VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ NÁVRH BOOLEOVSKÝCH FUNKCÍ PRO KRYPTOGRAFII

EVOLUTIONARY DESIGN OF BOOLEAN FUNCTIONS FOR CRYPTOGRAPHY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAN DVOŘÁK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAKUB HUSA**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Dvořák Jan**
Program: Informační technologie
Název: **Evoluční návrh booleovských funkcí pro kryptografii**
Evolutionary Design of Boolean Functions for Cryptography
Kategorie: Bezpečnost

Zadání:

1. Seznamte se s využitím booleovských funkcí v kryptografii, zaměřte se především na jejich použití při generování bezpečných pseudonáhodných posloupností pomocí LFSR.
2. Nastudujte problematiku evolučních algoritmů a genetického programování.
3. Navrhněte systém schopný evolučního návrhu booleovských funkcí s požadovanými kryptografickými vlastnostmi.
4. Navržený systém implementujte a experimentálně ověřte jeho schopnost generovat různé druhy booleovských funkcí.
5. Zhodnoťte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

1. Splnění bodů 1 až 3 zadání.

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

Vedoucí práce: **Husa Jakub, Ing.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 26. října 2018

Abstrakt

Cílem této bakalářské práce je porovnat různé selekční metody použité v kartézském genetickém programování aplikovaném na problém různých druhů kryptograficky významných booleovských funkcí. Zaměřil jsem se na tyto typy selekce: evoluční strategie $(1 + \lambda)$ a $(1, \lambda)$, turnajová selekce a selekce ruletou. Zvolený problém byl vyřešen implementací CGP se zmíněnými typy selekce a statistickým zpracováním dat získaných provedením experimentů. Vyhodnocením výsledků jsem zjistil, že nejlepších výsledků, v případě ohnutých funkcí, dosahuje evoluční strategie $(1 + \lambda)$. V případě vyvážených funkcí s vysokou nelinearitou dosáhla nejlepších výsledků selekce ruletou.

Abstract

The goal of this bachelor's thesis is to compare various selection methods used in cartesian genetic programming applied to a problem of various types of cryptographically significant boolean functions. I focused on these selection methods: evolutionary strategies $(1 + \lambda)$ and $(1, \lambda)$, tournament selection and roulette selection. The chosen problem was solved by an implementation of CGP with the above-mentioned selection methods and by a statistical evaluation of data acquired from conducted experiments. Evaluation of mentioned data has shown that the best results in case of bent functions were achieved while using $(1 + \lambda)$ evolutionary strategy. The roulette selection performed the best in case of balanced functions with high nonlinearity.

Klíčová slova

Ohnuté funkce, Vyvážené funkce s vysokou nelinearitou, Kartézské genetické programování, Evoluční strategie, Turnajový výběr, Výběr ruletou

Keywords

Bent functions, Balanced functions with high nonlinearity, Cartesian genetic programming, Evolutionary strategy, Tournament selection, Roulette selection

Citace

DVOŘÁK, Jan. *Evoluční návrh booleovských funkcí pro kryptografii.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jakub Husa

Evoluční návrh booleovských funkcí pro kryptografii

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jakuba Husy. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Dvořák
15. května 2019

Poděkování

Velice rád bych poděkoval vedoucímu práce Ing. Jakubu Husovi za ochotu, odborný přístup a cenné rady při konzultacích.

Obsah

Kapitola 1

Úvod

Šifrování komunikace je v dnešním světe naprosto nezbytné. Je to proces, kdy dochází k zabezpečení zprávy pro přenos přes nezabezpečenou linku. Šifrováním se zabývá obor kryptografie a kryptoanalýza je obor zabývající se lámáním šifer. V této práci se budeme věnovat proudovým šifrám a jejich realizaci pomocí lineárního posuvného registru se zpětnou vazbou a zabezpečení pomocí booleovských funkcí. Je důležité, aby tyto funkce měli potřebné vlastnosti, které jim poskytují ochranu proti různým útokům a zabraňují případnému prolomení šifry. K návrhu kryptografických funkcí je použit evoluční návrh založený na Darwinově teorii evoluce a přirozeného výběru.

Druhá kapitola (2) je věnována booleovským funkcím a jejich využití v kryptografii. V úvodu kapitoly je uvedena základní definice booleovských funkcí, princip fungování proudových šifer a způsob generování klíčového řetězce. Dále kapitola obsahuje popis pseudonáhodného generátoru na bázi lineárního registru se zpětnou vazbou (LFSR, viz. 2.1). Poté následuje výčet způsobů reprezentace booleovských funkcí (2.2) a podrobněji je popsána pravdivostní tabulka, algebraická normální forma (2.2.1) a Walsh-Hadamardovo spektrum (2.2.2). Na konci kapitoly jsou popsány kryptograficky významné vlastnosti funkcí, kterých se pokusíme dosáhnout při generování funkcí aplikací navrženou ve čtvrté kapitole (4).

Následující kapitola (3) je věnována evolučnímu návrhu a evolučním algoritmům, které se používají pro řešení optimalizačních úloh, tedy úloh, ve kterých hledáme nejlepší možné řešení. Nejdříve jsou uvedeny nezbytné pojmy, které se v oblasti evolučních algoritmů běžně používají. Následuje obecný popis evolučního algoritmu (3.1) a podrobnější popis jeho částí. Nakonec je popsán jeden konkrétní evoluční algoritmus a tím je kartézské genetické programování (3.2) a jeho odlišnosti oproti ostatním evolučním algoritmům.

Čtvrtá kapitola (4) obsahuje návrh systému schopného generovat booleovské funkce s požadovanými vlastnostmi pomocí kartézského genetického programování. Implementace tohoto systému je detailně popsána v další kapitole (5) a výsledky, kterých bylo dosaženo a jejich zhodnocení jsou uvedeny v šesté kapitole (6).

Kapitola 2

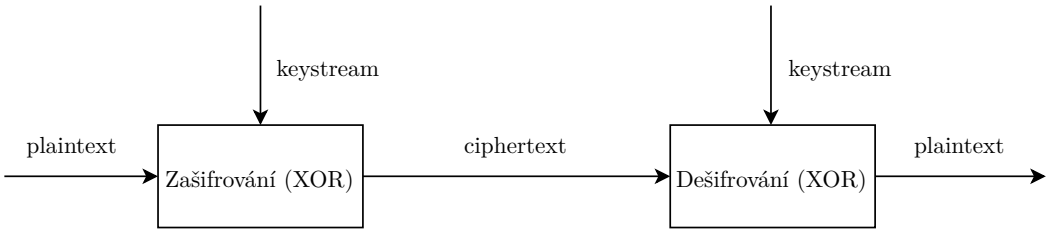
Využití booleovských funkcí v kryptografii

Booleovské funkce jsou funkce ve tvaru:

$$f : \mathbb{Z}_2^n \to \mathbb{Z}_2,$$

kde $n \in \mathbb{N}$ a $\mathbb{Z}_2$ je konečné pole o dvou prvcích $\{0, 1\}$ [14]. Tedy každá booleovská funkce f mapuje vstupní bitový vektor délky n na jedinou binární hodnotu. Skládají se z operandů, které nabývají hodnot z $\mathbb{Z}_2$ a logických operací. Mezi základní logické operace patří logický součet, logický součin a negace.

Pro aplikaci booleovských funkcí v kryptografii je důležité, aby tyto funkce měly určité kryptografické vlastnosti, které zajistí bezpečnost zprávy vůči různým druhům útoků. Z tohoto důvodu je primárním užitím kryptograficky užitečných booleovských funkcí návrh kryptografických algoritmů, zejména proudových šifer [28]. Tyto šifry jsou založeny na principu Vernamovy šifry, u které bylo dokázáno, že je zcela neprolomitelná, jsou-li dodržena následující pravidla: klíč musí být zcela náhodný, klíč může být použit maximálně jednou a klíč musí být alespoň stejně dlouhý jako šifrovaná zpráva [27]. Vernamova šifra otevřenou zprávu (*plaintext*) šifruje pomocí operace exkluzivního součtu (XOR) s privátním klíčem. Vzniká šifrovaný text (*ciphertext*), který lze opět dešifrovat kombinací (XOR) ciphertextu a klíče. Bitový řetězec privátního klíče (*keystream*) je obvykle generován pomocí generátoru



Obrázek 2.1: Princip Vernamovy šifry

pseudonáhodných čísel. Každá bitová sekvence, určená předem sdíleným klíčem, je použita jako „*one-time pad*“, tedy může být použita pouze jednou [9]. Jedním způsobem jak tento generátor implementovat je lineární posuvný registr se zpětnou vazbou (Linear-Feedback Shift Register, LFSR), který bude více popsán v sekci 2.1.

U generované posloupnosti existuje lineární závislost mezi vstupy a výstupy generátoru. Lineární závislosti jsou lehce analyzovatelné, tím pádem jsou proudové šifry navržené pouze

s LFSR, považovány za nedostatečně kryptograficky bezpečné a je snadné takové šifry prolomit. Zde přicházejí na řadu booleovské funkce s kryptografickými vlastnostmi. Booleovské funkce jsou použity k vyprodukování sekvence keystream na základě vstupů z LFSR. Díky těmto funkcím je závislost mezi plaintextem a ciphertextem složitější, než lineární a tedy poskytuje lepší zabezpečení. Konkrétně jednotlivé bity šifrovaného textu jsou získány kombinací (XOR) bitů zprávy (plaintext) a bitů keystreamu (v tomto případě výstupní hodnoty zvolené funkce). Složitost systémů používajících booleovské funkce záleží na vlastnostech zvolené funkce [14]. Konkrétní vlastnosti jsou popsány v sekci 2.3.

2.1 Linear-Feedback Shift Register

Lineární posuvný registr se zpětnou vazbou je registr, jehož stav určuje výstupní hodnotu a na základě současného stavu určuje stav následující. Jednou z hlavních výhod LFSR je snadná implementace v hardwaru i softwaru. Počáteční stav registru je dán tajným klíčem. Při každém taktu je obsah registru posunut o jednu pozici doprava a na pozici nejvíce vlevo je doplněna hodnota na základě zpětné vazby z vhodně vybraných pozic v registru. K výpočtu nové hodnoty je použita logická operace XOR. LFSR se může nacházet ve $2^n - 1$ stavech, kde n je délka registru. Pro LFSR počítající zpětnou vazbu pomocí XORu existuje jeden nepřípustný stav a tím jsou samé nuly (např. 4-bitový LFSR se nesmí nacházet ve stavu `0000`), z tohoto důvodu pouze $2^n - 1$ stavů. Tento stav je nepřípustný, protože pokud se do něho LFSR dostane, zůstane v něm zaseknutý [8].

Vzhledem ke konečnému počtu unikátních stavů LFSR se sekvence generovaná LFSR začne po dosažení určité délky periodicky opakovat. *Maximální délka sekvence* generované LFSR délky n je dána vztahem $2^n - 1$. Ne všechny sekvence generované pomocí LFSR dosahují maximální délky sekvence. LFSR lze dále specifikovat pomocí polynomů určujících pozice, které budou použity při vyhodnocování následujícího stavu. Polynom $P(x)$ určující pozice v LFSR pro nastolení následujícího stavu je dán ve tvaru uvedeném v rovnici 2.1.

$$P(x) = x^n + p_{n-1}x^n - 1 + \cdots + p_1 x + p_0, \tag{2.1}$$

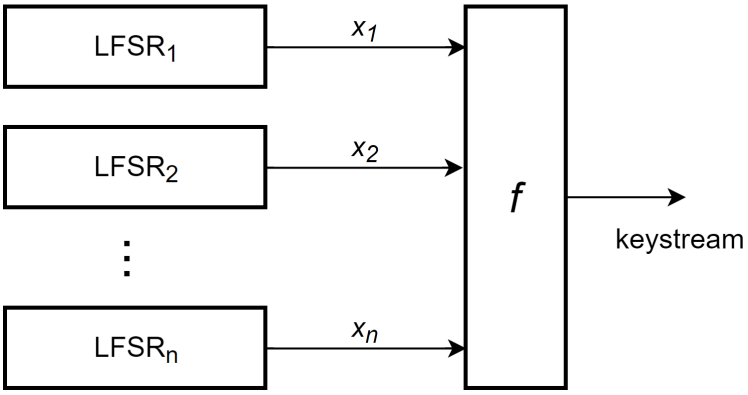
kde $p_i \in \{0, 1\}$ je *zpětnovazební koeficient*, který udává, zda bude hodnota v LFSR na pozici s indexem i použita ve zpětné vazbě (hodnota 1) či nebude (hodnota 0). LFSR generující sekvence maximální délky mají tzv. *primitivní polynomy*, to jsou polynomy, které jsou dále nedělitelné a právě tyto polynomy se používají při určení následujícího stavu [17].

