy jednotlivých kandidátských řešení, díky genetickým operátorům, ovlivňují vlastnosti celé populace.

Na začátku hledání řešení je vygenerována počáteční populace (množina) kandidátských řešení. Podle typu úlohy je zde možno použít zcela náhodně generované kandidáty nebo kandidáty získané jinými heuristickými procesy,

popřípadě jsou použita již existující řešení. Oba přístupy mají své klady a zápory. (Lažaňský, 2001)

Náhodně generovaná řešení nám umožňují pokrýt od začátku větší část prohledávacího prostoru, na druhou stranu ale mohou prodloužit délku běhu algoritmu a vést k divergenci populace.

Řešení vygenerována jinými heuristickými algoritmy, anebo již známá řešení získaná jinými způsoby, nám dovolují ušetřit čas, který bychom jinak pravděpodobně strávili v oblastech prohledávacího prostoru s nízkou hodnotou fitness.
Bohužel ale mohou vést k předčasné konvergenci algoritmu na lokální optimum a minutí řešení v jiných částech prohledávacího prostoru. To platí zvláště u problémů s neznámou hodnotou globálního extrému. (Lažaňský, 2001)

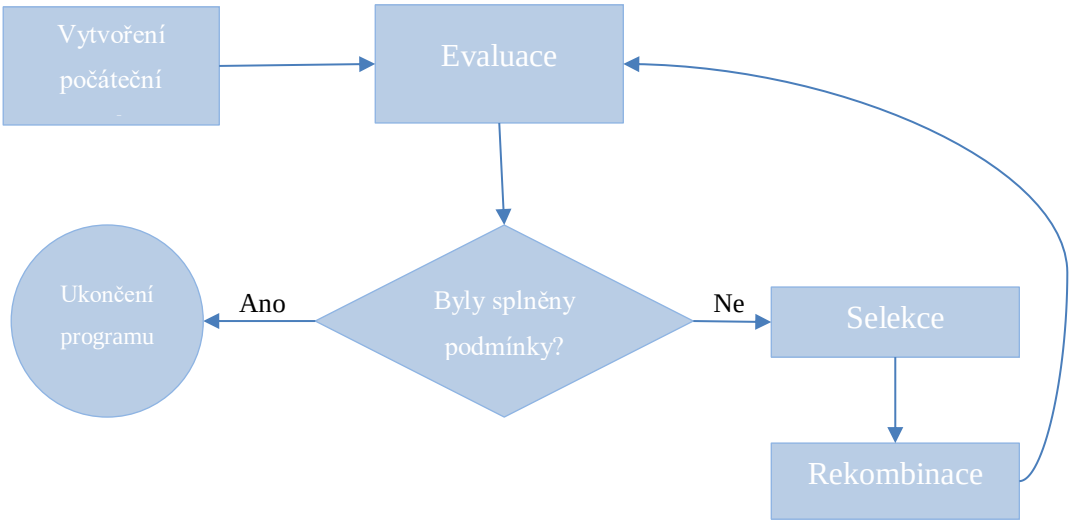
Takový pokles různorodosti dané populace nazýváme degenerací. (Sekanina & Vašíček, 2015)
Degenerace populace může značně prodloužit délku běhu algoritmu, a v případě špatně navržené fitness funkce nebo příliš nízké pravděpodobnosti vzniku nových rysů v populaci, dokonce zabránit nalezení optimálního řešení.

Kandidátní řešení jsou dále ohodnocena funkcí fitness.
Z množiny ohodnocených kandidátních řešení je vybrána podmnožina „rodičů“, kteří se podílejí na tvorbě nové generace, přičemž výběr kandidátních řešení do této množiny je ovlivňován jejich hodnotou fitness.
Obecně platí, že lépe ohodnocená řešení mají větší pravděpodobnost výběru do množiny rodičů, a z toho vyplývající šanci přenést své vlastnosti do nové populace, než řešení ohodnocená hůře. Síla zmíněného vztahu mezi hodnotou fitness a pravděpodobností výběru ale závisí na rozhodnutí návrháře algoritmu.
Průměrná fitness populace se tak postupně zvyšuje. Konvence převzatá z biologie určuje, že vyšší hodnoty funkce fitness označují kandidátská řešení blíže ke globálnímu optimu. (Lažaňský, 2001)

Na vybraná „rodičovská“ řešení jsou aplikovány genetické operátory, obvykle křížení a mutace, a tyto operátory mají protichůdný, ale životně důležitý vliv na chod algoritmu. Mutace umožňuje vznik nových rysů, vede tedy k divergenci populace, zatímco křížení vede k ustálení stávajících rysů, umožňuje tedy konvergenci populace a zachování rysů s pozitivním vlivem na fitness kandidátních řešení.

Z nově vzniklých řešení, takzvaných „potomků“ a jejich „rodičů“, jsou vybráni členové nové populace. Výběr může být implementován různými způsoby, vždy jsou však brány v potaz hodnoty fitness jedinců, přičemž vyšší hodnoty vedou k vyšší pravděpodobnosti vybrání daného jedince. (Lažaňský, 2001)

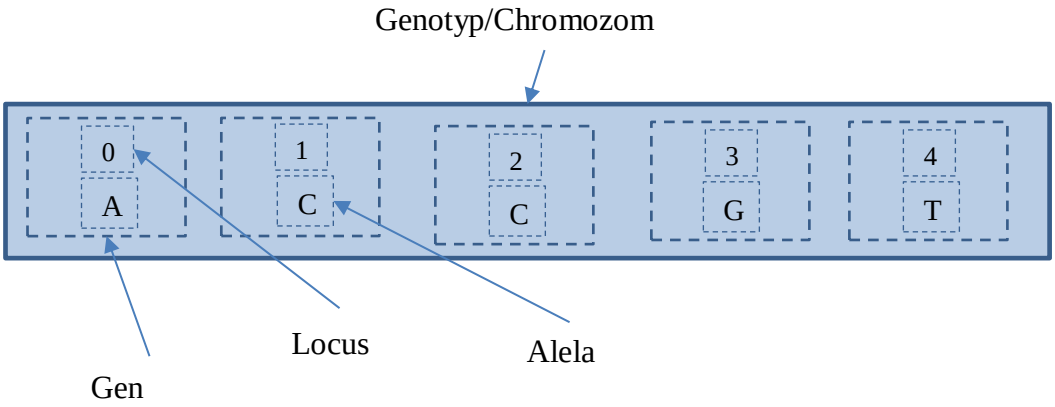
Nakonec jsou všechna kandidátní řešení ohodnocena funkcí fitness a celý proces se opakuje, dokud není dosaženo požadované hodnoty funkce fitness. Jeden tento cyklus se obvykle nazývá generací. (Lažaňský, 2001)



Obr. 1 Schéma genetického algoritmus Zdroj: Vlastní tvorba, podle (Lažaňský, 2001)

Pokud je algoritmus vhodně navržen, bude se postupně zvyšovat průměrná hodnota fitness celé populace, a to až do doby, než je nalezeno uspokojující řešení nebo není vyčerpán předem daný počet generací. (Lažaňský, 2001)

Kandidátní řešení jsou ve své zakódované podobě nazývána genotypy nebo chromozomy. Formát chromozomů se liší podle řešeného problému a může sestávat z binárního, celočíselného nebo jiného řetězce znaků nebo se může jednat, jako v případě genetického programování, o graf. V principu však chromozom může být v jakémkoliv formátu, který je počítač schopný zpracovat. Stejně tak může mít chromozom libovolnou délku.
V obou případech je ale nutné vzít v potaz faktor náročnosti zpracování na fyzické prostředky počítače a podstatu řešeného problému.



Obr. 2 Schéma chromozomu Zdroj: Vlastní tvorba, podle (Lažaňský, 2001)

Jednotlivé znaky řetězce, popřípadě uzly grafu, se nazývají geny, a jejich poloha v chromozomu se nazývá locus, hodnota daného genu se obecně označuje za alelu.

Z chromozomu nebo genotypu je dále odvozen fenotyp, samotná implementace, nebo simulace, řešení popsaného chromozomem. (Lažaňský, 2001)

V biologii se fenotypem rozumí soubor všech pozorovatelných znaků daného organismu. Může se jednat například o fyziologické vlastnosti, zabarvení nebo chování. Ve všech případech ale platí, že fenotyp nezávisí pouze na genotypu daného

organismu. Na fenotyp působí mimo genotypu i prostředí, a v důsledku toho můžeme i u geneticky identických organismů pozorovat různé projevy. (Britannica, 2016)

Není výjimkou, že organismy s téměř identickým genotypem vykazují radikálně odlišené fenotypy. (Taylor, 2017)

Oproti tomu genetické algoritmy nemusí, v závislosti na požadavcích dané úlohy, rozlišovat mezi genotypem a fenotypem.
Pokud má kupříkladu genetický algoritmus poskytnout řetězec znaků odpovídající zadanému vzoru, mohou být fenotyp a genotyp identické řetězce znaků, které jsou srovnávány se vzorem, a dále ohodnoceny fitness funkcí v závislosti na tom, jak se od něj liší. (Fogel, 1995) Slovo chromozom má v genetice také jiný význam. Zatímco názvosloví genetických algoritmů používá chromozom jako výraz ekvivalentní s termínem genotyp, genetika označuje chromozomem pouze specifickou strukturu, obsahující podmnožinu celkového genotypu dané buňky. (Sekanina, 2009)

I v ostatních případech termínů převzatých z genetiky se význam oproti původnímu mnohdy liší. Tyto rozdíly jsou podrobně popsány v následujících oddílech.

3.1.2 **Vztah mezi genotypem a fenotypem**

Aby mohla být získána hodnota fitness kandidátního řešení, je třeba genotyp neboli chromozom, tohoto řešení převést do podoby, umožňující získání hodnoty funkce fitness. Tuto podobu nazýváme fenotypem, nicméně charakteristiky fenotypu silně závisí na problémové doméně a volbách autora algoritmu.