Při použití v generátoru pseudonáhodných číslic se používá jeden nebo více takových registrů. Existují dva modely použití LFSR, a to filtrační (filter) a kombinační (combiner) model. Jak je znázorněno na obrázku 2.2, v kombinačním modelu je kombinován výstup několika LFSR pomocí vybrané booleovské funkce, která generuje keystream [3].

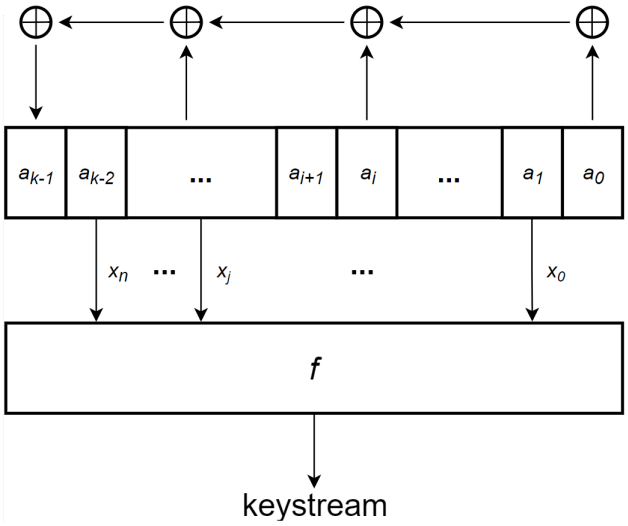
Ve filtračním modelu (obrázek 2.3) se používá jeden typicky delší LFSR a filtrační booleovská funkce o n proměnných, jejíž vstupy se nacházejí na předem daných neměnných pozicích uvnitř zmíněného LFSR [2].

2.2 Reprezentace booleovských funkcí

Booleovské funkce lze reprezentovat mnoha různými způsoby. Každá varianta poskytuje své výhody a nevýhody. Jednotlivé způsoby reprezentace jsou booleovských funkcí jsou ekvivalentní a tedy vzájemně převoditelné. Výběr způsobu reprezentace závisí na tom, k čemu bude funkce využita a jaké operace s ní budou prováděny.



Obrázek 2.2: Kombinační model [3]



Obrázek 2.3: Filtrační model

Prvním ze způsobů reprezentace, který si uvedeme, je pravdivostní tabulka. Je to tabulka, která obsahuje všechny možné kombinace vstupních proměnných a výslednou hodnotu reprezentované funkce. Na každém řádku se tedy nachází výsledná hodnota funkce a kombinace vstupních proměnných, pro které daná funkce této hodnoty nabývá. Pravdivostní tabulka pro funkci o n neznámých se skládá z 2^n řádků, tedy už u funkce o čtyřech neznámých může být značně nepřehledná. V tabulce 2.1 je uveden příklad pravdivostní tabulky.

x_1	x_2	$x_1 \oplus x_2$
1	1	0
1	0	1
0	1	1
0	0	0

Tabulka 2.1: Pravdivostní tabulka operace XOR

Druhým způsobem reprezentace booleovských funkcí jsou normální formy. Mezi základní normální formy patří disjunktivní normální forma (DNF) a konjunktivní normální forma

(CNF). Další normální formou, která je pro nás důležitá, je algebraická normální forma (ANF), které je věnována celá sekce 2.2.1.

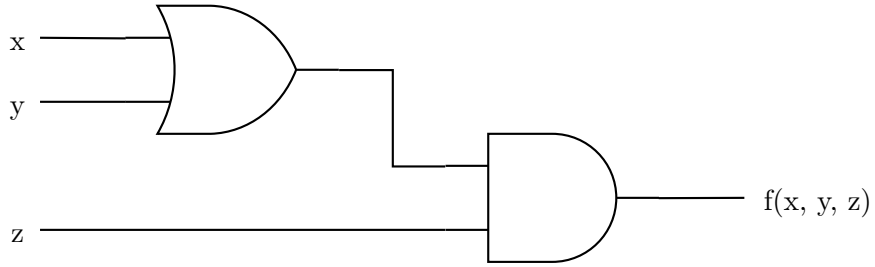
Jak je uvedeno v [16], normální forma matematického objektu je zjednodušená forma daného objektu získaná transformací, která zachová esenciální podstatu původního objektu. Logická formule je v DNF pouze tehdy, pokud je tvořena sjednoceními (jednou nebo více) průniků (negací) literálů [20]. Rovnice 2.2 je příklad rovnice, která je v disjunktní normální formě.

$$(x_0 \cap x_1) \cup (x_2 \cap \neg x_3) \tag{2.2}$$

Obdobně jako u DNF je tomu i u konjunktivní normální formy. Formule je v CNF, pokud je složena z průniků sjednocení logických literálů. Jakýkoli výraz ve výrokové logice může být transformován do CNF i DNF pomocí distributivního zákona, DeMorganových zákonů a odstraněním dvojí negace [19]. Rovnice 2.3 opět prezentuje příklad CNF.

$$(x_0 \cup x_1) \cap (x_2 \cup \neg x_3) \tag{2.3}$$

Další způsob jak reprezentovat booleovské funkce je pomocí logického obvodu. Jedná se o grafické schéma reprezentující jednotlivé operace pomocí odpovídajících hradel. Příklad reprezentace funkce logickým obvodem je k dispozici na obrázku 2.4.



Obrázek 2.4: Logický obvod znázorňující funkci $f(x, y, z) : (x \vee y) \wedge z$

Dalším způsobem zápisu logické funkce je Karnaughova mapa, pomocí které lze zobrazit n-rozměrnou logickou funkci do dvourozměrného pole. Používá se převážně pro minimalizaci logických obvodů. Na obrázku 2.5 je zobrazena Karnaughova mapa pro funkci z předešlého příkladu. Tyto dva způsoby nejsou z hlediska programování a kryptografie zcela relevantní a jsou zmíněny pouze pro úplnost [10].

z \ xy	00	01	11	10
0	0	0	0	0
1	0	1	1	1

Obrázek 2.5: Karnaughova mapa znázorňující funkci v obrázku 2.4

2.2.1 Algebraická normální forma

Algebraická normální forma je další způsob reprezentace booleovských funkcí. Z pohledu kryptografie je tato forma důležitá pro výpočet algebraického stupně, který je dán velikostí největšího termu, více informací k algebraickému stupni je uvedeno v sekci 2.3.1. Obecný tvar booleovské funkce zapsané v ANF uvádí rovnice 2.4.

$$f(x) = c_0 \bigoplus_{1 \leq i \leq n} c_i x_i \bigoplus_{1 \leq i < j \leq n} c_{ij} x_i x_j \bigoplus \cdots \bigoplus c_{1,\ldots,n} x_1 x_2 \ldots x_n, \tag{2.4}$$

kde $c_0, c_i, \ldots, c_{1,\ldots,n}$ jsou koeficienty nabývající hodnot z množiny $\{0, 1\}$. Bylo dokázáno, že každá booleovská funkce o n neznámých je převoditelná do ANF. Každá funkce v ANF je unikátní, tedy dvě funkce stejně zapsané v ANF musí vyprodukovat stejnou pravdivostní tabulku.

Podobně jako u ostatních normálních forem má i ANF svá pravidla. Všechny funkce zapsané pomocí ANF jsou tvořeny pouze operací exkluzivního součtu (XOR, značeno $\bigoplus$) a logickým součinem (AND). Přesněji funkce v ANF je tvořena XORem termů, skládajících se z násobku žádné (v takovém případě se jedná o konstantu) nebo více logických proměnných [28].

Na rozdíl od CNF a DNF se v ANF nepoužívá negace. Ačkoliv není povolena operace negace, lze hodnotu proměnné x znegovat pomocí XORu x s hodnotou logická 1, viz rovnice 2.5. S touto informací můžeme převést do ANF rovnice zapsané v CNF a DNF.

$$\neg x = x \oplus 1 \tag{2.5}$$

2.2.2 Walsh-Hadamardovo spektrum

Podle definice v [28] máme-li funkci $f(x)$ s definičním oborem $\mathbb{Z}_2^n$, jež nabývá reálných hodnot, potom Walsh-Hadamardovo spektrum funkce $f(x)$ je definováno jako:

$$F(w) = \sum_{x \in \mathbb{Z}_2^n} f(x)(-1)^{x \cdot w}, \tag{2.6}$$

kde x a w jsou binární vektory o n prvcích a $x \cdot w = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$ je jejich skalární součin. V tomto případě je výhodnější reprezentovat booleovské hodnoty hodnotami množiny $\{-1, 1\}$. Pro mapování klasických booleovských hodnot $\{0, 1\}$ na $\{-1, 1\}$ se používá vztah:

$$\delta(f(x)) = -1^{f(x)}. \tag{2.7}$$

Po nahrazení $f(x)$ v rovnici 2.6 za $\delta(f(x))$ z rovnice 2.7 dostaneme konečný vztah:

$$F(w) = \sum_{x \in \mathbb{Z}_2^n} \delta(f(x))(-1)^{x \cdot w} = \sum_{x \in \mathbb{Z}_2^n} (-1)^{f(x) + x \cdot w}. \tag{2.8}$$

Walsh-Hadamardovo spektrum se v oblasti kryptografických funkcí používá pro výpočet hodnoty nelinearity booleovské funkce. Nelinearita funkce je dána Hammingovou vzdáleností její pravdivostní tabulky od pravdivostní tabulky nejbližší afinní funkce. Samotný výpočet lze provést pomocí rychlé Walsh-Hadamardovy transformace (*Fast Walsh-Hadamarad Transform, FWHT*). FWHT je algoritmus na výpočet Walsh-Hadamardova spektra, který využívá rekurzivní definice Hadamardovy matice. FWHT rekurzivně rozděluje Walsh-Hadamardovo spektrum na poloviny a tím snižuje časovou náročnost z kvadratické ($O(n^2)$, kde n je počet prvků Walsh-Hadamardova spektra) na lineárně logaritmickou $O(n \log n)$. Nelinearita je podrobněji popsána v sekci 2.3.2.

2.3 Vlastnosti booleovských funkcí užitečné pro kryptografii

Booleovské funkce jsou používány jako významné kryptografické minimum (*Cryptographic primitive*), které slouží k zabezpečení lineární sekvence generované pomocí lineárních generátorů (viz 2.1). Jednou z výhod LFSR je, že pracuje rychle a lze jej snadno implementovat v hardware i software. Naopak slabinou generátoru LFSR je, že generované stavy jsou na sobě lineárně závislé. Toho může útočník zneužít a pomocí *Berlekamp-Masseyho algoritmu* zrekonstruovat původní nastavení generátoru a odhalit tím tajný klíč [4]. Booleovské funkce poskytují prostředky pro skrytí těchto závislostí, a tedy zabezpečení šifry. V kryptografii nemohou být použity ledajaké funkce, ale musí mít určité vlastnosti, které poskytují ochranu vůči různým druhům útoků. Mezi základní vlastnosti kryptograficky významných funkcí patří:

- Algebraický stupeň (Algebraic degree)
- Nelinearita (Nonlinearity)
- Vyváženost (Balancedness)
- Korelační imunita a odolnost (Correlation immunity and resiliency)

Mezi další vlastnosti patří algebraická imunita (algebraic immunity), rychlá algebraická imunita (fast algebraic immunity), lineární struktura (linear structure) a propagační kritérium (propagation criterion). Tyto vlastnosti jsou uvedeny pouze pro úplnost a nebudeme se jimi v této práci dále zabývat [2].

Najít funkce s optimální mírou všech vlastností je nemožné, jelikož některé vlastnosti jsou závislé na ostatních a dochází mezi nimi ke kompromisu. Příkladem takového kompromisu je algebraický stupeň (2.3.1) a korelační imunita (2.3.4).

2.3.1 Algebraický stupeň

Jak bylo uvedeno v sekci 2.2.1, funkce zapsaná v algebraické normální formě se skládá z exkluzivních součtů termů. Každý term je součin konstanty a žádné nebo více logických proměnných, nejvíce však n pro funkci n-proměnných. Každý term funkce má algebraický stupeň, který je dán počtem vstupních proměnných obsažených v daném termu. Algebraický stupeň funkce je roven nejvyššímu algebraickému stupni termu obsaženého ve funkci, kterému nepředchází konstanta rovna $log.0$. Algebraický stupeň funkce f je značen jako $deg(f)$. Například algebraický stupeň funkce 2.9 je dán velikostí posledního termu, tedy platí, že $deg(f) = 3$.

$$f(x_1, x_2, x_3) = x_1 \oplus x_1x_3 \oplus x_1x_2x_3 \tag{2.9}$$

Hodnota $deg(f)$, kde f je funkce o n-neznámých, může dosahovat nejvýše hodnoty n. Pokud existují takové konstanty $a_0, a_1, \ldots a_n \in GF(2)$ pro:

$$f(x) = a_0 \oplus a_1x_1 \oplus \cdots \oplus a_nx_n, \tag{2.10}$$

potom se taková funkce nazývá *afinní*. Z rovnice 2.10 je vidět, že algebraický stupeň nabývá hodnoty maximálně 1, tedy pro afinní funkce platí vztah $deg(f) \leq 1$. Pokud je funkce afinní a navíc se koeficient $a_0 = 0$, je tato funkce také lineární [28].