Na přesné definici vztahu mezi fenotypem a genotypem se literatura neshoduje. Obecným pravidlem ale je, že genotyp je méně abstrahovanou reprezentací řešení

než fenotyp. Je tedy v principu možné, že více genotypů odpovídá jednomu fenotypu, ale opak neplatí. (Fogel, 1995)

V některých případech je genotyp a fenotyp ekvivalentní, a hodnotu funkce fitness je možné získat srovnáním genotypu se vzorem optimálního řešení. Případně vykonáním programu, pokud se jedná o vykonavatelný kód nebo pokud jde o vhodnou úlohu z oblasti genetického programování.

Jiné úlohy ale vyžadují, aby byl z genotypu kandidátního řešení před jeho ohodnocením vytvořen jeho fenotyp, neboť v důsledku podstaty řešeného problému neumožňuje genotyp samotný zjištění hodnoty fitness.

Tvorba fenotypu kandidátního řešení potom může mít různou podobu.
U jednodušších úloh může každý gen určovat jeden parametr fenotypu kandidátního řešení, zatímco ostatní parametry jsou určené samotným zadáním úlohy.
Pokud jsou všechny parametry řešení určeny geny, jedná se o vzájemně jednoznačné zobrazení množiny genů do množiny rysů kandidátního řešení.
Tento postup je ve své podstatě velmi podobný výše zmíněnému případu, kdy je fenotyp a genotyp shodný.

Mnohé úlohy ovšem vyžadují napsání algoritmu, schopného vytvořit fenotyp kandidátního řešení podle série příkazů reprezentovaných jednotlivými geny chromozomu. Algoritmus v tomto případě nachází inspiraci v embryogenezi živých organismů, a vyžaduje vytvoření sady instrukcí vhodných pro daný problém.

U složitějších úloh, jako jsou kupříkladu umělé neuronové sítě, se ale definice fenotypu stává více problematickou.
Parametry použité k tvorbě sítě jsou sice genotypem řešení, ale k určení hodnoty fitness není možné použít struktury sítě samotné, nýbrž jejího chování v závislosti na daných vstupech. Podle výše uvedené definice se tedy chování stává fenotypem, a struktura sítě pouze vyšší abstrakcí genotypu. (Fogel, 1995)

Systémy vykazující velmi složité chování potom jednoduché definici fenotypu brání množstvím chování a rysů, které vykazují. Příkladem mohou být sebe replikující programy v prostředích Tierra a Avida. (Csontó, 2001)

V takovém případě musí výzkumník definovat fenotyp sám jako soubor znaků, které u dané populace studuje, a jejichž vliv hodnotí.
Tento postup je přitom velice blízký analogickému postupu biologů.

3.1.3 **Fitness funkce**

Účelovou funkci genetického algoritmu označujeme za fitness funkci.
Fitness funkce ohodnocuje jednotlivá kandidátní řešení podle kritérií zvolených návrhářem algoritmu. Konvence převzatá z evoluční biologie určuje, že jejím oborem hodnot jsou kladná reálná čísla a vyšší hodnoty označují lepší kandidátní řešení. (Lažaňský, 2001)

Definice toho, co je lepší řešení, je zásadním krokem navrhování fitness funkce a má kritické důsledky na fungování algoritmu.
Nevhodné nastavení kritérií úspěšnosti kandidátního řešení se nutně negativně projeví na fungování algoritmu, a může vést ke stagnaci hodnoty fitness, nebo i k označení suboptimálního řešení za řešení optimální.

Hodnota fitness funkce může být v závislosti na řešeném problému vyjádřena několika způsoby. (Karásek & Crvrk, 2013)

Jeden z nejjednodušších je takzvaná hrubá fitness, jejíž hodnoty jsou přímo závislé na daném problému. Vyšší hodnoty přitom označují lepší kandidátní řešení.
Toto vyjádření dále neprochází žádnou úpravou a může být rovnou použito při vybírání „rodičů“ následující generace.

Standardizovaná fitness transformuje hrubou fitness do podoby, kde nižší hodnota fitness označuje lepší kandidátní řešení. Ve standardizované fitness je za nejlepší možné ohodnocení považována 0. (Karásek & Crvrk, 2013)

Ze standardizované hodnoty fitness je možné získat přizpůsobenou hodnotu fitness. Ta je převrácenou hodnotou součtu již zmíněné standardizované fitness a čísla 1.

$$\frac{1}{standardizovaná\ hodnota\ fitness + 1}$$

Hodnoty fitness tak vždy leží v intervalu

$$(0, 1\rangle$$

Stejně jako u hrubé fitness, i přizpůsobené hodnoty fitness, označují vyšší hodnoty lepší kandidátní řešení. Nicméně interpretace přizpůsobené fitness je snadnější než u fitness standardizované.

Normalizovaná hodnota fitness kandidátního řešení je získána součtem hodnot hrubé fitness celé populace kandidátních řešení a následným vydělením hrubé fitness kandidátního řešení takto získanou hodnotou.

$$\frac{hrubá\ hodnota\ fitness\ kandidátního\ řešení}{\sum_{i=0}^{N-1} hrubá\ hodnota\ fitness\ i-tého\ jedince}$$

Normalizované hodnoty fitness tedy leží v intervalu

$$\langle 0,1\rangle$$

kde vyšší hodnoty značí lepší kandidátní řešení, a součet normalizovaných hodnot fitness celé populace je roven 1. (Karásek & Crvrk, 2013)

Normalizovaná fitness nám ale poskytuje pouze srovnání fitness jednotlivých kandidátních řešení v dané populaci a stejné generaci.
Pomocí normalizované fitness tedy není možné určit, jak je dané řešení blízké řešení optimálnímu, ani jaký trend má průměrná fitness populace.

Pokud problém nemá známé optimální řešení, nebo pokud není možné srovnat kandidátské řešení s řešením optimálním, výše zmíněné negativum neplatí.

3.1.4 **Selekce rodičů**

Algoritmus výběru rodičů následující generace tvoří klíčovou část celého algoritmu. Nesprávně nastavené podmínky výběru mohou vést ke značnému zpomalení chodu algoritmu, popřípadě k jeho úplnému selhání a nenalezení požadovaného řešení.

Efektivita selekčního algoritmu také závisí na množství jedinců původní populace, kteří bez úprav přejdou do generace následující.

Nově vzniklá populace kupříkladu může být tvořena pouze novými jedinci; takto vzniklá množina jedinců se nazývá populací nepřekrývající.
Výhodou tohoto postupu je zvýšení různorodosti nové populace, ačkoliv je v principu možné, že někteří nově vzniklí jedinci budou mít genom shodný s jedinci předchozích generací. Obecně lze však díky tomuto postupu dosáhnout vzdálenějších oblastí prohledávacího prostoru než při zachování neupravených jedinců.
Nevýhodou však je možnost ztráty výhodných rysů v dané populaci. (Sekanina, 2009)

Překrývající populace je oproti tomu tvořena jak jedinci, kteří prošli úpravou genetickými operátory, tak jedinci, kteří již existovali v generaci předchozí.
Podstatně se tak snižuje pravděpodobnost ztráty již existujících výhodných rysů nacházejících se v předchozí generaci.

Při použití takzvaného elitismu je tato možnost dokonce zcela eliminována, neboť nejlépe ohodnocené kandidátní řešení je vždy zachováno beze změn, a pak přeneseno do následující generace.

Algoritmů výběru rodičů následující generace je celá řada, ale v praxi se nejčastěji setkáváme se selekcí deterministickou, turnajovou, ruletovou a selekcí podle pořadí.

Algoritmus deterministické selekce seřazuje kandidátní řešení podle jejich hodnot fitness, a následně vybírá N kandidátních řešení, která mají ohodnocení nejvyšší.
Genotypy zbývajících M jedinců se tak na tvorbě následující generace nepodílí vůbec a jejich rysy tak mohou být ztraceny. (Sekanina, 2009)

Výhodou deterministického selekčního algoritmu je jeho snadná implementace a rychlost výběru rodičů. Populace získané deterministickou selekcí jsou ale méně různorodé než v případě algoritmů zahrnujících náhodný prvek.

Selekce turnajová z populace náhodně vybírá N M-tic jedinců.
Hodnoty jejich fitness jsou dále srovnány mezi sebou navzájem, přičemž nejlépe ohodnocený jedinec dané M-tice se stává rodičem následující generace.

Výsledkem je tedy znovu N vybraných jedinců, kteří se stávají rodiči následující generace. Vybraní jedinci jsou však různorodější než v případě selekce deterministické, a algoritmus tak může dosáhnout vzdálenějších oblastí prohledávacího prostoru.
Pravděpodobnost výběru jedince s nižší hodnotou fitness je nepřímo úměrná hodnotě parametru M. S rostoucí velikostí M-tic stoupá pravděpodobnost, že bude jedinec srovnán s jedincem lépe ohodnoceným.
Oproti tomu jedinec s nejvyšší hodnotou fitness v dané populaci je vybrán vždy, nehledě na velikost M-tic. (Blicke & Thiele, 1995)

V závislosti na hodnotě parametru M tedy rozlišujeme dvě krajní možnosti, obě stejně nežádoucí.

Pokud

$$M = N$$

je vybrán vždy pouze nejlepší jedinec v dané populaci, nová generace tak sestává pouze z potomků tohoto jedince a vzdálenější oblasti prohledávacího prostoru jsou zcela opomenuty. Dochází ke ztrátě různorodosti populace a její degeneraci.

Oproti tomu, v případě že

$$M = 1$$

je do následující generace vybráno všech N jedinců generace stávající, nehledě na hodnotu jejich fitness. Selekce se tak stává zcela náhodnou a algoritmus selhává, neboť pozitivní rysy neposkytují svým nositelům žádnou výhodu.