Vyšší hodnota algebraického stupně poskytuje vyšší lineární složitost produkovaného klíče, tato vlastnost poskytuje vyšší odolnost proti algoritmu Berlekamp–Massey [13]. Bylo

dokázáno, že počet vstupních proměnných funkce n, hodnota algebraického stupně deg a korelační imunita CI (o ní více v 2.3.4) jsou vzájemně protichůdné. Tedy dochází ke kompromisu mezi algebraickým stupněm a korelační imunitou. Jejich vztah je dán rovnicí $CI + deg \leq n$, pokud je navíc funkce vyvážená (sekce 2.3.3), poté platí $CI + deg \leq n - 1$. Tomuto vztahu se říká Siegenthalerova nerovnost a byla dokázána T. Siegenthalerem v [26]. Jelikož dochází ke kompromisu nemůže u kryptograficky významných funkcí ani jedna vlastnost nabývat optimální hodnoty, přesto by neměla hodnota algebraického stupně klesnout pod hranici $\frac{n}{2}$ [28].

2.3.2 Nelinearita

Jak již bylo zmíněno výše, z bezpečnostních důvodů by pseudonáhodné sekvence generované pomocí LFSR neměly být použity v šifrování přímo. Místo toho je výstup LFSR předán zvolené nelineární filtrační nebo kombinační funkci. Nelinearita booleovské funkce f o n proměnných, značená $nl(f)$, je definována jako Hammingova vzdálenost funkce f, od nejbližší afinní funkce o n-neznámých. Nejbližší afinní funkcí je myšlena funkce z množiny všech afinních funkcí o n proměnných, která má nejnižší Hammingovu vzdálenost od f [24]. Hammingova vzdálenost je určena počtem pozic, na kterých se dva řetězce (pravdivostní tabulky) liší. Nelinearitu tedy lze vyjádřit jako:

$$nl(f) = \min_{l(x) \in A_n} d(f(x), l(x)), \tag{2.11}$$

kde A_n je množina všech afinních funkcí o n vstupech a $d(x, y)$ udává Hammingovu vzdálenost mezi x a y. Nelinearitu také lze vyjádřit pomocí Walsh-Hadamardova spektra:

$$nl(f) = 2^{n-1} - \frac{1}{2} \max_{w \in \mathbb{Z}_2^n} |F(w)|. \tag{2.12}$$

Míra nelinearity pro funkce se sudým počtem proměnných je shora ohraničená hodnotou $2^{n-1} - 2^{n/2-1}$. Maximální nelinearita funkcí s počtem proměnných $n = 1, 3, 5, 7$ je dána vztahem $nl_{max} = 2^{n-1} - 2^{\frac{n-1}{2}}$ [2]. U funkcí s lichým počtem proměnných můžeme říci, že pro maximální nelinearitu funkce s více než sedmi proměnnými platí vztah: $nl_{max} \geq 2^{n-1} - 2^{\frac{n-1}{2}}$. Funkce se sudým počtem proměnných dosahující maximální hodnoty nelinearity jsou nazývány *ohnuté funkce*. Funkce s lichým počtem proměnných ohnuté být nemohou. Ohnuté funkce jako takové nejsou vhodné pro použití při šifrování z důvodu jejich nevyváženosti. Vysoká nelinearita je jedna z nejdůležitějších vlastností kryptograficky významných booleovských funkcí. V případě proudových šifer je vysoká nelinearita významná pro obranu vůči *rychlým korelačním útokům* a *lineárním útokům* [14].

2.3.3 Vyváženost

Funkce je vyvážená právě tehdy, když její pravdivostní tabulka je tvořena shodným počtem jedniček a nul. To znamená, že platí:

$$wt(f) = 2^{n-1},$$

kde $wt(f)$ je Hammingova váha funkce f a n je počet vstupních proměnných funkce f. Hammingova váha udává počet znaků řetězce, jež nabývají jiné hodnoty, než nula. Vyváženost funkce je ještě znázorněna v tabulce 2.2, kde funkce f je dána rovnicí $x_1 \oplus x_2$ a funkce g rovnicí $x_1 \wedge x_2$. Z tabulky 2.2 je vidět, že funkce f je vyvážená, kdežto funkce g nikoli.

x_1	x_2	$f(x_1, x_2)$	$g(x_1, x_2)$
1	1	0	1
1	0	1	0
0	1	1	0
0	0	0	0

Tabulka 2.2: Pravdivostní tabulka funkcí f a g

Při použití booleovských funkcí je jejich vyváženost považována za základní vlastnost, protože pokud by funkce generovala jednu hodnotu častěji než druhou, vznikla by statistická závislost mezi vstupní zprávou (plaintext) a výstupní šifrovanou sekvencí (ciphertext). Taková závislost vytváří příležitost pro útočníka použít k prolomení šifry tzv. *distinguishing attack*, pomocí kterého je útočník schopný rozlišit pseudonáhodnou sekvenci od náhodné [2].

2.3.4 Korelační imunita a odolnost

Funkce f má korelační imunitu CI stupně t, pokud její výstupy jsou statisticky nezávislé na kombinaci jejích t vstupních proměnných. Pokud rozdělíme pravdivostní tabulku na dvě poloviny podle vstupů jedné proměnné a obě poloviny obsahují stejný počet jedniček, tak má funkce f korelační imunitu $CI(f) = 1$. Pro korelační imunitu druhé úrovně rozdělíme každou polovinu tabulky opět na polovinu podle druhé proměnné (původní tabulku na čtvrtiny). Pokud opět každá část obsahuje stejný počet jedniček má korelační imunitu druhé úrovně. Stejným způsobem lze pokračovat pro další úrovně. Je zřejmé, že nejvyšší možná úroveň korelační imunity pro funkce n proměnných je právě n. Této hodnoty dosahují pouze funkce, jejichž pravdivostní tabulka se skládá ze samých jedniček nebo nul. Takové funkce jsou zjevně kryptograficky nezajímavé.

Jak již bylo zmíněno v 2.3.1, korelační imunita je závislá na hodnotě algebraického stupně a dochází mezi těmito hodnotami ke kompromisu. Pokud je funkce vyvážená a má korelační imunitu stupně t, je nazývána t-odolná. Vysoká korelační imunita poskytuje ochranu proti korelačním útokům. Obvykle je ve filtračním modelu dostačující korelační imunita první úrovně [21]. Kombinační funkce má korelační imunitu n-tého řádu, pokud je sekvence generovaných znaků pomocí m LFSR nezávislá na n vstupech [26]. Z toho vyplývá, že korelační imunita kombinační funkce musí odpovídat počtu kombinovaných LFSR.

Kapitola 3

Evoluční návrh a genetické programování

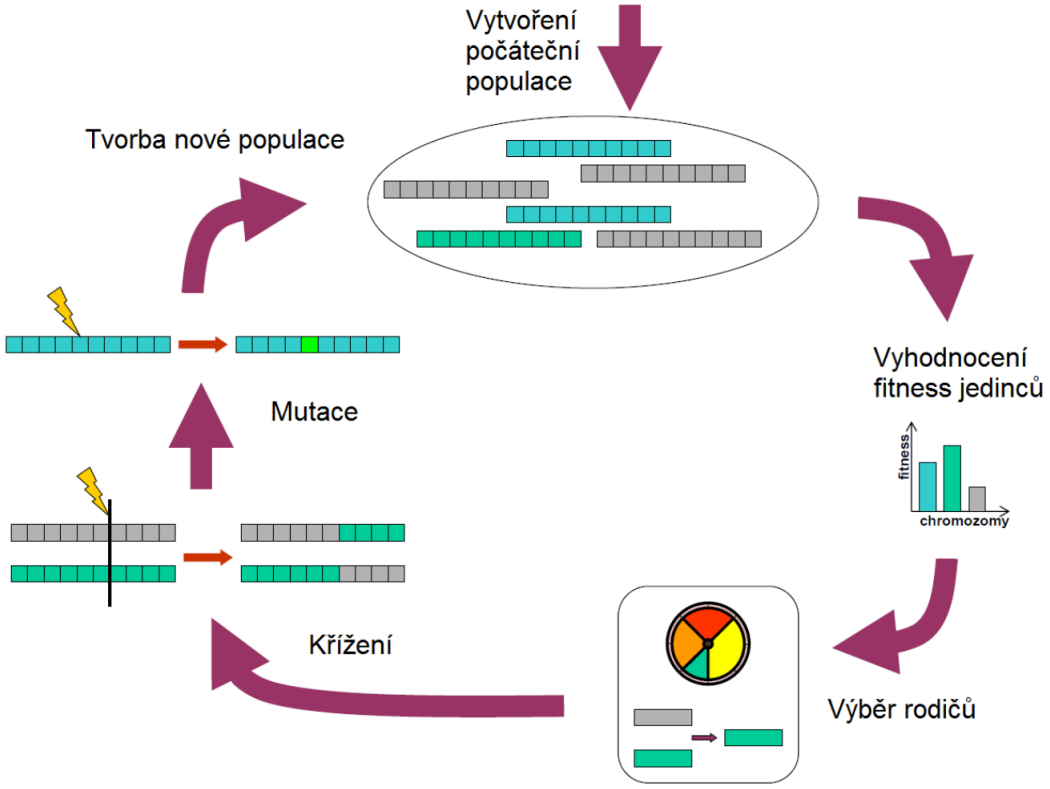
Evoluční algoritmy patří mezi algoritmy inspirované přírodou, konkrétně vývojem živočichů a jejich schopnosti přežít a rozmnožit se. Tento proces popsal Charles Darwin ve své teorii evoluce a přirozeného výběru [7]. Evoluční algoritmy jsou nedeterministické prohledávací algoritmy, které k prohledávání stavového prostoru používají heuristiku. První pokusy o modelování evoluce se datují do 60. a 70. let 20. století. V 90. letech byly nezávisle na sobě vyvinuty různé přístupy, které jsou dnes souhrnně nazývány *evolučními algoritmy*. Mezi evoluční algoritmy patří genetické algoritmy, evoluční strategie, evoluční programování a genetické programování [18]. V rámci evolučních algoritmů se používá řada termínů převzatých z biologie [1].

- **Jedinec, fenotyp** v evolučních algoritmech představuje možné řešení daného problému.
- **Chromozom, genotyp** je struktura představující konkrétní řešení. Reprezentace chromozomů se liší podle řešeného problému, může mít podobu řetězce znaků, grafu, matice nebo dalších.
- **Gen** je základní jednotka chromozomu, jeho hodnota se nazývá *alela* a jeho pozice v chromozomu *lokus*.

3.1 Evoluční algoritmus

Evoluční algoritmus se skládá z inicializace počáteční populace a cyklické evoluce nových populací, dokud nepřekročíme podmínku ukončení evoluce nebo dokud nenajdeme dostatečně dobré řešení. Inicializace počáteční populace probíhá náhodným generováním chromozomů nebo může být inicializována pomocí nějakých stávajících řešení. V momentě, kdy máme počáteční populaci, ohodnotíme každého jedince. Tomuto procesu se říká *evaluace* populace. Fitness se určuje pomocí fitness funkce (viz 3.1.2). Zvolení nevhodné fitness funkce může mít velký dopad na nalezení optimálního řešení. Dalším krokem je výběr jedinců, kteří se stanou rodiči a vytvoří potomky, tento proces se nazývá *selekce* a je mu věnována podsekce 3.1.3. Ze zvolených rodičovských chromozomů se pomocí operace *křížení*, vytvoří potomci. Následně se náhodně vyberou jedinci, nad kterými bude provedena operace *mutace* (viz 3.1.5). Po vytvoření potomků a zmutování některých jedinců, je vytvořena nová

populace. Tento proces se opakuje dokud dokud nedojde ke splnění ukončujících podmínek (viz 3.1.7). Evoluční cyklus je zobrazen na obrázku 3.1.



Obrázek 3.1: Evoluční cyklus [25]

Následující algoritmus (1) obecně popisuje evoluční algoritmus uvedený výše [15]. Všechny jeho části budou dále podrobněji popsány v této kapitole.

Algorithm 1: Evoluční cyklus

Result: Nejlepší nebo optimální řešení problému
1 Inicializuj počáteční populaci;
2 **repeat**
3 Urči fitness všech jedinců v populaci;
4 Vyber vhodné jedince pro rozmnožení;
5 Proveď rozmnožení;
6 Proveď mutaci;
7 Sestav novou populaci;
8 **until** *ukončující podmínka*;

3.1.1 Populace

Populaci tvoří předem dané množství jedinců. Počáteční populace je generována náhodně, avšak může být použita vhodná heuristika pro vygenerování kvalitnější počáteční populace. Populace se může skládat ze dvou nebo až několika tisíc jedinců. Jak ukazují výsledky v [22], různé evoluční algoritmy dokáží efektivně pracovat s různými velikostmi populace. V každém cyklu evolučního algoritmu se vytváří nové generace populace.