Ruletová selekce přiřazuje každému jedinci úsek uzavřeného intervalu $\langle 0,1 \rangle$ přímo úměrný jeho fitness, přičemž přiřazený úsek $\langle a_i, b_i \rangle$ musí splňovat následující podmínky:

$$a_i \geq 0$$

$$b_i \leq 1$$

$$|a_i - b_i| \leq 1$$

$$\sum_{i=0}^{n} |a_i - b_i| = 1$$

Po přiřazení úseku každému jedinci z původní populace je vygenerováno číslo c_j ležící v intervalu $\langle 0,1 \rangle$, pokud

$$a_i \geq c_j \leq b_i$$

je jedinec i vybrán a stává se rodičem následující generace. Generování se dále opakuje, dokud není vybrán požadovaný počet rodičů. Vybraní jedinci se ale z původní množiny potencionálních rodičů nevylučují a přiřazené úseky se nemění. Jedinec tak může být vybrán vícekrát, přičemž pravděpodobnost opakovaného výběru je přímo úměrná jeho fitness. Všichni jedinci mají ale nenulovou pravděpodobnost výběru. (Sekanina, 2009)

Pokud má ovšem jeden jedinec výrazně vyšší ohodnocení než ostatní jedinci účastnící se výběru, pravděpodobnost vstupu horších jedinců mezi rodiče klesá na úkor různorodosti celé populace, což vede k její degeneraci.
Vzdálenější oblasti prohledávacího prostoru tak mohou být opomenuty.

Tento problém se projevuje zvláště u populací s menším počtem jedinců a velkým rozptylem hodnot fitness.

Takzvaná selekce podle pořadí staví na ruletové selekci, a řeší výše zmíněný problém klesající různorodosti populace, a z ní plynoucí degenerace.

Stejně jako u ruletové selekce je každému kandidátnímu řešení přiřazen úsek $\langle a_i, b_i \rangle$ z uzavřeného intervalu $\langle 0,1 \rangle$ splňující:

$$a_i \geq 0$$
$$b_i \leq 1$$
$$|a_i - b_i| \leq 1$$
$$\sum_{i=0}^{n} |a_i - b_i| = 1$$

Délka přiřazeného úseku ale nezávisí na fitness jedince, nýbrž na jeho pořadí.
Poměr délek úseků je předem dán a během chodu algoritmu se nemění, délka jednotlivých úseků závisí pouze na počtu jedinců v populaci.

Lépe ohodnocený jedinec má tak stále větší pravděpodobnost výběru než jedinec ohodnocený hůře. Ale výrazně lépe ohodnocení jedinci nesnižují pravděpodobnost

výběru jedinců s ohodnocením horším. Selekce podle pořadí tak zamezuje degeneraci populace. (Sekanina, 2009)

Vzhledem ke stochastické podstatě genetických algoritmů a náhodném prvku v mnoha selekčních algoritmech, hrozí při vzniku nové generace ztráta kladných rysů jedinců z generace předchozí. Průměrná fitness populace tak může stagnovat, potažmo dokonce klesat. Tento jev se projevuje zvláště u menších populací.

Jedním z řešení toho problému je zavedení takzvaného elitismu.
Elitistická selekce přenáší nejzdatnější jedince z generace stávající do generace nové, a to beze změn.
Proces je deterministický, takže rysy neschopnějších jedinců jsou zachovány vždy.
Nevýhodou tohoto přístupu je jisté zvýšení pravděpodobnosti degenerace populace, neboť nejzdatnější jedinci mají na složení následující generace větší vliv, než pokud by prošli mutací a křížením. (Lažaňský, 2001)

Degeneraci v důsledku je ale možné zamezit, pokud je populace dostatečně velká.

3.1.5 **Operace genetických algoritmů**

I když samotný proces selekce zajišťuje větší pravděpodobnost přetrvání kladných rysů a zlepšování kvality populace, nedokáže vytvořit rysy nové.
K tomuto účelu slouží takzvané genetické, nebo také rekombinační, operátory.
Základními operátory genetických algoritmů jsou mutace a křížení. Tyto operátory jsou analogické k biologickým procesům křížení a mutace genomu, liší se však v několika podstatných ohledech. (Lažaňský, 2001)

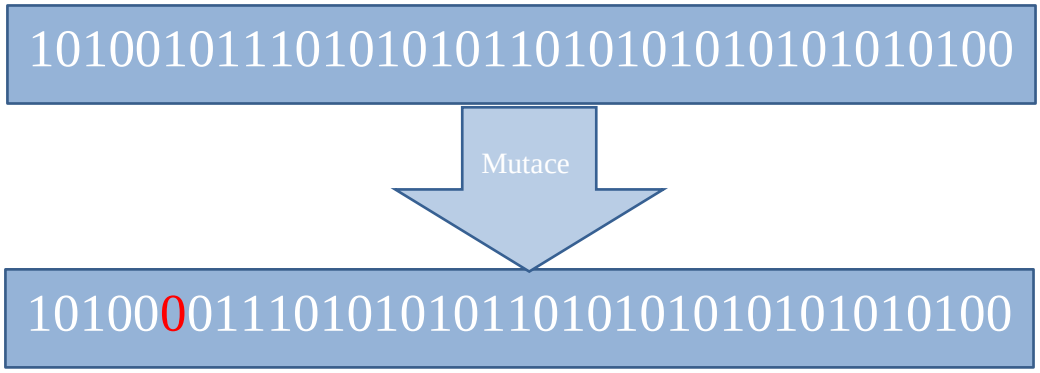
Genetickými operátory je upravena v každé generaci pouze náhodně vybraná část celkové populace. V opačném případě by populace nemohla konvergovat k hledanému řešení a algoritmus by prohledával prostor řešení v podstatě náhodně.

Pravděpodobnost, že na jedince bude aplikován daný operátor, závisí na rozhodnutí návrháře algoritmu. Obecně se však pravděpodobnost aplikace operátoru mutace pohybuje v intervalu $\langle 0.015; 0.02 \rangle$ a pravděpodobnost aplikace operátoru křížení v intervalu $\langle 0.6; 1 \rangle$. (Lažaňský, 2001)

Zvláště pravděpodobnost mutace musí být zvolena poměrně malá, neboť i minimální změna genotypu jedince může vést k radikální změně jeho fenotypu, a tedy i jeho ohodnocení funkcí fitness. Příliš časté mutace mohou tedy vést ke ztrátě kladných rysů jedinců a postupnému poklesu průměrné fitness populace.

Operace křížení může být oproti tomu aplikována s pravděpodobností značně vyšší, protože nezpůsobuje ztrátu genů jedinců. Naopak, operace křížení je zásadní pro rozšíření kladných rysů mezi jedinci dané populace, a tedy i pro růst její fitness.

Operátor mutace mění hodnotu náhodně vybraného genu, popřípadě více genů, na jinou náhodnou hodnotu. Přičemž nová hodnota genu musí být v souladu s požadavky dané problémové domény. Mutace tak zvyšuje různorodost populace a umožňuje algoritmu dosáhnout vzdálenějších oblastí prohledávacího prostoru. (Lažaňský, 2001)
Přímým důsledkem tohoto faktu je skutečnost, že mutace je naprosto zásadní pro zabránění degenerace a vnesení nových rysů do populace. Příliš časté mutace genotypu ale snižují pravděpodobnost udržení stávajících kladných rysů, zvláště u problémů, kde i malá změna genotypu vede k velkým změnám fenotypu. Tak může mutace bránit nalezení optima.



Obr. 3 Schéma mutace binární reprezentace genotypu Zdroj: Vlastní tvorba, podle (Lažaňský, 2001)

Extrémním případem by byla pravděpodobnost mutace $P_m = 1$. Takto nastavený genetický algoritmus by prohledával prostor řešení v podstatě náhodně a kladné rysy by se nemohly přenést na následující generaci.

V závislosti na problémové doméně může být mutace implementována řadou způsobů. Obecně však platí, že z množiny všech genů daného genotypu je vybrána podmnožina velikosti zvolené uživatelem. Hodnoty prvků této podmnožiny jsou změněny tak, aby byly stále hodnotami přípustnými pro danou problémovou doménu. Nové hodnoty mohou být náhodné, anebo v případě binární reprezentace, inverze hodnot původních. (Lažaňský, 2001)

Pokud je například genotyp reprezentován grafem, jako je tomu u genetického programování, mutace náhodně vybírá uzel v grafu a nahrazuje jeho podstrom stromem novým, náhodně vygenerovaným. (Sekanina, 2009)

Podobně jako v biologii, rozdělujeme mutace podle jejich vlivu na fitness jedince:
na mutace kladné, záporné a neutrální. Význam mutací kladných a záporných je zřejmý. (Flegr, 2009)

Mutace neutrální nemění hodnotu fitness, a zvláště v některých složitějších úlohách ani neovlivní fenotyp daného jedince, nicméně jejich vliv se může projevit v důsledku následujících mutací a křížení.

Důležitou vlastností neutrálních mutací je, že pokud není algoritmus navržen s přihlédnutím k jejich existenci, jsou pro něj v podstatě neviditelné. Neovlivňují totiž hodnotu fitness, podle které algoritmus třídí jedince.

Pokud algoritmus nezohledňuje délku genotypu při výpočtu hodnoty fitness a délka genotypu není ani omezena zadáním, vedou neutrální mutace k takzvanému bloatu.