3.1.2 Fitness

Určování fitness populace, tj. všech jedinců populace, je stěžejním procesem celého vývoje. Hodnota fitness udává kvalitu jedince a jakou má jedinec pravděpodobnost, že přežije dostatečně dlouho na to, aby se mohl rozmnožit. Fitness jedince je explicitně určena nějakou vhodně navrženou funkcí. Taková funkce se nazývá *fitness funkce*. Existují čtyři druhy fitness [12]:

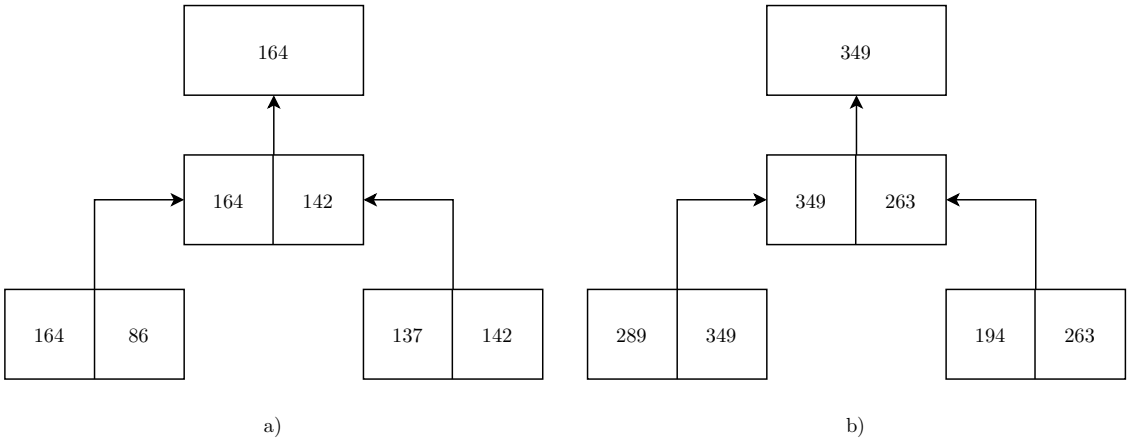
- **Hrubá (raw)** udává fitness ve formě, která se přímo vztahuje k řešenému problému. Například při hledání nejkratší cesty z bodu A do bodu B, je raw fitness délka cesty mezi těmito body. Může platit, že raw fitness lepšího řešení je vyšší, ale také že čím nižší raw fitness, tím lepší řešení.
- **Standardizovaná** přehodnotí raw fitness tak, že nižší hodnota je vždy lepší. Pokud u raw fitness platí, že čím nižší hodnota, tím lepší jedinec, tak standardizovaná fitness je stejná jako hrubá. Pokud je to naopak, tak je standardizovaná fitness daná rozdílem maximální hodnoty fitness a hrubé fitness.
- **Přizpůsobená** mapuje standardizovanou fitness na interval $[0, 1]$ a platí, že vyšší fitness určuje lepšího jedince.
- **Normalizovaná** udává relativní fitness jedince v rámci populace. Je dána podílem přizpůsobené fitness a sumou přizpůsobených fitness všech jedinců v populaci. Platí, že její hodnoty jsou na intervalu $[0, 1]$, vyšší hodnota značí lepšího jedince a suma normalizovaných fitness populace je rovna 1.

3.1.3 Selekce

Selekce nebo také výběr rodičů je proces, při kterém se vybírají dvojice jedinců, které vytvoří potomky. Rodiče se vybírají z aktuální populace na základě hodnot fitness funkce. Z toho vyplývá, že jedinci s vyšší fitness mají větší šanci vytvořit potomky, než ti s nižší. Při vybírání rodičů se musí dbát na to, aby hodnota fitness byla rovnoměrně rozložená napříč populací. Je to dáno tím, že pokud by byli vybráni pouze jedinci s vyšší fitness, tak by se populace zaplnila pouze potomky těchto rodičů. To by vedlo k tomu, že by si všichni jedinci populace byli podobní a koncentrovali se do stavového prostoru okolo jediného řešení, které nemusí být optimální. Evoluce by tedy konvergovala k lokálnímu maximu. K tomuto jevu dochází, pokud má vybraný selekční algoritmus vysoký selekční tlak (*selection pressure*). Algoritmus s vysokým selekčním tlakem silně preferuje jedince s vysokou fitness [18]. Jedinci s nižší hodnotou jsou sice vzdálenější od optimálního řešení, ale mohou obsahovat užitečné geny.

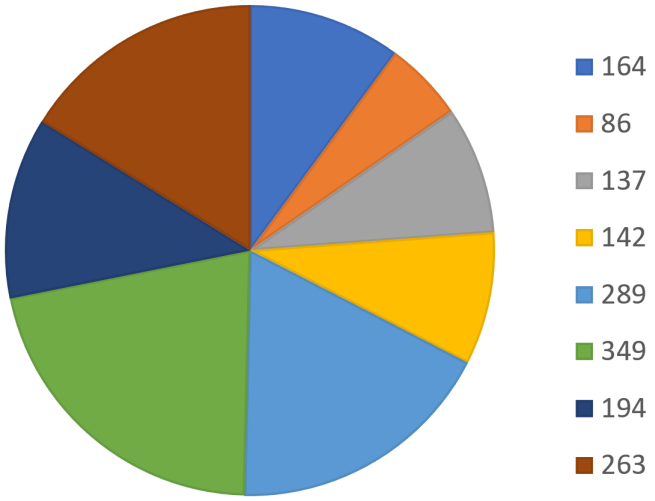
Nejčastěji používaným algoritmem pro selekci rodičů je *turnajový výběr*. Při turnajovém výběru je náhodně vybrán předem daný počet jedinců, kteří se budou účastnit turnaje. Tito jedinci jsou mezi sebou porovnáni a lepší z nich postupuje do dalšího kola. Toto se opakuje dokud z turnaje nevzejde jeden vítěz. Počet kol turnaje závisí na počtu účastníků. Tento způsob zajistí, že budou rodiče vybráni rovnoměrně, protože výběr účastníku jednoho turnaje je náhodný a tudíž může vyhrát i jedinec s relativně nízkou fitness [23].

Dalším způsobem, jak vybírat rodiče, jsou proporcionální selekční algoritmy. Takový algoritmus si lze představit jako ruletu, která má jednotlivá pole přizpůsobena podle relativní velikosti fitness jedinců v rámci populace. Každý jedinec v populaci má své políčko, které proporcionálně odpovídá normalizované fitness jedince (viz obrázek 3.3). Výběr probíhá



Obrázek 3.2: Turnajový výběr, kde je vybíráno ze 4 jedinců, turnaj v a) vyhraje jedinec, který by v turnaji b) neměl šanci

„roztočením rulety a vhozením míčku“ a podle toho, ve kterém poli rulety míček skončí, je vybrán rodič. V případě použití tohoto algoritmu je možné, že jedinec s vysokou hodnotou nebude vybrán vůbec a naopak mohou být vybrání pouze jedinci s nízkou fitness. Tomuto se dá předejít zvolením dostatečně velké populace [18]. Na druhou stranu pokud má jeden jedinec výrazně vyšší fitness, než zbytek populace vzniká zde příliš vysoký selekční tlak.



Obrázek 3.3: Ruleta proporcionálně odpovídající normalizované fitness

Potlačení nadměrného vlivu nadprůměrných jedinců lze dosáhnout výběrem podle pořadí (tzv. *rank selection*). Výběr dle pořadí předpokládá, že jedinci budou vzestupně seřazeni a velikost pole rulety bude úměrná jejich pozici. Pravděpodobnost, že bude vybrán jedinec na i-té pozici, je dána rovnicí 3.1.

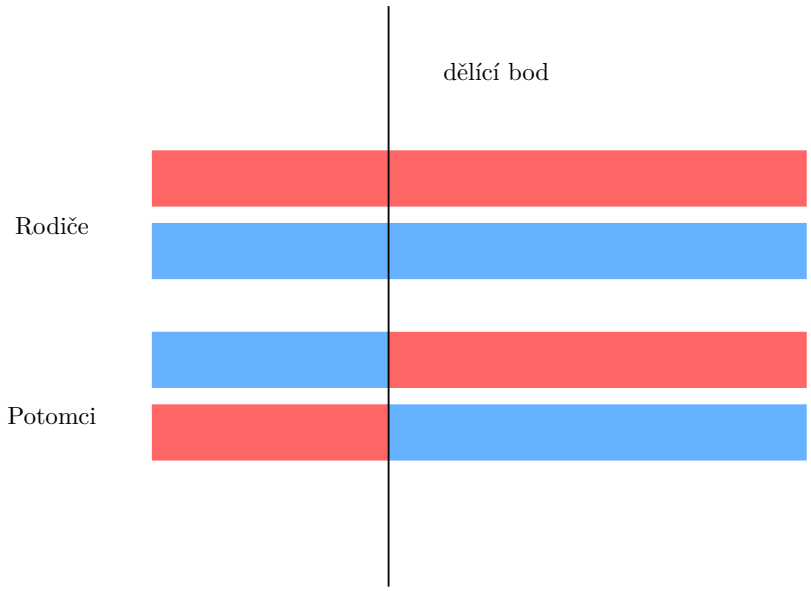
$$p_i = \frac{i}{\sum_{j=1}^{N} j} = \frac{2 \cdot i}{N \cdot (N+1)} \tag{3.1}$$

Tímto jsou eliminovány výrazné rozdíly mezi jedinci. Na druhou stranu v případě, že fitness jedinců se liší pouze nepatrně, jsou tyto rozdíly nepatřičně zveličeny [11].

Nejjednodušším výběrem je tzv. *elitistická selekce*, kdy se vybere potřebný počet jedinců s nejlepším ohodnocením a nejlepší jedinec přežívá, dokud není nahrazen lepším nebo alespoň stejně dobrým [1].

3.1.4 Křížení

Křížení je jeden z genetických operátorů používaný pro vytváření nových jedinců. Existuje mnoho různých technik křížení. Všechny ale mají společnou vlastnost, a sice to, že vždy jde o vzájemnou výměnu částí chromozomů. Nejjednodušší způsob křížení je tzv. *jednobodové křížení*. Jednobodové křížení probíhá tak, že se v chromozomu náhodně zvolí jeden gen a ten slouží jako hranice, která určuje část, kterou si potomek převezme z jednoho rodičovského chromozomu, a kterou z druhého (viz obrázek 3.4).



Obrázek 3.4: Jednobodové křížení

Stejným způsobem funguje i vícebodové křížení. Při k-bodovém křížení se určí k pozic, podle kterých se rozdělí rodičovské chromozomy na $k+1$ částí a zkombinují se způsobem, jaký je zobrazen na obrázku 3.5. Postupným zobecňováním do chvíle, kdy $k-1 = |c_p|$, kde $|c_p|$ určuje velikost rodičovského chromozomu, dojdeme k tzv. *uniformnímu křížení*, kdy se rozhodujeme, po kterém z rodičů potomek zdědí jednotlivé alely. První potomek zdědí alely od rodiče s pravděpodobností $p_p = 0,5$, druhý potomek pak dostane alely od opačného rodiče [11].

Existují i další možnosti křížení v případě, že chromozom není reprezentován vektorem genů. Např. v genetickém programování jsou chromozomy reprezentovány pomocí binárních stromů. V tomto případě jsou zvoleny dva uzly, které se budou účastnit procesu křížení. Křížení je následně provedeno záměnou jejich podstromů [12].

3.1.5 Mutace

Mutace je další genetický operátor používaný k vytváření nových potomků. Spočívá ve změně hodnoty náhodného genu náhodně vybraného potomka. Realizace mutace záleží na způsobu reprezentace chromozomu např. pokud je chromozom reprezentován řetězcem

Obrázek 3.5: Dvoubodové křížení

binárních hodnot, invertuje se hodnota bitu. U reálných čísel se k hodnotě přičte nebo odečte malá náhodná reálná hodnota. Pokud je chromozom tvořen permutací nějakých hodnot, zvolí se náhodně dva geny a ty si vymění své pozice. Mutace jsou považovány za vedlejší operátor a dochází k nim s malou pravděpodobností. Avšak na rozdíl od operátoru křížení, mutace umožňuje vznik nových genů, a proto je vždy potřebná. Poměr zmutovaných genů v potomcích se rovná *mutation rate*. Mutation rate bývá dost nízký, obvykle $0,01 - 0,1$ [18].

3.1.6 Vytvoření nové populace

Po vytvoření potomků a zmutování vybraných rodičů a potomků je zapotřebí vytvořit novou generaci populace. Existuje několik přístupů, jedním z nich tzv. *generační model*, ve kterém jsou všichni jedinci původní generace nahrazeny novými. Toto je zjevně nepraktické, protože můžeme nenávratně ztratit velmi dobré řešení, které by mohlo vést k optimálnímu řešení. Dalším způsobem je *inkrementační model*, který je opakem generačního. V nové populaci je potomkem nahrazen vždy pouze jeden jedinec z původní populace. Dále existují *modely s překrytím generací*. V těchto modelech je nahrazena pouze část původní populace.

3.1.7 Ukončující podmínka

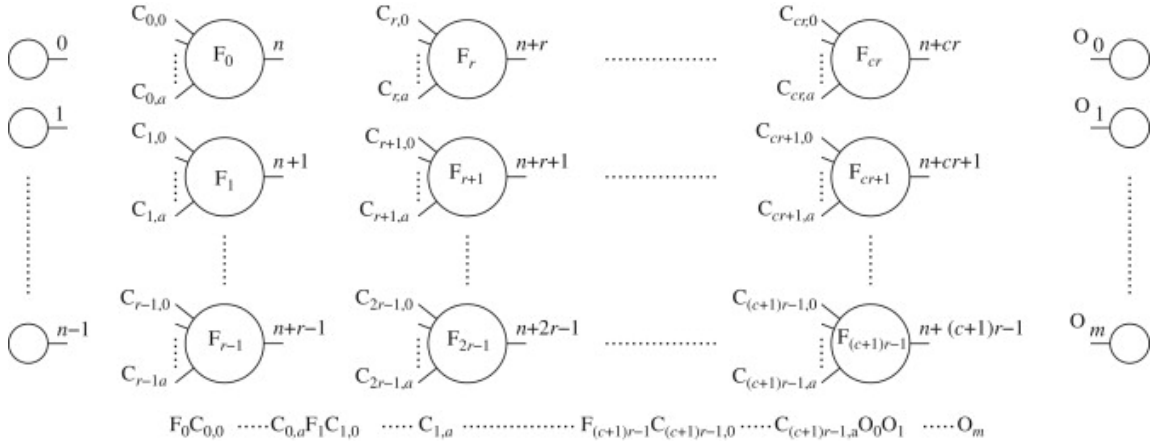
Evoluce je ukončena v momentě, kdy dojde k vygenerování maximálního počtu generací nebo pokud je nalezeno optimální řešení. Ukončující kritérium může být také dáno výpočetním časem nebo počtem generací bez zlepšení fitness. Pokud dojde k ukončení po dosažení maximálního počtu generací, je za řešení považováno dosud nejlepší nalezené řešení.

3.2 Kartézské genetické programování

Genetické programování (GP) oproti genetickým algoritmům nehledá pouze optimální hodnoty chromozomů, ale automaticky generuje spustitelné programy. Určování fitness spočívá ve spuštění vygenerovaného programu a ohodnocení jeho výstupu. V kartézském genetickém programování (CGP) jsou programy reprezentovány acyklickým orientovaným grafem. Tyto grafy mají podobu dvourozměrné mřížky s výpočetními uzly. Jednotlivé geny tvořící genotyp jsou reprezentovány celočíselnými hodnotami. V genotypu je specifikováno, jaké operace jednotlivé uzly provádějí, kde berou vstupní data a jaké uzly jsou přímo připojeny

na programové výstupy. V CGP má genotyp fixní velikost, kdežto fenotyp se může skládat z libovolného počtu uzlů definovaných v genotypu.

Operace, které můžou jednotlivé uzly provádět, jsou definovány uživatelem a jsou uloženy ve vyhledávací tabulce. Každý uzel reprezentuje určitou primitivní výpočetní funkci a je kódován daným počtem genů. Jeden z nich odkazuje ve vyhledávací tabulce určitou funkci, tento gen je nazýván *funkční gen*. Zbylé geny odkazují na uzly, jejichž výstupy budou sloužit jako vstupy pro výpočet a jsou nazývány *spojující geny*. Uzly mohou získávat data od předešlých výpočetních uzlů nebo přímo ze vstupu programu. Počet spojujících genů každého uzlu je dán maximálním počtem vstupů (*aritou*) funkcí ve vyhledávací tabulce. Data ze vstupu programu mají pevně dané pozice v mřížce na indexech od 0 do $n_i - 1$, kde n_i je počet programových vstupů [15]. Na obrázku 3.6 je vidět obecná podoba CGP. Skládá se z mřížky, která má n programových vstupů, r řádků, c sloupců a m výstupů. Pod mřížkou je znázorněn genotyp, skládající se z primitivní funkce F_i a vstupů $C_{i,a}$, kde $i \in \{0, 1, \ldots, (c+1)r - 1\}$ a a je maximální arita.



Obrázek 3.6: Obecná forma CGP [15]

Možnosti propojení uzlů je určeno třemi parametry, které jsou zvoleny uživatelem. Tyto parametry jsou *počet řádku* n_r a *počet sloupců* grafu n_c a parametr *levels-back*, značen l. První dva parametry určují celkový počet výpočetních uzlů $L_n = n_r n_c$. Platí, že vstupy jednotlivých uzlů nesmějí být připojeny na výstupy uzlů, nacházejících se ve stejném sloupci a ve sloupcích napravo od nich. Levels-back určuje vzdálenost sloupců, ze kterých můžou jednotlivé uzly brát své vstupní hodnoty. Tedy pokud $l = 1$, může uzel své vstupy připojit pouze na výstupy uzlů, nacházejících se ve sloupci bezprostředně vlevo od daného uzlu nebo programového vstupu.

Na rozdíl od genetických algoritmů a genetického programování založeného na práci se syntaktickými stromy není primárně používaným genetickým operátorem křížení, ale mutace. Mutace spočívá ve změně alely náhodně vybraného genu. Po mutaci musí zmutovaný gen nabývat validní hodnoty. Pokud je pro mutaci vybrán funkční gen, musí nová hodnota reprezentovat nějakou z definovaných primitivních funkcí ve vyhledávací tabulce. Pokud se jedná o spojující gen, musí zmutovaná alela představovat adresu programového vstupu nebo adresu nějakého uzlu, který se nachází ve sloupcích nalevo od daného uzlu a také vyhovuje hodnotě parametru levels-back. Lze také zmutovat pozici, ze které výstupní uzel získává svou hodnotu. Tato pozice může obsahovat adresu jakéhokoliv funkčního uzlu nebo programového vstupu.

Selekce v CGP probíhá zcela elitisticky způsobem evoluční strategie $(1 + \lambda)$. Princip evoluční strategie spočívá v náhodném vygenerování počáteční populace. Z populace se opět náhodně vybere jedinec, ze kterého je pomocí mutace vytvořeno λ potomků. Následně jsou z nových potomků a rodičů deterministicky vybráni nejlépe ohodnocení jedinci, kteří zformují novou populaci. Strategie $(1 + \lambda)$-ES je tzv. *PLUS*-selekce, ve které se jedinci, formující novou populaci, vybírají z jedinců původní generace a nově vytvořených potomků. V této práci se ještě budeme věnovat evoluční strategii $(1, \lambda)$. Při využití této strategie jedinci přežívají pouze jednu generaci a jsou nahrazeny nově vytvořenými potomky. Tato strategie je označována jako *COMMA*-selekce [1].

Kapitola 4

Evoluční návrh systému schopného generovat booleovské funkce

V této kapitole bude popsán cíl této bakalářské práce a dále budou uvedeny parametry specifikující použité CGP.

Cílem testování aplikace kartézského genetického programování na problému generování booleovských funkcí bude zjistit, jaký vliv má vybraný způsob selekce na proces evoluce funkcí. Jelikož CGP nejčastěji používá evoluční strategii $(1 + \lambda)$, jejíž použitím dochází k elitistickému výběru jedinců, chceme zjistit, jaký efekt by mělo vypuštění vlastnosti elitismu v selekčním procesu. Míru vlivu elitismu zjistíme srovnáním selekce $(1 + \lambda)$, $(1, \lambda)$, turnajovou selekcí a selekcí ruletou. V následujících podkapitolách budou uvedeny parametry specifikující CGP, např. velikost populace, počet řádků a sloupců, fitness funkce použité při hodnocení jedinců, velikost a formát genotypu a další.

4.1 Kartézské genetické programování

Pro tento případ bude použito kartézské genetické programování v nejčastější podobě, tzn., že graf bude tvořen pouze jedním řádkem a parametr l-back bude roven počtu sloupců grafu. Tedy kterýkoliv uzel se bude moci připojit na jakýkoli předcházející uzel. Počet sloupců je určen velikostí genotypu a jak je uvedeno v [22], CGP zpravidla dosahuje dobrých výsledků je-li použit velký genotyp a nízký mutační poměr (mutation rate). Ve výše zmíněném díle ([22]) se pro námi zkoumanou úlohu jako nejlepší kombinace ukázaly velikosti genotypu $1300, 1500$ a 1700 a hodnoty mutačního poměru $7\,\%, 11\,\%$ a $13\,\%$. Sada primitivních funkcí se skládá z logických funkcí `AND`, `OR`, `XOR`, `XNOR` a `AND` s jedním vstupem invertovaným.

4.1.1 Populace a selekce

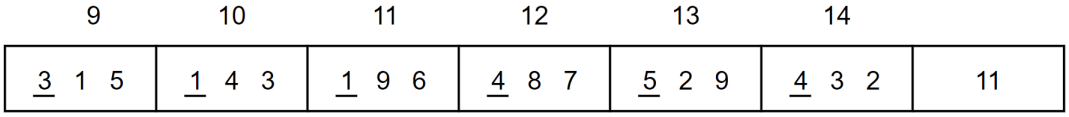
Velikost populace se v tomto případě odvíjí od způsobu selekce. Pro klasické CGP využívající evoluční strategii $(1+\lambda)$ se nejčastěji volí $\lambda = 4$ ([15]), a tedy velikost populace v každé generaci je rovna 5. V případě evoluční strategie $(1, \lambda)$ zvolíme jako u předchozího příkladu hodnotu $\lambda = 4$ a to z důvodu zachování co nejvíce společných faktorů za účelem porovnání výsledků jednotlivých způsobů selekce. Zvolení velikosti populace při použití turnajové selekce a selekce pomocí rulety proběhlo empiricky. Experimentovali jsme se s hodnotami 20, 40 a 60. Z těchto hodnot pro ohnuté funkce nejlépe vyšla hodnota 40 jak pro turnaj tak pro ruletu. V případě ohnutých vyvážených funkcí bylo dosaženo nejlepších výsledků

s hodnotami 60 jedinců pro selekci ruletou a 40 jedinců pro selekci turnajem. Výsledky testů jsou uvedeny v kapitole 6.

4.1.2 Formát genotypu

Genotyp je reprezentován vektorem uzlů různých typů. Konkrétně se genotyp skládá z osmi vstupních uzlů, dále funkčních uzlů a jednoho výstupního uzlu. Počet funkčních uzlů v případě použití evolučních strategií je převzat z [22], kde se jako nejlepší volba ukázala velikost genotypu 1500 funkčních uzlů a mutační poměr 13 %. Stejné hodnoty byly použity i pro selekci ruletou a turnajem.

Funkční uzly se skládají ze tří genů, dvou spojujících a jednoho funkčního. Na spojující i funkční geny se vztahují určitá omezení. Na obrázku 4.1 je zobrazena podoba genotypu z hlediska genů. První gen (podtržené číslo) je funkční gen a jeho hodnota odkazuje jednu z definovaných primitivních funkcí, viz 4.1. Další dvě čísla jsou spojující geny, které odkazují na nějaký předešlý uzel. Pole bez indexu je programový výstup. Vstupní geny byly v tomto případě vynechány.



Obrázek 4.1: Složení genotypu z genů

4.1.3 Fitness funkce

Hlavním cílem této práce je porovnat různé způsoby selekce na problému evoluce booleovských funkcí. Tyto funkce musí splňovat určité požadavky. V této práci se zaměříme na ohnuté funkce a vyvážené funkce s vysokou nelinearitou. V podkapitole 2.3.2 je uvedeno, že ohnuté funkce jsou takové, které mají sudý počet neznámých a dosahují maximální hodnoty nelinearity. Maximální nelinearita pro funkce se sudým počtem proměnných je tedy dána vztahem 4.1.

$$N_{max} = 2^{n-1} - 2^{\frac{n}{2}-1} \tag{4.1}$$

Pro funkce s osmi vstupními proměnnými je hodnota $N_{max} = 120$. Míru nelinearity nl lze určit pomocí Walsh-Hadamardova spektra, viz 2.12. Pro první typ funkcí, tedy pro funkce ohnuté, je fitness funkce daná rovnicí 4.2.

$$F_{f1} = nl \tag{4.2}$$

Vyvážená funkce je taková funkce, jejíž pravdivostní tabulka obsahuje stejný počet jedniček jako nul. Jelikož funkce buď vyvážená je nebo vyvážená není, je nutné nějak určit míru nevyváženosti. Pokud je funkce vyvážena je ji udělena „odměna“ rovna 1. Pokud vyvážená není je penalizována tak, že rozdíl jedniček a nul je vynásoben určitou zápornou hodnotou x. V práci [22] bylo experimentálně zjištěno, že pro funkce s osmi neznámými je pro x vhodná hodnota -5.

$$BAL_f = \begin{cases} 1 & \text{pro } wt(f) = 2^{n-1} \\ -5d & \text{jinak, } d = 2|2^{n-1} - wt(f)| \end{cases} \tag{4.3}$$

Rovnicí 4.3 je dána míra vyváženosti. Proměnná d je dána rozdílem jedniček a nul v pravdivostní tabulce funkce f. Tedy fitness funkce pro vyvážené funkce s vysokou nelinearitou bude dána rovnicí 4.4.

$$F_{f2} = nl + BAL \tag{4.4}$$

4.1.4 Ukončující podmínka

Jelikož hlavním cílem této práce není generovat co nejvíce funkcí s maximální fitness, ale porovnat způsoby selekce, z tohoto důvodu je ukončující kritérium evoluce dáno počtem vyhodnocení fitness funkce.