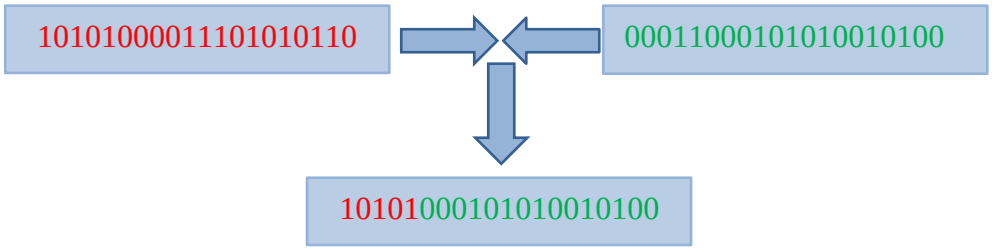
Tento jev spočívá v růstu genotypu bez jakéhokoliv nebo jen minimálního vlivu na fitness jedince a jeho fenotyp. Části genotypu poté nabývají vlastnosti analogické k intronům známých z genetiky.
(Sekanina, 2009)
Ačkoliv se takovéto úseky chromozomu nepodílejí na tvorbě fenotypu jedince, mohou mít značný vliv na dlouhodobou stabilitu rysů v populaci, jelikož stále podléhají vlivu genetických operátorů.

I malá změna genotypu potom může vést k radikální změně fenotypu a podstatné změně hodnoty fitness. Takto rapidní změna vede k divergenci populace, a v důsledku své náhodné podstaty pravděpodobně i ke snížení hodnoty fitness.

Bloat je nežádoucí také kvůli vzrůstajícím nárokům na výkon, které vycházejí z prodlužující se délky genotypů, a rostoucí náročnosti jejich zpracování.

Operátor křížení je druhý ze základních genetických (či rekombinačních) operátorů. Svým vlivem doplňuje a vyrovnává vliv operátoru mutace.
Zatímco vliv mutací na genom vede k divergenci populace a prohledávání širšího prostoru možných řešení, křížení vede ke konvergenci populace a zúžení prohledávané oblasti. (Lažaňský, 2001)

Křížení ke svému fungování vyžaduje, na rozdíl od mutace, dva jedince, které obecně nazýváme rodičovské genotypy.
Operátor křížení kombinuje genotypy těchto jedinců podle předem daných pravidel. Z nich generuje N jedinců nových, přičemž N je nastaveno návrhářem algoritmu: zpravidla $N = 2$.



Obr. 4 Schéma křížení Zdroj: Vlastní tvorba, podle (Lažaňský, 2001)

Stejně jako v případě mutace, i křížení může probíhat několika způsoby. Nejobvyklejším je takzvané jednobodové křížení. To rozděluje chromozomy obou jedinců na dvě části náhodné či předem zvolené délky. Obecně se používá především u chromozomů reprezentovaných binárními řetězci. (Lažaňský, 2001)

Pravá část genotypu prvního rodiče a levá část genotypu rodiče druhého jsou spojeny a tvoří prvního potomka. Potomek druhý je tvořen analogicky levou částí genotypu prvního a pravou částí druhého rodiče. Rysy obou rodičů tak mají šanci se přenést do potomků a mít vliv na fitness následující generace.

Je nutné připomenout, že ačkoliv křížení může podstatně změnit fenotyp jedinců následující generace, neumožňuje vznik alel, které by již nebyly přítomny v genotypech jedinců první generace. Zároveň ale neumožňuje ani ztrátu žádné z alel původní generace – z tohoto hlediska se jedná o „bezpečnou“ operaci.

Díky tomu je možné, s jistými výjimkami, nastavit pravděpodobnost křížení o mnoho výše než pravděpodobnost mutace.

Proto není křížení samo o sobě postačující k nalezení optimálního řešení, vyjma vzácných případů, kdy se nutné alely nacházejí již v první generaci algoritmu.

Jednoduchým zobecněním jednobodového křížení je křížení vícebodové.
Vícebodové křížení probíhá analogicky jako jednobodová varianta, ale s tím rozdílem, že genotyp rodičů je rozdělen na více částí, ze kterých jsou vytvořeni jedinci generace následující. (Sekanina, 2009)

Mezi další možnosti patří záměna genů na náhodně vybraných pozicích nebo použití více rodičů, popřípadě generování více potomků.

Operace křížení se dá zobecnit jako rozdělení m-tice řetězců znaků délek $N_0 \dots N_m$ na k úseků délky n, a následné vytvoření l-tice řetězců délek $N_0 \dots N_l$ spojením dříve jmenovaných úseků. Není přitom nezbytně nutné aby $m = l$, $n = l$ nebo $m = n$, stejně tak není nutné, aby byl bod, popřípadě body, křížení určeny náhodně.
Nastavení těchto hodnot a parametrů závisí na dané problémové doméně.

Některé genetické algoritmy nevyužívají křížení vůbec a vystačí si pouze s mutací.

Mezi tyto algoritmy patří Evoluční programování navržené Lawrencem Fogelem (Lažaňský, 2001) a Evoluční strategie navržené Rachenbergem, Schwefelem a Bienertem (Beyer & Scwefel, 2002).

4 Vlastní práce

4.1 Návrh programu

Genetický algoritmus, spolu s aplikací simulující kandidátská řešení,
byl implementován v jazyku C# využívajícím platformu .NET 4.0.
Verze 4.0 byla vybrána především kvůli své rozšířenosti, a z toho vyplývající snadno dostupné podpoře a podrobné dokumentaci.

Během vývoje aplikace jsem bral v úvahu především nutnost snadného ladění algoritmu a rychlé identifikace a řešení možných chyb.

Většina proměnných aplikace, včetně množství použitých hradel, formátu výstupu, typu požadovaného obvodu a většiny parametrů genetického algoritmu, tedy není pevně zadána a je zvolena uživatelem před spuštěním hledání řešení.

Další parametry, včetně algoritmu výběru rodičů a množství povolených vstupů hradel, nejsou uživateli přístupné, a jsou zvoleny s přihlédnutím k ladění aplikace.

4.1.1 Generátor náhodných čísel

Náhodné hodnoty nutné pro tvorbu kandidátských řešení první generace, testování řešení a jejich následnou úpravu genetickými operátory, poskytuje generátor pseudonáhodných čísel definovaný v třídě Random platformy .NET.

Třída Random využívá algoritmus generování pseudonáhodných čísel založený na modifikované verzi subtraktivního náhodného generátoru popsaného D.E. Knuthem.

Ačkoliv generátor třídy Random není, vzhledem ke svým mnoha zdokumentovaným vadám, vhodný pro užití v kryptografii, vícevláknových aplikacích a obecně aplikacích vysoce citlivých na náhodnost, je zcela dostačující pro generování

kandidátských řešení. Případně pro jejich následnou úpravu rekombinačními operátory, vzhledem k omezené množině přípustných znaků, které mohou být v chromozomu obsaženy a jednovláknové podstatě aplikace.

Nicméně podstatnou výhodou generátoru implementovaném třídou Random je oproti generátorům kryptograficky bezpečnějším, rychlost generování čísel.

Vzhledem k tomu, že genetický algoritmus využívá generátor náhodných čísel mnohokrát během jednoho cyklu (především při zjišťování fitness generace stávající a tvorbě generace nové), má rychlost generátoru značný vliv na dobu běhu algoritmu.

4.1.2 **Struktura chromozomu**

Chromozom kandidátského řešení je implementován jako pole 32 bitových číselných znaků, typu int. Jednotlivá hradla mají n vstupů, vstupy mohou být, podle potřeb daného obvodu, konstanty, výstupní hodnoty jiných hradel nebo hodnoty poskytované aplikací během testování obvodu.

Vlastnosti hradel jsou definovány n-ticí znaků, z nichž každý určuje zdroj jedné ze vstupních hodnot hradla. Množina přípustných znaků je definována počtem bran kandidátního řešení a vstupů obvodu, operátor mutace i náhodně generovaná kandidátská řešení využívají pouze přípustné znaky.
Algoritmus tak brání vzniku kandidátských řešení využívajících neexistující brány a značně tak omezuje prohledávací prostor řešení.

Poloha jednotlivých znaků určuje hradlo, jehož vlastnosti daný znak určuje.

Délka chromozomu se dá určit jako $délka\ chromozomu = počet\ hradel * n$ přičemž tato hodnota, stejně jako hodnota n, se během chodu algoritmu nemění, a je stejná pro všechna kandidátní řešení.

Poloha n-tic v chromozomu určuje indexy logických hradel, jejichž vlastnosti jsou

n-ticemi definovány. Hodnoty jednotlivých znaků v n-tici určují indexy hradel, jejichž výstupy slouží jako vstupní hodnoty hradla definovaného danou n-ticí.

V ranné fázi vývoje aplikace bylo zvažováno přidání dvou dalších znaků definujících logickou funkci a počet povolených vstupů hradla. Tato možnost byla opuštěna v zájmu rychlejšího vývoje a snadnějšího debuggování aplikace.

Ze stejného důvodu bylo během sběru dat $n = 2$, i když aplikace by byla schopna pracovat i s jinými hodnotami.

4.1.3 **Genetické operátory**

Genetický algoritmus k tvorbě nových kandidátních řešení využívá dva rekombinační operátory: jednobodovou mutaci a křížení.

Operace mutace je implementována jako záměna jednoho náhodně vybraného znaku v chromozomu za znak jiný, zvolený z množiny přípustných znaků.

Původní implementace operátoru dávala všem genům v chromozomu stejnou pravděpodobnost změny na znak jiný. Během testování aplikace bylo zjištěno, že délka běhu algoritmu se značně zkrátí, pokud je pravděpodobnost mutace genu nacházejícího se v chromozomu na pozici n určena podle vztahu:

$$P(m_n) = P(M) * \prod_{k=0}^{k=n-1} 1 - P(m_k)$$

Kde $P(M)$ značí pravděpodobnost mutace zadanou uživatelem a $P(m_k)$ pravděpodobnost mutace genu na pozici k.
Důsledkem takto určené pravděpodobnosti mutace je zvýšená stabilita genů, nacházejících se na konci chromozomu, a tedy i jimi určenými vlastnostmi kandidátského řešení.

Pokud je algoritmus nastaven, aby každé hradlo mělo k vstupů, určuje posledních $m * k$ znaků chromozomu vlastnosti posledních m hradel.
A protože výstup posledních n hradel je současně výstupem celého obvodu, a změna vstupů těchto hradel má silnější vliv na výstup obvodu než změna vstupů hradel ostatních, má stabilita těchto genů značný vliv na stabilitu hodnoty fitness.

Aplikace využívá jednobodového křížení s pevně určeným bodem uprostřed chromozomu. Samotná operace je implementována podle vztahu:

$$G_n^{D1} = G_n^{P1}\ n \in \langle 0, \frac{k}{2}) \quad G_n^{D1} = G_n^{P2}\ n \in \langle \frac{k}{2}, k)$$
$$G_n^{D2} = G_n^{P2}\ n \in \langle 0, \frac{k}{2}) \quad G_n^{D2} = G_n^{P1}\ n \in \langle \frac{k}{2}, k)$$

Kde n značí index prvku v chromozomu, k délku chromozomu, G^{D1} a G^{D2} chromozomy potomků. G_n^{P1}a G_n^{P2} značí genotypy rodičů.

Vyjma rekombinačních operátorů implementuje program operátor selekční v podobě ruletové selekce s elitismem.
Selekční algoritmus vybírá z populace $n + 1$ jedinců, z toho n jedinců je vybráno ruletovou selekcí a jeden selekcí deterministickou jako nejlépe ohodnocený jedinec v celé populaci. Přičemž $n = 10$ pro populace menší než 50 jedinců, a $n = \frac{počet\ jedinců\ v\ populaci}{10}$ pro populace větší.

Vybraní jedinci jsou poté, s výjimkou deterministicky vybraného jedince, upraveni rekombinačními operátory. Deterministicky vybraný jedinec je do následující generace přenesen beze změny genotypu, aby se předešlo ztrátě výhodných rysů.

4.1.4 **Výpočet fitness funkce**

Hodnota fitness kandidátského řešení je kombinací míry přesnosti výstupů navrženého obvodu a jeho validity vzhledem k určeným parametrům.

V programu je hodnota fitness funkce implementována jako proměnná typu double.

K určení přesnosti výstupu je použit seznam všech možných vstupů, a k nim příslušných požadovaných výstupů. Je realizován jako datová struktura typu List<T>.

Během testování kandidátského řešení jsou z tohoto seznamu náhodně vybírány dvojice vstup-výstup, přičemž každá dvojice je otestována právě jednou.
Náhodné pořadí testovacích vstupů je nutné pro zamezení vzniku kandidátského řešení, které poskytuje stejnou sekvenci výstupů nehledě na dodané vstupní hodnoty. Případně vykazuje jiné anomální, a se zadáním neslučitelné, chování.
Takové řešení může být mylně označeno za optimální, ačkoliv se nejedná o požadovaný typ obvodu.
Během ladění fitness funkce se potvrdilo, že genetický algoritmus má tendenci poskytovat pouze nejjednodušší možná řešení, která stále ukazují požadované výstupní hodnoty.

Pokud výstupní hodnoty odpovídají hodnotám očekávaným, je hodnota fitness obvodu zvýšena o Δ.

$$\Delta = \frac{1}{Počet\ validních\ výstupů\ požadovaného\ obvodu}.$$

Po srovnání výstupních hodnot s hodnotami očekávanými je zjištěno, zda kandidátské řešení neobsahuje zpětnovazební smyčky, které mohou dát vzniknout anomálnímu chování řešení nebo samotného programu.

Za každé hradlo, které je součástí smyčky, či získává některé své vstupní hodnoty ze smyčky, je hodnota fitness obvodu snížena o 0.0000001.

Penalizace zpětných vazeb zabraňuje vzniku sekvenčního obvodu, aproximujícího chování požadovaného obvodu kombinačního, který by mohl být programem považován za optimální řešení.

Zvolená hodnota penalty zpětné vazby 0.0000001, byla vybrána s přihlédnutím k přesnosti datového typu double, implementovaného standardem .NET, a snaze zamezit ztrátě potenciálně nadějných řešení v důsledku existence smyček.

Během testování funkčnosti programu bylo zjištěno, že příliš vysoká hodnota penalty může oddálit nalezení řešení úlohy, snížením konečné fitness o hodnotu $h \geq \Delta$.

4.1.5 **Zadané úlohy**

Zadání pro algoritmus sestávají ze seznamu všech možných vstupů požadovaného obvodu a seznamu odpovídajících výstupů. Společně tak seznamy tvoří pravdivostní tabulku požadovaného obvodu. Délka obou seznamů se rovná n^2, kde n značí množství vstupů obvodu.

Spolu s pravdivostní tabulkou jsou algoritmu zadány omezující podmínky: množství povolených hradel a jejich vzájemných propojení a parametrů, pravděpodobnost křížení, pravděpodobnost mutace a velikost populace.

V první generaci pracuje algoritmus s několikanásobně větší populací než v generacích následujících, aby byla hned zpočátku obsažena co největší plocha prohledávacího prostoru. V základním nastavení se velikost populace v první generaci rovná 10000 .

Před samotným testem je zvolena proměnná, jejíž vztah k délce běhu algoritmu studujeme. Podle nastavení se může jednat o velikost populace, pravděpodobnost mutace nebo o pravděpodobnost křížení.

Během průběhu testu se hodnota vybrané proměnné po určeném počtu běhů algoritmu mění, zatímco hodnoty ostatních proměnných zůstávají stejné.
Intervaly hodnot, kterých mohou proměnné nabývat, jsou určeny druhem sledované proměnné a spolu se základními hodnotami proměnných jsou popsány v tabulce.

Tabulka 1 Sledované proměnné a jejich hodnoty

Proměnná	Základní nastavení	Rozpětí hodnot
Velikost populace	10	$\langle 10, 100 \rangle$
Pravděpodobnost mutace	0.01	$\langle 0.01, 0.19 \rangle$
Pravděpodobnost křížení	0	$\langle 0, 0.9 \rangle$
Počet uzlů	20	Netestováno

Množství bran poskytnutých algoritmu bylo po dobu běhu algoritmu neměnné a nepodléhalo vlivu genetických operátorů. Počet bran byl určen vzhledem k minimálním požadavkům na implementaci zadaných obvodů v NAND logice a ve snaze uvolnit podmínky nutné k přijetí řešení.

Během testování funkčnosti programu se ukázalo, že příliš nízký počet bran značně prodlužuje dobu běhu algoritmu v počtu generací nutných k získání řešení.
Se zvyšujícím se počtem bran ovšem rostly i nároky programu na systémové zdroje. Přičemž délka běhu programu v reálném čase rostla s přímou úměrou k počtu bran v kandidátských řešeních, a postupně smazala úspory získané na počtu generací.

Pro snadnější testování algoritmu byl počet bran omezen na 20; jako kompromis mezi snahou zlepšit efektivitu algoritmu a zkrátit dobu sběru dat.

4.1.6 **Výstup**

Hlavním výstupem programu jsou hodnoty uložené ve formátu csv.
Zde se na jednotlivých řádcích nacházejí, čárkami oddělené hodnoty, získané při stejném nastavení algoritmu.

Podle druhu vybraného výstupu, mohou hodnoty v souboru reprezentovat počet generací nutných k získání optimálního řešení, případně časový údaj v sekundách, indikující délku běhu algoritmu, než bylo optimální řešení nalezeno.

V druhém jmenovaném případě je ovšem výstup do značné míry závislý na vnějších podmínkách; především na zátěži počítače jinými procesy.
Z důvodu obtížné analýzy, způsobené nekonstantní zátěží systému a nízké vypovídací hodnotě časových dat, vyplývající z podstaty experimentu a využité platformy, byl namísto časových hodnot ke sběru dat použit výstup indikující počet generací, který je nezávislý na výpočetní zátěži počítače.
Formát csv byl zvolen především díky své jednoduchosti a snadnému převedení získaných dat do jiných aplikací.
Výsledné tabulky jsou, po minimálních úpravách, použity v programu Excel 2016 jako vstup statistických funkcí.

Druhým výstupem programu je soubor, popřípadě více souborů, formátu svg, které reprezentují fenotyp optimálního řešení, převedený do podoby zjednodušeného schématu elektronického obvodu.
Vzhledem k časové náročnosti analýzy a značné redundanci těchto dat, je svg výstup programu v základním nastavení vypnutý.

Schémata ze svg výstupu programu byla použita k potvrzení funkčnosti a optimality řešení získaných algoritmem. Pro přesné zjištění vlastností řešení, byla náhodně vybraná schémata manuálně převedena do programů Logisim a CedarLogic, kde byla ověřena jejich funkčnost.

4.2 Zpracování a analýza dat

Výstup programu ve formátu csv byl zpracován v tabulkovém procesoru Excel 2016 a v prostředí programovacího jazyka R.
Z dat byl v programu Excel získán: průměr, medián, vypočítány kvartily, Spearmanův korelační koeficient a rozptyl.
Ze získaných hodnot byly vypracovány předběžné boxploty a grafy rozdělení naměřených hodnot k potvrzení validity získaných dat a předběžné analýze fungování algoritmu.

Data byla následně převedena do jednotného formátu, ze kterého byly v prostředí R vytvořeny boxploty a získány hodnoty použité v samotné práci.

5 Výsledky a diskuse

Program byl spuštěn na osobním počítači s operačním systémem Windows 10 Home verze 15063, v jeho 64bitové variantě, a dostupnými systémovými prostředky sestávajícími z 8 GB RAM paměti a čtyřjádrového procesoru Intel i5-750.

Dostupná paměť počítače se ukázala být pro potřeby programu zcela dostačující. Během chodu program konstantně využíval pouze 5.9 MB paměti.