Kapitola 5

Implementace evolučního návrhu

V této kapitole je popsána implementace evolučního návrhu generátoru kryptograficky významných booleovských funkcí. V sekci 5.2 je popsán způsob reprezentace chromozomu pomocí programovacího jazyka. Dále v sekci 5.3 je uveden popis generování jedinců populace a jejich následné evaluace, neboli vyhodnocení fitness. V následující sekci 5.5 je popsána implementace jednotlivých selekčních mechanismů, které jsou v práci použity, konkrétně se jedná o evoluční strategii (1+λ)-ES, evoluční strategii (1,λ)-ES, turnajový výběr a selekci pomocí rulety.

Práce je implementována v multiparadigmatickém jazyce C++. Je využito jak objektově orientovaného tak imperativního paradigmatu, jež tento jazyk poskytuje. Tento jazyk byl zvolen především z důvodu, že je kompilovaný a poskytuje vyšší výkon, než jazyky interpretované jako např. Python.

5.1 Implementace kartézského genetického programování

Celý běh CGP je implementován funkcí `runCPG`, která požaduje jeden parametr a tím je celočíselná hodnota reprezentující způsob selekce. Konkrétní hodnoty přiřazené jednotlivým selekcím, definice funkce `runCGP` a další funkce související s CGP jsou definovány v hlavičkovém souboru `cgp.h`.

Funkce `runCGP` odpovídá algoritmu 1. Nejdříve je tedy vygenerována počáteční populace. Populace je reprezentována vektorem genotypů. Více informací ke genotypům je uvedeno v následující podkapitole 5.2. Každý jedince populace je následně ohodnocen a hodnota fitness nejlepšího jedince je uložena do proměnné `bestFitness`. Následně je cyklicky opakován proces vytvoření nové generace a jejího ohodnocení. Vygenerování nové populace je uskutečněno voláním funkce `generateNewGeneration`, která přijímá dva parametry a těmi jsou hodnota reprezentující selekční mechanismus (viz 5.5) a reference na vektor reprezentující populaci. Tento cyklus se opakuje dokud nebylo dosaženo maximálního počtu evaluací fitness nebo dokud nebyl nalezen jedinec s maximální fitness.

Funkce `generateNewGeneration` vygeneruje novou populaci pomocí specifikovaného selekčního mechanismu. Implementace jednotlivých selekčních mechanismů jsou popsány v podkapitole 5.5. Nově vygenerovaná populace je uložena ve vektoru, který byl odkazován referencí předanou při volání funkce. Původní populace je přepsána novou.

5.2 Reprezentace chromozomu

V kapitole s návrhem 4 v sekci 4.1.2 je uvedena forma genotypu. Genotyp je reprezentován třídou `Genotype`. Tato třída obsahuje veřejné atributy obsahující jednotlivé typy uzlů. Konkrétně atribut `InputNodes` typu `std::vector` z knihovny `<vector>` obsahující instance třídy `InputNode`, dále atribut `FunctionNodes` typu `std::vector` obsahující instance třídy `FunctionNode` a atribut `OutputNode` typu `OutputNode`. Třídy reprezentující jednotlivé typy uzlů jsou podrobněji popsány v následující sekci 5.2.1. Třída `Genotype` ještě obsahuje privátní atributy `activeNodes`, což je vektor nesoucí adresy aktivních funkčních uzlů, více viz 5.2.2 a atribut `fitness`, ve kterém je uložena hodnota fitness příslušného jedince. Výpočet hodnoty fitness je popsán v sekci 5.3.

5.2.1 Třídy reprezentující uzly

Třída `InputNode` reprezentuje vstupní uzel a obsahuje atribut `Index`, udávající pozici v genotypu a atribut `OutputValue` udávající výstupní hodnotu tohoto uzlu, tedy hodnotu, kterou použije uzel připojený na tento uzel jako vstupní svou hodnotu.

Třída `FunctionNode` reprezentuje funkční uzel a obsahuje veřejné atributy `Input1`, `Input2`, `Function`, `Position`, `Index` a `OutputValue` typu `int`. Atributy `Input1` a `Input2` nesou adresy uzlů, ze kterých budou pro tento uzel získány vstupní hodnoty. `Function` je celočíselná konstanta reprezentující primitivní výpočetní funkci. Všechny konstanty reprezentující primitvní funkce jsou uvedeny v hlavičkovém souboru `nodeFunctions.h`. Dále atribut `Position` obsahuje pozici uzlu v rámci genotypu, je důležité uvést, že tato pozice neodpovídá indexu ve vektoru `FunctionNodes` příslušné instance třídy `Genotype`. Pozice v rámci vektoru funkčních uzlů je uložena v atributu `Index`. Nakonec atribut `OutputValue` podobně jako v `InputNode` udržuje výstupní hodnotu uzlu, tedy hodnotu vypočtenou ze vstupních hodnot pomocí dané primitivní funkce. Výstupní hodnota je vypočtena a vložena do `OutputNode` pomocí metody `setOutputValue`.

Poslední třídou reprezentující jeden z uzlů, konkrétně výstupní uzel, je třída `OutputNode`. Tato třída se skládá ze dvou atributů jimiž jsou `Input`, udávající pozici uzlu v rámci genotypu a `OutputValue`, ve kterém je uložena výstupní hodnota celého chromozomu.

5.2.2 Dekódování chromozomu

Dekódováním chromozomu je myšlena extrakce konkrétní booleovské funkce reprezentované pomocí daného genotypu. Fenotyp je určen aktivními uzly, tzn. uzly, které jsou přímo nebo nepřímo spojeny s programovým výstupem. Aktivní uzly dostaneme při průchodu genotypem od výstupního uzlu směrem ke vstupním. K provedení této procedury je použit vektor `activeNodes`, ve kterém jsou uloženy adresy aktivních uzlů a fronta obsahující uzly, které se musí zpracovat. Zpracování uzlu v tomto případě znamená zjištění, zda jeho vstupní uzly jsou programové vstupy nebo předcházející funkční uzly. Pokud se jedná o funkční uzly, jsou jejich adresy vloženy do fronty pro zpracování, pokud se jedná o programové vstupy je uzel vyřešen, v obou případech je zpracovávaný uzel z fronty vyjmut. Tento proces se opakuje dokud není fronta vyprázdněna. Po nalezení všech aktivních uzlů je vektor s aktivními uzly vzestupně seřazen a následně z něho jsou odstraněny duplicity.

Dekódování chromozomu je znázorněno pomocí algoritmu 2 a implementováno metodou `findActiveNodes` třídy `Genotype`.

Algorithm 2: Dekódování chromozomu

Result: Vektor indexů aktivních uzlů

```
1  Vytvoř vektor aktivních uzlů activeNodes a frontu queue;
2  if OutputNode.Input odkazuje funkční uzel then
3  |   vlož adresu uzlu uloženou v OutputNode.Input do queue;
4  |   Vlož index výstupního uzlu OutputNode do activeNodes;
5  end
6  while queue není prázdná do
7  |   nastav aktuální uzel na první uzel z fronty;
8  |   if Input1 aktuálního uzlu odkazuje funkční uzel then
9  |   |   vlož adresu uzlu do queue;
10 |   |   vlož adresu uzlu do activeNodes;
11 |   end
12 |   if Input2 aktuálního uzlu odkazuje funkční uzel then
13 |   |   vlož adresu uzlu do queue;
14 |   |   vlož adresu uzlu do activeNodes;
15 |   end
16 |   odstraň první uzel z fronty
17 end
18 seřaď vzestupně activeNodes podle adres;
19 vymaž duplicity z activeNodes;
20 vrať vektor activeNodes;
```

Dekódování chromozomu a získání aktivních uzlů je nutné pro výpočet výstupní hodnoty funkce a následné ohodnocení jedince pomocí fitness funkce.

5.3 Generování a evaluace jedinců

V této podkapitole je popsána implementace generování jedinců populace (sekce 5.3.1) a postupu vyhodnocení fitness (5.3.2)

5.3.1 Generování jedinců

V podkapitole 5.2 je uvedeno, že jedinec je reprezentován třídou `Genotype`. Generování jedinců je uskutečněno pomocí volání konstruktoru této třídy. Konstruktor třídy `Genotype` inicializuje všechny uzly pomocí konstruktorů příslušných tříd.

V případě vstupních uzlů se jedná o naplnění vektoru `InputNodes` instancemi třídy `InputNode`. V konstruktoru této třídy je provedena inicializace atributu `Index` na odpovídající index vstupu. Pro funkce o osmi proměnných tedy nabývá hodnot $\{0, 1, \ldots, 7\}$.

V případě funkčních uzlů je nastavena hodnota atributů `Index` a `Position` a příslušné hodnoty (význam všech atributů je uveden v sekci 5.2.1). Stěžejní částí je inicializace atributů představující spojující geny a funkční gen. Hodnoty těchto atributů musejí odpovídat pravidlům uvedených v podkapitole 3.2. Konkrétně jde o to, že spojující geny mohou být napojeny pouze na uzly s nižší hodnotou atributu `Position` nebo na uzly představující programové vstupy. Tato vlastnost je ošetřena funkcí `getRandomInputGene` v hlavičkovém souboru `nodeFunctions.h`. Obdobně tomu je v případě funkčního genu, který musí odkazovat

některou z primitivních funkcí definovaných v podkapitole 4.1. Funkce je odkazována celočíselnou hodnotou. Hodnota funkčnímu genu je přiřazena funkcí `getRandomFunctionGene`, která je deklarována spolu s definicemi konkrétních hodnot odpovídajících primitivním funkcím v souboru `nodeFunctions.h`.

Dále je inicializován uzel `OutputNode` reprezentující programový výstup. To je provedeno voláním konstruktoru třídy `OutputNode`, ve kterém je nastavena hodnota atributu `Input`, kterým je určen uzel, ze kterého bude převzata výstupní hodnota programu.

Nakonec je v konstruktoru třídy `Genotype` inicializován atribut `activeNodes`. Aktivní uzly jsou nalezeny pomocí metody `findActiveNodes`. Postup při hledání těchto uzlů je popsán v sekci 5.2.2 a algoritmem 2.

5.3.2 Evaluace jedince

Výpočet fitness jedince se skládá z několika kroků. Nejdříve je nutné z genotypu dekódovat fenotyp, který reprezentuje vygenerovanou booleovskou funkci, tento proces byl popsán v sekci 5.2.2.

Následujícím krokem je vyhodnocení funkce pro všechny kombinace vstupů, jinými slovy vytvoření pravdivostní tabulky dané funkce. Vytvoření pravdivostní tabulky realizuje metoda `getTruthTable` třídy `Genotype`. Tato metoda pouze zaštiťuje získání vstupu a vyhodnocení funkce pro aktuální vstup. Získání vstupů a vyhodnocení funkce je provedeno pomocí metod `fillInputValues` a `evaluateFunction` po řadě. Načtení vstupu do vstupních uzlů je realizováno nastavením atributu `OutputValue` vstupního uzlu s příslušným indexem. Vektory vstupních hodnot jsou při spuštění programu inicializovány funkcí `generateNInputVectors` a uloženy v konstantní globální proměnné `INPUT_TABLE` hlavičkového souboru `nodeFunctions.h`, ze které jsou následně načítány při procesu vytváření pravdivostní tabulky. Po načtení vstupních hodnot do programových vstupů je funkce vyhodnocena a získaná hodnota vložena do pravdivostní tabulky. K vyhodnocení funkce je využit vektor s aktivními uzly daného jedince. Atribut `activeNodes` obsahuje vzestupně seřazené celočíselné hodnoty reprezentující pozice aktivních uzlů. Pro účel vyhodnocení jednotlivých uzlů jsou z tohoto vektoru odstraněny duplicitní hodnoty (aby nedocházelo k několikanásobnému vyhodnocení jednoho uzlu), viz 5.2.2. Samotné vyhodnocení probíhá procházením vektoru s aktivními uzly. Nejdříve jsou načteny výstupní hodnoty uzlů, na které je zpracovávaný uzel napojen. Následně je vyhodnocena primitivní funkce přiřazena danému uzlu a vypočtená hodnota je vložena do atributu `OutputValue`. Toto se opakuje dokud nejsou vyhodnoceny všechny aktivní uzly. Nakonec je výstupní hodnota posledního aktivního uzlu nastavena jako výstupní hodnota programového výstupu. Tato hodnota je pak vložena do pravdivostní tabulky. Načtení vstupů a vyhodnocení se provádí pro všechny kombinace vstupů, kterých je 2^n.

Po získání pravdivostní tabulky přichází řada na samotný výpočet jednotlivých částí fitness funkce. Konkrétně se jedná o výpočet nelinearity a vyváženosti funkce. Se získanými hodnotami nelinearity a případně vyváženosti, stačí dopočítat skutečnou hodnotu fitness podle rovnic 4.2 v případě ohnutých funkcí a 4.4 v případě vyvážených funkcí s vysokou nelinearitou.

Implementace výpočtu nelinearity

Výpočet nelinearity se skládá z několika kroků. Z předchozího kroku již máme vypočtenou pravdivostní tabulku pro danou funkci. Jako první je nutné upravit tabulku tak, že nahradíme hodnoty 0 za hodnoty -1. Toto lze provézt funkcí `adjustVector`, která poža-

duje referenci na vektor reprezentující pravdivostní tabulku a tuto tabulku upraví, podle dříve zmíněných potřeb. Je nutné zmínit, že původní hodnoty budou přepsány. Z tohoto důvodu je v případě generování vyvážených, ohnutých funkcí nejprve počítána vyváženost a až poté nelinearita. Dalším krokem po úpravě pravdivostní tabulky je provedení FWHT a tím dostaneme Walsh-Hadamardovo spektrum, viz 2.3.2. Výstupní vektor získaný pomocí aplikace FWHT na upravenou pravdivostní tabulku je nutné opět opravit. Opravení výstupního je vektoru je provedeno získáním absolutní hodnoty každého prvku a následným dělením dvěma. V opraveném výstupním vektoru najdeme maximální hodnotu a tím získáme míru korelace *cor* s nejbližší afinní funkcí. Nyní, s nalezenou korelací, můžeme vypočítat nelinearitu pomocí rovnice $nl = 2^{n-1} - cor$.

Implementace výpočtu vyváženosti

Jak bylo specifikováno v sekci 2.3.3, vyváženost je určena rozdílem jedniček a nul v pravdivostní tabulce. Vyvážená funkce má stejný počet nul jako jedniček. V sekci 4.1.3 je uvedeno, že vyvážená funkce dostane „odměnu“ a nevyvážená funkce bude penalizována úměrně její nevyváženosti. Výpočet vyváženosti je znázorněn algoritmem 3.

Algorithm 3: Výpočet vyváženosti

```
Result: Míra vyváženosti BAL
Nastav počítadlo jedniček na 0;
foreach hodnota v pravdivostní tabulce do
    if hodnota je rovna jedné then
        Inkrementuj počítadlo jedniček;
    end
end
Nastav počet jedniček na rozdíl velikosti pravdivostní tabulky a počtu jedniček;
if počet jedniček je roven počtu nul then
    Nastav BAL na 1;
else
    Spočti rozdíl jedniček a nul d;
    Nastav BAL na −5 · d;
end
Vrať hodnotu BAL;
```

5.4 Mutace

V kartézském genetickém programování je běžné, že mutace je jediným použitým genetickým operátorem, viz 3.2. Mutovány mohou být různé geny. V případě CGP jsou geny dvou druhů: funkční a spojující. Jak je ukázáno v obrázku 4.1 funkční uzel se skládá ze třech genů a každý z těchto genů může být mutovaný. Dále ještě může být zmutován spojující gen programového výstupu.

Operace mutace je implementována funkcí `mutate`, které je předána reference na rodiče. Funkce `mutate` nejdříve vytvoří potomka hlubokou kopií rodiče, se kterým dále pracuje. Dalším krokem je náhodně vygenerována celočíselná hodnota z rozmezí $\{0, 1, \ldots, n\}$. Hodnota n je dána vztahem $n = 1500 \cdot 3 + 1$, kde 1500 je počet funkčních uzlů, 3 jsou geny ve funkčním uzlu a 1 gen je obsažen ve výstupním uzlu. Vygenerovanou hodnotou je dán uzel a zároveň i konkrétní gen, který bude mutován. Index uzlu je z hodnoty vypočítán celočíselným

dělením a konkrétní gen je určen operací modulo. Vybraný gen je následně zmutován na jakoukoliv validní hodnotu. V případě, že byl mutován spojující gen je aktualizován vektor s aktivními uzly.

5.5 Selekční mechanismy

Tato práce je zaměřená na způsoby selekce jedinců, kteří se stanou rodiči a budou z nich vytvořeni noví jedinci. Po vytvoření potomků následuje výběr jedinců, kteří budou tvořit novou generaci. Výběr rodičů a potomků probíhá různými způsoby na základě zvoleného způsobu selekce. Implementace konkrétních způsobů selekce jsou uvedeny dále.

5.5.1 Evoluční strategie $(1 + \lambda)$

Evoluční strategie $(1+\lambda)$ je nejčastěji používaným způsobem selekce v případě CGP. Tento způsob selekce je zastřešen pomocí funkce `onePlusLambda`, která zároveň vygeneruje novou populaci. Princip $(1 + \lambda)$-ES je popsán v podkapitole 3.2. Nejdříve je tedy z aktuální populace náhodně vybrán jeden jedinec, který se stane rodičem. Poté je vygenerováno λ potomků, kteří budou s původní populací „soutěžit“ o místo v nové generaci. Vytvoření nového jedince se skládá ze dvou částí. Nejprve se rozhodne zda bude jedinec mutován či ne. O to se stará funkce `willMutate`, která na základě mutačního poměru rozhodne, zda mutace proběhne. V případě, že mutace proběhnout má, je jedince zvolený jako rodič zmutován. Po mutaci je vyhodnocena fitness nového jedince a tento jedinec je vložen do dočasného vektoru potomků. V případě, že k mutaci nedojde, je rodič v nezměněné podobě vložen mezi potomky. V takovém případě není potřeba vyhodnocovat fitness, protože rodič již ohodnocen byl. Nejedná se o nejvhodnější způsob použití mutace, protože tím je velice zpomalena konvergence k optimálnímu řešení. Nicméně tento způsob je využit ve všech selekčních mechanismech stejně a tedy při porovnání jednotlivých mechanismů by se neměl projevit. Tento proces vytváření nových jedinců se ve stejné podobě opakuje napříč všemi způsoby selekce. Provedení mutace bylo popsáno v podkapitole 5.4.

Nově vytvoření potomci jsou vloženi do dočasného vektoru, nazvaného `selectionVector`, který je použit k selekci nových jedinců, kteří budou tvořit novou populaci. Následně jsou do selekčního vektoru vloženy jedinci původní populace. V momentě kdy `selectionVector` obsahuje nové i původní jedince, jsou všichni jedinci ve vektoru seřazeni sestupně podle fitness.

Řazení je provedeno pomocí funkce `std::stable_sort` z knihovny `<algorithm>`, která seřadí prvky tak, že prvky se stejnou hodnotou zanechá v pořadí, ve kterém se původně nacházeli. Tedy potomek se stejnou fitness jako některý z rodičů bude umístěn na přednější pozici, než zmíněný rodič. Tímto je realizován požadavek, že nejlepší jedince setrvá v populaci, dokud není nahrazen lepším nebo alespoň stejně dobrým jedincem. Funkce `std::stable_sort` jazyka C++ implicitně řadí prvky vzestupně pomocí operátoru `<`. Sestupného řazení lze docílit specifikací komparátoru při volání funkce `std::stable_sort` [5]. Komparátor je funkce vracející hodnotu datového type `bool`, které jsou předány dvě hodnoty, které mají být porovnány. V tomto případě jako komparátor slouží funkce `descendingComparator`, která porovnává dvě instance třídy `Genotype` podle hodnoty fitness.

V seřazeném selekčním vektoru je ponecháno prvních n prvků, kde n je velikost populace. Těmito prvky jsou nejlépe ohodnocení jedinci napříč původní a novou generací s tím, že potomek má přednost před rodičem se stejnou fitness. Nakonec je původní populace nahrazena nově vybranými jedinci.

5.5.2 Evoluční strategie $(1, \lambda)$

Rozdíl mezi evoluční strategií $(1, \lambda)$ a $(1 + \lambda)$ je popsán v podkapitole 3.2. Spočívá tedy v tom, že jedinci původní populace jsou nahrazeni novými. Jinými slovy, při sestavování nové populace jsou jedinci vybíráni pouze z potomků.

Potomci jsou vytvářeni způsobem, který byl uveden v sekci 5.5.1. Zkráceně se tedy jedná o mutaci rodiče, ke které dochází s danou pravděpodobností (mutační poměr). Rodič byl opět vybrán náhodně z původní populace.

Vzhledem k tomu, že je vytvořeno λ potomků, ale v kapitole 4 byla velikost populace stanovena na 5, je do nové populace také přidán rodič. Ze stejného důvodu tedy není potřeba selekční vektor seřadit a odebírat nejhorší jedince.

Tento způsob selekce je implementován funkcí `oneDashLambda`, která společně s výběrem rodičů a vytvořením nových potomků nahradí původní populaci novou.

5.5.3 Selekce turnajem

Selekce turnajem, na rozdíl od evolučních strategií, volí rodiče náhodně pouze částečně. Nejprve jsou náhodně vybráni jedinci, kteří se budou účastnit turnaje. Vítěz turnaje už je vybrán deterministicky na základě jeho fitness. Každého turnaje se účastní daný počet jedinců. Počet účastníků byl stanoven na hodnotu 4 a celkový počet turnajů odpovídá velikosti populace. Tedy vítěz každého turnaje se stane rodičem, ze kterého následně bude vytvořen potomek. Tento způsob selekce je reprezentován funkcí `tournamentSelection`.

Turnaj se čtyřmi účastníky je zobrazen na obrázku 3.2. Vyhodnocení turnaje je implementováno funkcí `tournament`. Tato funkce požaduje jeden argument a tím vektor účastníků daného turnaje. Turnaj se čtyřmi účastníky se skládá ze dvou kol, ale funkci `tournament` lze použít také pro turnaje jejichž počet účastníků, odpovídá jakékoliv mocnině čísla 2. Tedy pokud se počet účastníků rovná 2^n, počet kol turnaje je roven n. První kolo vyhodnocování probíhá porovnáváním fitness sousedních prvků vektoru s účastníky. Jedinec s nižší hodnotou fitness je z vektoru odstraněn pomocí metody `erase`, která odstraní prvek na daném indexu a upraví velikost vektoru [6]. Totožným způsobem probíhá i druhé kolo a také by tak probíhala kola další, kdyby byl vyšší počet účastníků (počet by musel odpovídat výše zmíněnému pravidlu). Vyhodnocování kol probíhá tak dlouho, dokud se ve vektoru nenachází pouze jeden prvek. Poslední jedinec je vítězem turnaje a stává se tak rodičem.

Ve funkci `tournamentSelection` jsou dále z rodičů vytvořeni potomci a sestavena nová populace. Toto je provedeno stejným způsobem, jako v případě evoluční strategie $(1 + \lambda)$, viz 5.5.1.

5.5.4 Selekce ruletou

Pro selekci ruletou byla zvolena variace rulety, ve které jsou jednotlivá pole rulety úměrně přizpůsobena pozici jedince. Jedná se tedy o rank selection. viz 3.1.3. Ruletová selekce je implementována funkcí `rouletteSelection`.

Nejprve je tedy nutné přiřadit všem jedinců v populaci odpovídající pozici a ohodnocení dané pozice. Toho je docíleno vytvořením vektoru `positions`, který obsahuje ohodnocení pozic od nejnižší pozice po nejvyšší. Následně jsou podle fitness vzestupně seřazeni jedinci populace. V tento moment index jedince v populaci odpovídá indexu ohodnocení pozice. Následuje samotný výběr jedince, který se stane rodičem. Za tímto účelem je vygenerováno náhodné číslo `x` z rozsahu $\{0, 1, \ldots, n\}$, kde n je rovno sumě ohodnocení všech pozic. Na základě tohoto čísla je následně určen index jedince v rámci populace, který bude zvolen jako

rodič. Zjištění indexu je implementováno průchodem vektoru `positions`. Při procházení vektoru `positions` dochází k porovnávání čísla x a aktuálního prvku vektoru `positions`. V momentě kdy je nalezen prvek, jehož hodnota je vyšší než x, je uložena hodnota aktuálního indexu a prohledávání je ukončeno. Uložená hodnota indexu udává jedince, který byl vybrán jako rodič. Tímto způsobem je vybrán počet rodičů odpovídající velikosti populace.

Opět následuje vytvoření potomků a deterministický výběr jedinců, kteří vytvoří novou populaci.