V důsledku jednovláknové podstaty aplikace bylo programem využito pouze jedno z jader procesoru. Z hlediska výkonu programu byl procesor limitujícím faktorem.
I v nejjednodušším nastavení algoritmu s minimální velikostí populace, bylo jádro využívané programem zcela vytíženo.

Pro každou kombinaci zadané úlohy, měřeného vztahu a hodnoty sledovaného parametru byl program spuštěn 150krát.
Rozpětí hodnot testovaných parametrů, stejně jako počet spuštění programu, byla určena vzhledem k omezeným systémovým prostředkům, v jejichž důsledku se běh programu neúnosně prodlužoval pro hodnoty mimo testovaná rozpětí.

Zvláště vyšší hodnoty pravděpodobnosti mutace se ukázaly jako problematické, v důsledku náhodné a v závislosti na poloze genu v chromozomu radikální, podstaty vlivu operátoru na výsledný fenotyp a fitness kandidátského řešení.

Tabulka 2 Nastavení programu při sběru dat

Parametr	Základní nastavení	Rozpětí hodnot
Velikost populace v první generaci	10000	
Velikost populace	10	$\langle 10, 100 \rangle$
Pravděpodobnost mutace	0.01	$\langle 0.01, 0.19 \rangle$
Pravděpodobnost křížení	0	$\langle 0, 0.9 \rangle$
Počet uzlů	20	Netestováno
Počet vstupů uzlu	2	Netestováno
Algoritmus selekce	Ruleta	
Křížení	Jednobodové	
Mutace	Jednobodová	
Typ hodnoty fitness	Hrubá	$\langle 0, 1 \rangle$
Maximální počet generací	Neomezený	
Požadovaná hodnota fitness	1	$\langle 0, 1 \rangle$

Ve všech nastaveních bylo hledání řešení neomezené počtem generací, v zájmu získání co nejširšího rozpětí hodnot.

Vzhledem k jednoduchosti zadaných úloh a zvoleným hodnotám parametrů, nehrozila možnost neexistence hledaného řešení. V opačném případě by nepřítomnost omezení mohla vést k selhání algoritmu a omezení by bylo nutno zavést.

5.1 Výsledky experimentů

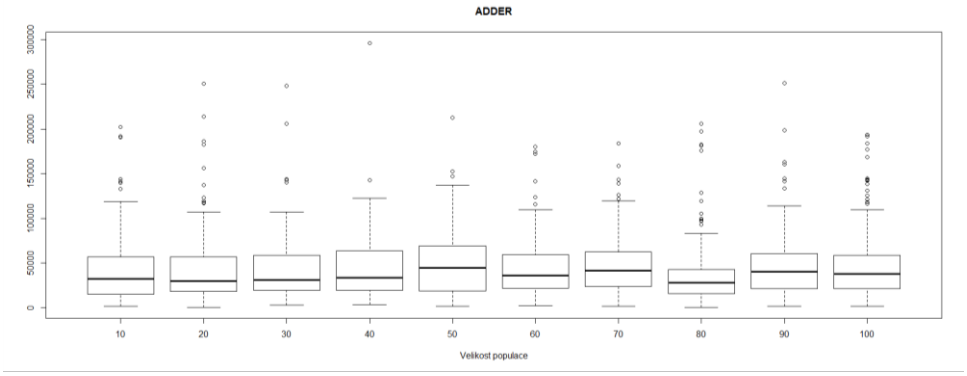
Efektivita zvoleného nastavení parametrů algoritmu byla hodnocena mediánem a extrémy počtu generací nutných k získání řešení.

Náhodně vybraná optimální řešení z množiny všech optimálních řešení poskytnutých algoritmem, byla simulována v programech Logisim a CedarLogic, pro ověření jejich vlastností a potvrzení správného fungování funkce fitness.

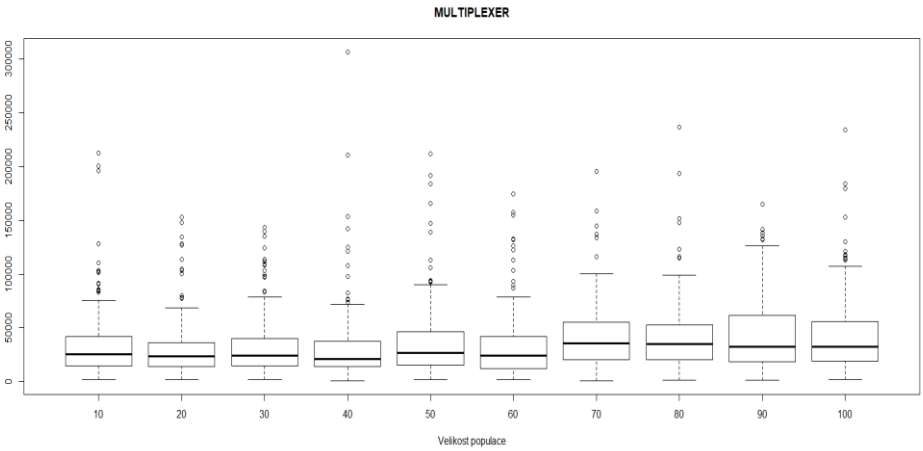
Ve všech zadaných úlohách se projevil silný vztah mezi pravděpodobností mutace $P(m)$ a efektivitou algoritmu, přičemž hodnoty $P(m) \geq 0.18$ vedly ke značnému zvýšení počtu generací nutných k zpracování úlohy.

5.1.1 Vztah velikosti populace a délky zpracování

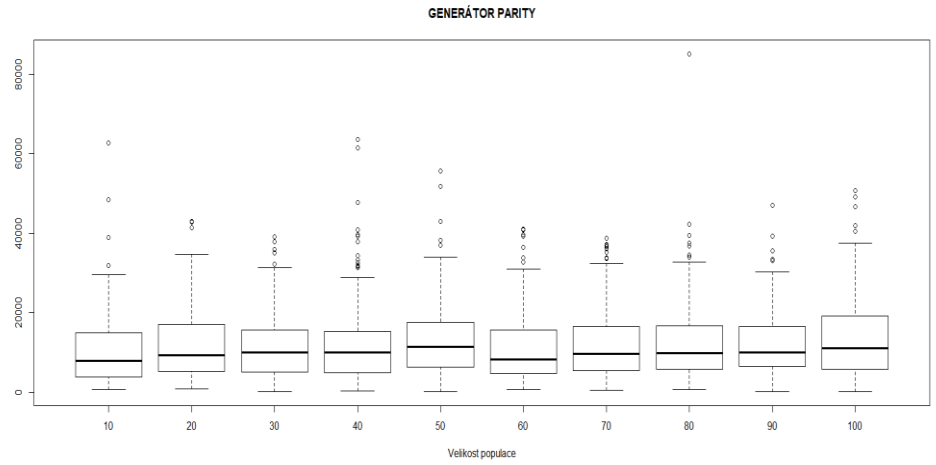
Vliv velikosti populace na výkonnost algoritmu se ve všech zadaných úlohách ukázal jako měřitelný, ale téměř zanedbatelný oproti vlivům ostatních konstant.
Medián uplynulých generací byl při maximální velikosti populace o tisíce generací vyšší než u velikosti minimální.



Graf 1 Vliv velikosti populace na dobu vytvoření sčítačky



Graf 2 Vliv velikosti populace na dobu vytvoření multiplexoru



Graf 3 Vliv velikosti populace na dobu vytvoření generátoru parity

U všech zadaných úloh byl mezi velikostí populace a mediánem uplynulých generací, naměřen kladný Spearmanův korelační koeficient v rozpětí $\langle 0.4061, 0.697 \rangle$. Koeficient mediánu uplynulých generací a velikosti populace byl znatelně nižší než u vztahů mezi mediánem a ostatními sledovanými konstantami.

Pro vztah mezi maximálním počtem uplynulých generací a velikostí populace byl naměřen koeficient v hodnotách $\langle -0.2485, 0.2364 \rangle$ a Spearmanův koeficient minima generací a velikosti populace byl $\langle -0.3818, 0.0061 \rangle$.
Záporné hodnoty korelačního koeficientu byly naměřeny pouze pro vztahy mezi velikostí populace, minimem a maximem počtu generací nutných k získání řešení v daném nastavení algoritmu.

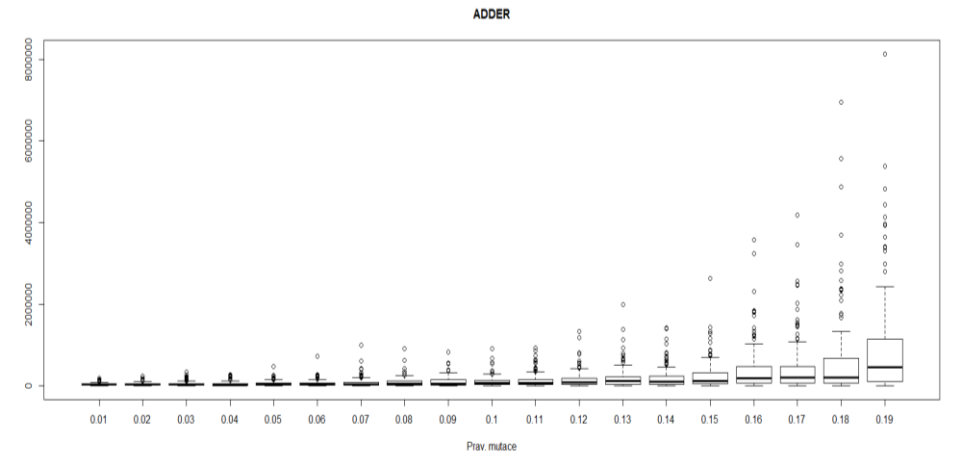
Důvodem slabší korelace mezi velikostí populace mediánem generací může být, že na rozdíl od zbytku sledovaných proměnných, se velikost populace neprojevuje na fungování algoritmu přímo, a její vliv je jemnější, než vliv pravděpodobností mutace a křížení.

5.1.2 Vztah pravděpodobnosti mutace a délky zpracování

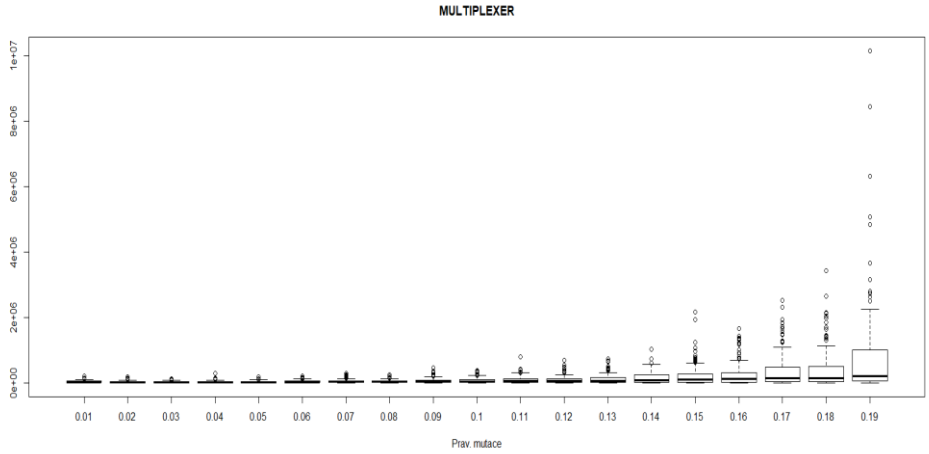
U všech zadaných úloh se vztah pravděpodobnosti mutace a délky běhu algoritmu ukázal jako znatelně silnější než ostatní sledované vztahy.
Především u hodnot $Pm \geq 0.1$ docházelo k rapidnímu nárůstu počtu generací nutných k získání optimálního řešení. V extrémních případech dosahovala délka doby zpracování ≥ 10000000 generací pro $Pm = 0.19$.

S rostoucí pravděpodobností mutace také rostl rozdíl mezi minimální a maximální dobou zpracování úlohy; i tento trend dosahoval svého extrému při $Pm = 0.19$. Oba jmenované jevy měly zásadní vliv na délku zpracování úlohy v reálném čase. Především z tohoto důvodu bylo testované rozpětí hodnot pravděpodobnosti mutace omezeno na $\langle 0.01, 0.19 \rangle$.

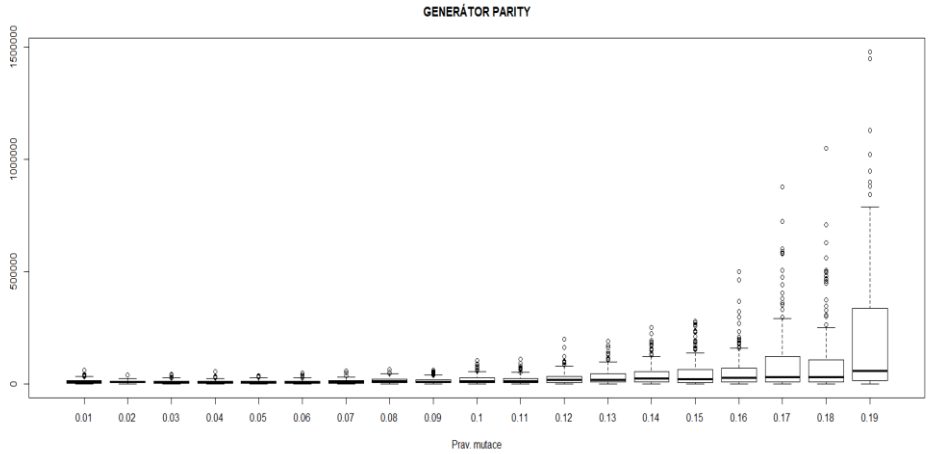
Korelace mezi pravděpodobností mutace a mediánem počtu generací byla znatelně silnější než u ostatních sledovaných proměnných, a naměřený Spearmanův koeficient se pohyboval v rozpětí $\langle 0.93508, 0.9789 \rangle$.



Graf 4 Vliv pravděpodobnosti mutace na dobu vytvoření sčítačky



Graf 5 Vliv pravděpodobnosti mutace na dobu vytvoření multiplexoru



Graf 6 Vliv pravděpodobnosti mutace na dobu vytvoření generátoru parity

Korelace byla také zjištěna mezi pravděpodobností mutace a extrémy naměřených hodnot, ačkoliv se nejednalo o vztah stejné síly.

Spearmanův koeficient vztahu maximálního počtu uplynulých generací a pravděpodobnosti mutace nabýval hodnot $\langle 0.636, 0.8704 \rangle$.

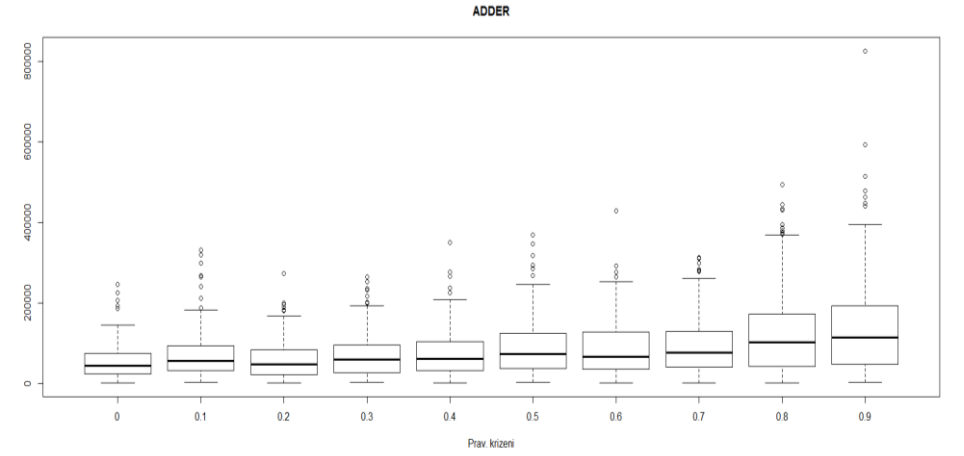
Nejslabší z naměřených vztahů byl mezi minimálním počtem generací a pravděpodobností mutace s hodnotami koeficientu v rozpětí $\langle 0.0192, 0.4456 \rangle$.

5.1.3 Vztah pravděpodobnosti křížení a délky zpracování

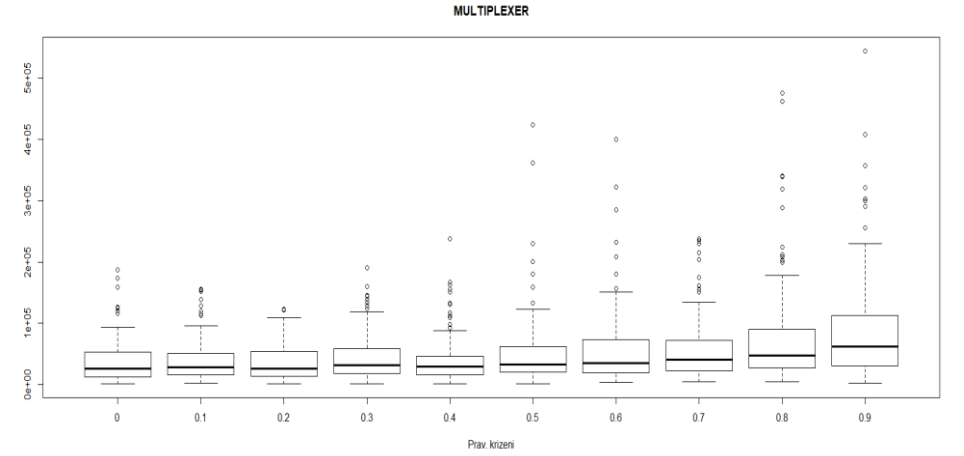
Pravděpodobnost křížení, nehledě na druh zadané úlohy, měla značný vliv na délku doby zpracování zadání. Medián počtu uplynulých generací při $Pc = 0.9$ v průměru převyšoval medián naměřený v nastavení $Pc = 0$ o 65953 generací.

Hodnoty Spearmanova korelačního koeficientu mediánu naměřených hodnot a pravděpodobnosti křížení spadaly do rozpětí $\langle 0.9273, 0.9758 \rangle$.

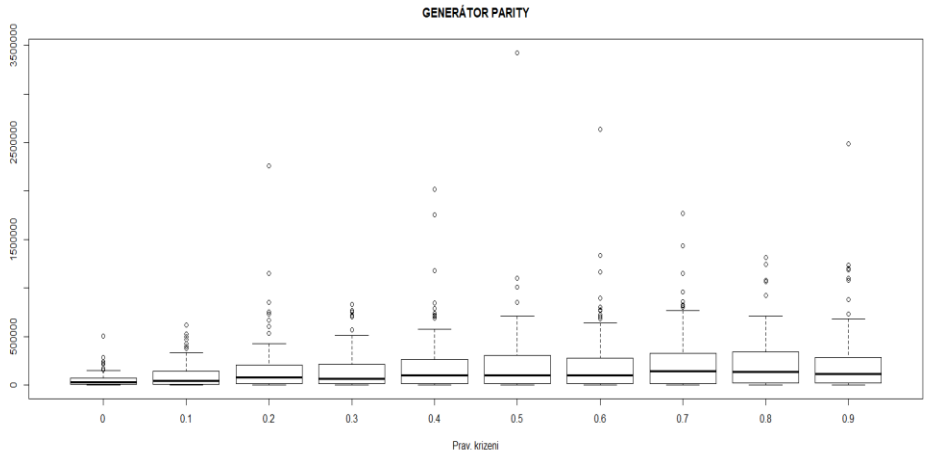
Vliv křížení na výkon algoritmu byl zvláště výrazný pro $Pc \geq 0.4$.



Graf 7 Vliv pravděpodobnosti křížení na dobu vytvoření sčítačky



Graf 8 Vliv pravděpodobnosti křížení na dobu vytvoření multiplexoru



Graf 9 Vliv pravděpodobnosti křížení na dobu vytvoření generátoru parity

Korelační koeficienty vztahu mezi minimem generací a pravděpodobností křížení se nacházely v rozpětí $\langle 0.1532, 0.7091 \rangle$ a koeficienty naměřené pro vztah maximálního počtu generací a pravděpodobnosti křížení spadaly do rozsahu $\langle 0.5394, 0.8667 \rangle$.

5.2 Shrnutí výsledků

Vliv pravděpodobnosti mutace na výkon algoritmu se ukázal jako nejsilnější ze všech sledovaných vztahů. Vyšší hodnoty pravděpodobnosti mutace se ukázaly mít zásadní vliv na délku zpracování úlohy. Medián počtu generací potřebných k získání řešení dosahoval pro $Pm = 0.19$ až desetinásobku mediánu hodnot naměřených při nastavení $Pm \leq 0.04$. Přičemž maxima naměřených hodnot v nastavení $Pm = 0.19$ dosahovala více než 40násobku oproti hodnotám v nastavení $Pm = 0.01$.

Zvyšující se pravděpodobnost křížení se ukázala mít slabší, ale stále měřitelný vliv na výkon algoritmu. Srovnání výsledků experimentů, provedených v extrémech rozmezí sledovaných hodnot, ukazuje negativní vliv vysoké pravděpodobnosti křížení.

Mediány hodnot naměřených v nastaveních $Pc = 0.1$ a $Pc = 0.9$ se liší řádově o desetitisíce generací, přičemž rozdíly mezi maximálními a minimálními naměřenými hodnotami v krajních nastaveních jsou srovnatelné.

Nalezený vztah mezi výkonem algoritmu a pravděpodobností křížení je podobný vztahu mezi stejnými proměnnými, který byl pozorován u kartézského genetického programování.

Velikost populace měla ve srovnání s pravděpodobností mutace a křížení, zanedbatelný vliv na výkon algoritmu.
Ačkoliv korelace byla u všech zadaných úloh měřitelná, byl rozsah hodnot korelačního koeficientu větší než pro ostatní sledované proměnné.
Dalším zajímavým údajem byl koeficient korelace minimálních naměřených hodnot s velikostí populace, který nabýval hodnot převážně záporných, zatímco hodnoty korelačního koeficientu maximálního počtu generací a velikosti populace se pohybovaly v rozmezí $\langle -0.2485, 0.2364 \rangle$.

Podstata vlivu velikosti populace na výkon algoritmu tedy v důsledku naměřených hodnot není jednoznačná a může být závislá na hodnotách ostatních proměnných více, než vliv pravděpodobnosti mutace a křížení.

Vzhledem k povaze provedených experimentů ale není vyloučeno, že jiné kombinace hodnot proměnných mohou podstatně zvýšit výkon algoritmu.
Popřípadě mohou zcela změnit povahu zjištěných vztahů mezi proměnnými a výkonem algoritmu.

Zkoumání vlivu kombinací hodnot proměnných na výkon genetického algoritmu ale není předmětem této práce.

6 Závěr

Cílem práce bylo navrhnout a implementovat fitness funkci genetického algoritmu určeného k návrhu kombinačních logických obvodů a následně zjistit závislost mezi délkou doby zpracování zadané úlohy a nastavením algoritmu.

Algoritmus byl implementován v jazyce C#.
Spolu s algoritmem byl v jazyce C# vytvořen program poskytující uživatelské rozhraní a grafickou reprezentaci řešení vytvořeného algoritmem.

Efektivita jednotlivých nastavení konstant algoritmu byla otestována třemi zadanými úlohami: jednobitovou sčítačkou, multiplexorem a generátorem parity.

Ze sledovaných konstant měla na délku doby zpracování největší vliv pravděpodobnost mutace.
Ačkoli byl vliv pravděpodobnosti křížení a velikosti populace na běh algoritmu měřitelný, nedosahoval Spearmanův korelační koeficient mezi nimi a délkou běhu algoritmu, hodnotě vypočítané pro korelaci s pravděpodobností mutace.

Kladný koeficient korelace byl naměřen i pro vztah mezi maximálním a minimálním počtem generací a každou sledovanou konstantou, s výjimkou velikosti populace.

Velikost populace kandidátských řešení měla na efektivitu algoritmu méně jasný vliv než ostatní sledované konstanty.
Korelace s mediánem uplynulých generací byla znatelně slabší než u ostatních vztahů, a rozdíl hodnot mediánu počtu generací mezi nejmenší a největší povolenou populací nepřekonal ani u jednoho ze zadání hodnotu 3σ.
Korelační koeficient velikosti populace a minimálního počtu generací spadal do rozsahu $\langle -0.3818, 0.0061 \rangle$ přičemž koeficient pro vztah mezi maximálním počtem generací a velikostí populace se pohyboval v rozsahu $\langle -0.2485, 0.2364 \rangle$.

Zjištěné hodnoty naznačují, že velikost populace ve sledovaném rozpětí má na efektivitu zkoumaného algoritmu minimální vliv. Oproti ostatním sledovaným konstantám se dokonce jedná o vliv téměř zanedbatelný.

Nicméně vzhledem k nutnosti změřit fitness celé populace, může mít množství jedinců zásadní vliv na délku běhu algoritmu v reálném čase. Především v případě, že algoritmus není optimalizován pro využití více jader nebo pokud sdílí systémové prostředky s jinými procesy.

Zmíněné obtíže nastaly během testování algoritmu.
V důsledku omezení plynoucích z dostupného hardware a vlastností přítomného OS, se čas nutný k získání optimálního řešení značně prodlužoval s velikostí populace.

Doba běhu algoritmu se také prodlužovala se zvyšující se pravděpodobností mutace a křížení, nicméně oproti zmíněnému vlivu velikosti populace byla prodloužení v reálném čase úměrná zvyšujícímu se počtu uplynulých generací.

Přes zmíněné obtíže se potvrdila schopnost genetických algoritmů navrhovat jednoduché kombinační obvody. Nasazení této techniky pro řešení složitějších úloh ale silně závisí na optimalizaci algoritmu, popřípadě na dostupných systémových prostředcích.

7 Seznam použitých zdrojů

Beyer, H. G. & Scwefel, H. P., 2002. Evolution strategies – A comprehensive introduction. *Natural Computing*, Březen, pp. 3-52.

Blicke, T. & Thiele, L., 1995. *A Comparison of Selection Schemes used in Genetic Algorithms.* [Online]
Available at: http://www.tik.ee.ethz.ch/file/6c0e384dceb283cd4301339a895b72b8/TIK-Report11.pdf
[Přístup získán 1 9 2017].

Britannica, T. E. o. E., 2016. *Phenotype.* [Online]
Available at: https://www.britannica.com/science/phenotype
[Přístup získán 1 2 2017].

Csontó, J., 2001. Evoluce s otevřeným koncem - Tierra. V: A. Baďura, editor *Umělá inteligence (3).* Praha: Academia, pp. 105-108.

Flegr, J., 2009. *Frozen Evolution.* [Online]
Available at: http://www.frozenevolution.com/iii6-mutations-can-be-differentiated-positive-negative-and-selectively-neutral-basis-their-effect-bi
[Přístup získán 1 11 2017].

Fogel, D. B., 1995. Phenotypes, Genotypes, and Operators in Evolutionary Computation. *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*, p. 1.

Karásek, J. & Crvrk, L., 2013. Stav vědy a techniky v oblasti genetického programování. *Electrorevue*, pp. 146-155.

Lažaňský, J., 2001. Evoluční programování a evoluční strategie. V: A. Baďura, editor *Umělá inteligence (3).* Praha: Academia, pp. 153-155.

Lažaňský, J., 2001. Evoluční výpočetní techniky. V: A. Baďura, editor *Umělá inteligence (3).* Praha: Academia, pp. 117-160.

Lažaňský, J., 2001. Genetické algoritmy. V: A. Baďura, editor *Umělá inteligence (3).* Praha: Academia, pp. 121-152.

Mařík, V. a další, 2001. *Umělá inteligence (3).* Umělá inteligence editor Praha: Academia.

Sekanina, L., 2009. Evoluční algoritmy. V: A. Baďura, editor *Evoluční hardware.* Praha: Academia, pp. 105-106.

Sekanina, L., 2009. *Evoluční hardware.* GERSTNER editor Praha: Academia.

Sekanina, L., 2009. Genetický algoritmus. V: A. Baďura, editor *Evoluční hardware.* Praha: Academia, pp. 112-115.

Sekanina, L., 2009. Princip evolučního algoritmu. V: A. Baďura, editor *Evoluční hardware.* Praha: Academia, pp. 108-109.

Sekanina, L., 2009. Selekční mechanismy. V: A. Baďura, editor *Evoluční hardware.* Praha: Academia, pp. 113-114.

Sekanina, L. & Vašíček, Z., 2015. Evolutionary Approach to Approximate Digital Circuits Design. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, Červen, pp. 432-444.

Taylor, P. a. L. R., 2017. *The Genotype/Phenotype Distinction.* [Online] Available at: https://plato.stanford.edu/archives/sum2017/entries/genotype-phenotype/

[Přístup získán 6 1 2018].

Wolper, D. & Macready, W., 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, Duben, 1(1), pp. 67 - 82.

8 Přílohy

8.1 Seznam grafů

8.2 Seznam tabulek

8.3 Seznam obrázků