Kapitola 6

Experimentální ověření a vyhodnocení výsledků

Účelem provedení experimentů s implementovaným systém bylo získat data, která poslouží k vyhodnocení zvolených selekčních mechanismů. Zvolenými selekčními mechanismy jsou: evoluční strategie $(1 + \lambda)$, evoluční strategie $(1, \lambda)$, turnajová selekce a selekce ruletou.

Experimentování se skládalo z optimalizačních testů a rozsáhlejšího testování všech selekčních metod. Účelem optimalizačních testů bylo nalézt velikosti populací, které dosahují nejlepších výsledků. Optimalizační testy byly provedeny pro případy selekce ruletou a turnajem. Velikosti populací byly zkoumány jak z hlediska ohnutých funkcí, tak z hlediska vyvážených funkcí s vysokou nelinearitou. Následně byly nejlepší nalezené hodnoty použity při testování všech typů selekce. Experimenty týkající se všech selekcí opět byly aplikovány na problém generace ohnutých funkcí a vyvážených funkcí s vysokou nelinearitou.

Optimalizační testy se týkaly pouze selekce pomocí turnaje a rulety a jediným proměnlivým faktorem byla velikost populace. V případě evolučních strategií se populace skládaly z 5 jedinců, tyto hodnoty byly převzaty z [15]. Velikosti populace, se kterými bylo experimentováno, byly rovny hodnotám 20, 40 a 60. Pro ohnuté funkce se v obou případech ukázala jako nejlepší velikost populace rovna 40. V případě ohnutých vyvážených funkcí vyšla pro selekci ruletou nejlépe velikost populace 60 a pro selekci turnajem opět hodnota 40. Podrobné výsledky optimalizačních testů jsou uvedeny v podkapitole 6.1.

Sbíraná data jsou složena z hodnot maximální dosažené fitness v jednom běhu CGP a délek každého běhu. Nasbíraná časová data, nakonec nebyla použita při vyhodnocování dílčích testů, protože kvůli nevyváženému zatížení počítače, na kterém testy běžely, byly výsledky do určité míry zkresleny.

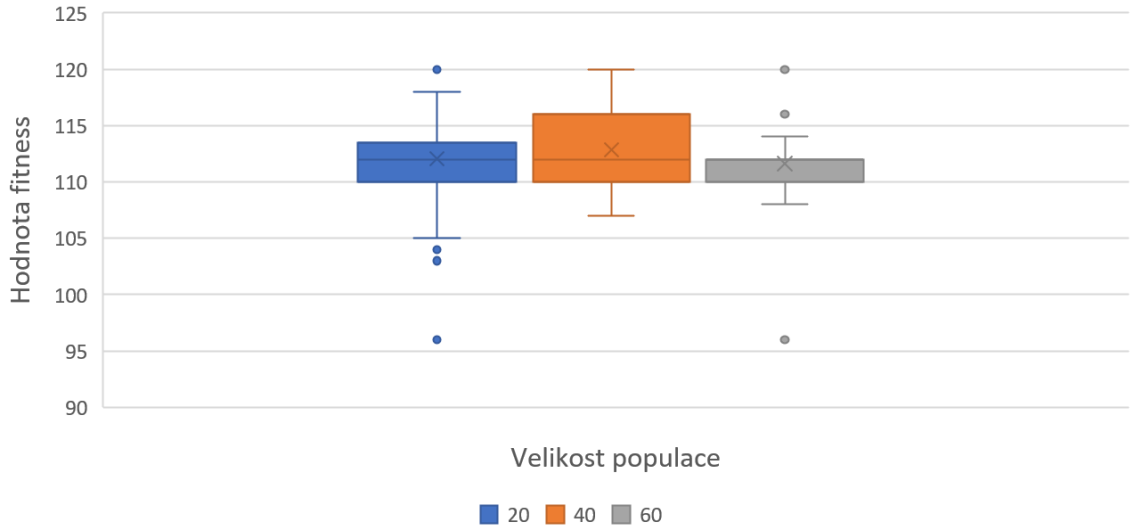
6.1 Optimalizační experimenty

Optimalizační testy byly provedeny za účelem nalezení optimálních hodnot ze zvolených velikostí populace. Nalezením optimálních hodnot bylo sledováno zmírnění rozdílů mezi evolučními strategiemi. Parametry pro evoluční strategii byly převzaty z díla [22], které se, mimo jiné, věnuje nalezení optimálních parametrů pro CGP použité pro generování kryptograficky významných booleovských funkcí.

Optimalizační testy jsou spouštěny v etapách po 60 bězích.

6.1.1 Ohnuté funkce, selekce ruletou

V případě selekce ruletou využité v CGP aplikovaném na problém generování ohnutých funkcí, tedy funkcí se sudým počtem vstupů a maximální nelinearitou, bylo experimentováno s velikostmi populace. Jak již bylo zmíněno výše, testované hodnoty velikosti populace byly stanoveny na hodnoty 20, 40 a 60. V následujícím grafu 6.1 jsou znázorněny rozdíly průměrné, maximální a minimální hodnoty pro jednotlivé velikosti populace. Z grafu je



Obrázek 6.1: Závislost průměrné fitness na velikosti populace v případě ohnutých funkcí a selekce ruletou

vidět, že při všech velikost populace byli nalezeni jedinci dosahující maximální hodnoty fitness (pro ohnuté funkce s osmi vstupy se jedná o hodnotu 120, viz 4.1.3). Průměrné hodnoty fitness pro jednotlivé velikosti populace jsou uvedeny v tabulce 6.1, které udává, že průměrná fitness vyšla nejlépe pro velikost populace 40. Avšak od průměrné fitness dosažené při použití populace o 20 jedincích se příliš neliší. Nicméně z obrázku 6.1 je zřejmé, že rozptyl hodnot je znatelně menší. Hodnoty rozptylů pro velikosti populace 20 a 40 jsou rovny $22,6142$ a $13,5822$ po řadě.

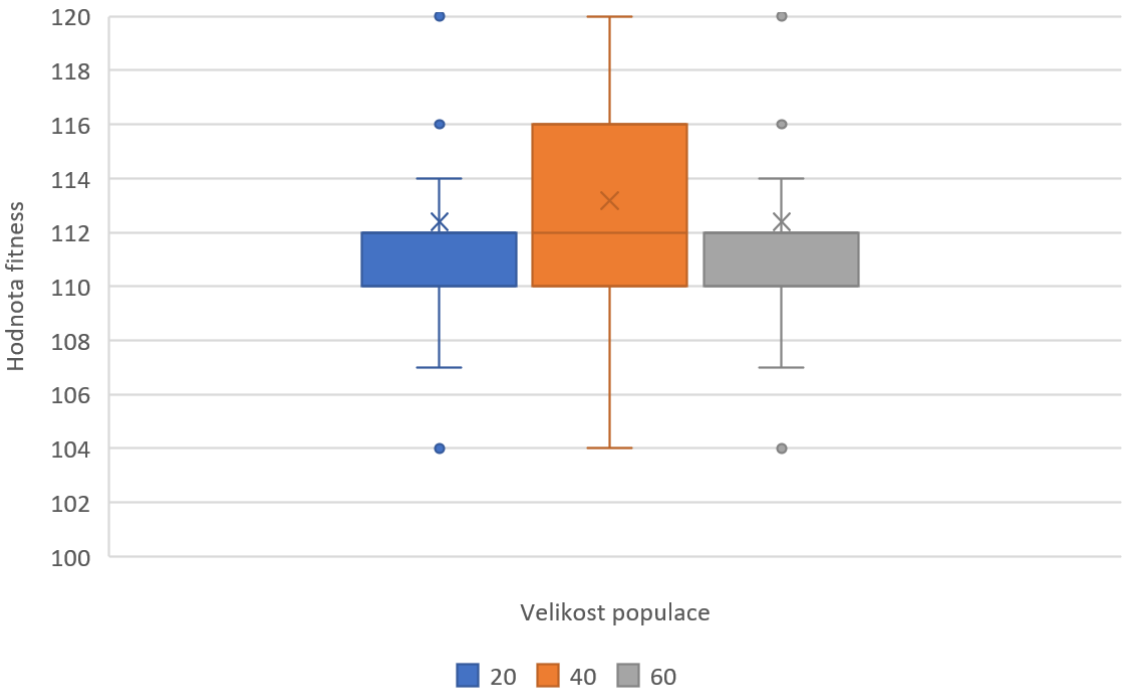
	Velikost populace		
	20	40	60
Průměr	112,0500	112,8667	111,6167
Rozptyl	22,6142	13,5822	13,0697
Úspěšnost	16,67 %	13,33 %	8,33 %

Tabulka 6.1: Průměrná fitness, rozptyl hodnot a úspěšnost jednotlivých velikostí populace

Ze získaných výsledků byla zvolena velikost populace 40, která byla následně použita při rozsáhlejším testování, které se věnovalo všem zvoleným selekčním mechanismům použitým při generování ohnutých funkcí.

6.1.2 Ohnuté funkce, selekce turnajem

Optimalizace velikosti populace při použití selekce turnajem probíhala obdobně jako v případě ruletové selekce. Celkové výsledky jsou zobrazeny v grafu 6.2, ze kterého je vidět, že podobně jako u ruletové selekce bylo ve všech případech dosaženo maximální fitness. V případě turnajové selekce bylo v téměř ve všech případech nalezeno více jedinců, kteří



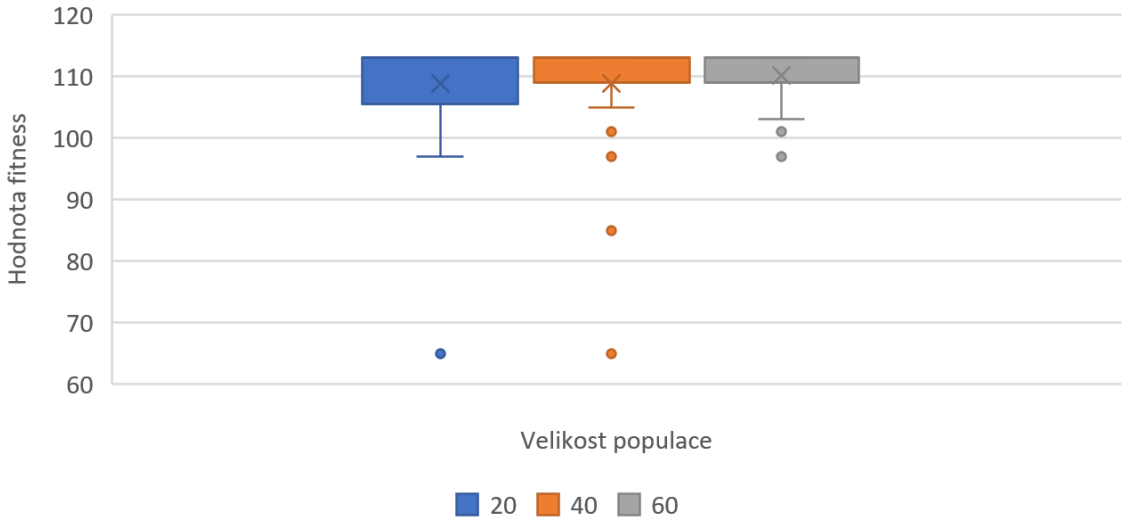
Obrázek 6.2: Závislost průměrné fitness na velikosti populace v případě ohnutých funkcí a selekce turnajem

dosahovali maximální hodnoty fitness, s výjimkou populace o 20 jedincích. V tomto případě byl vygenerován stejný počet jedinců s maximální fitness. Průměrné hodnoty nejlepších jedinců vygenerovaných během jednotlivých běhů jsou uvedeny v tabulce 6.2. Z této tabulky

	Velikost populace		
	20	40	60
Průměr	112,1333	113,1667	112,3833
Rozptyl	15,2156	18,3056	14,4031
Úspěšnost	15 %	23,33 %	15 %

Tabulka 6.2: Průměrná fitness, rozptyl hodnot a úspěšnost jednotlivých velikostí populace

je vidět, že i průměrná fitness vygenerovaných ohnutých funkcí byla pro všechny velikosti populace vyšší, než ta v případě selekce pomocí rulety. Na základě získaných výsledků byla z testovaných hodnot jako nejlepší zvolena velikost populace rovna 40. Tato hodnota se vztahuje pouze na použití selekce turnajem v případě generování ohnutých funkcí.



Obrázek 6.3: Závislost průměrné fitness na velikosti populace v případě ohnutých vyvážených funkcí a selekce ruletou

6.1.3 Vyvážené funkce s vysokou nelinearitou, selekce ruletou

Optimalizační testy týkající se ohnutých vyvážených funkcí probíhaly stejně jako v případě funkcí pouze ohnutých. Výsledky evoluce ohnutých vyvážených funkcí s použitím selekce pomocí rulety jsou znázorněny v obrázku 6.3. Z tohoto grafu je vidět, že s žádnou ze zvolených velikostí populace nebyl nalezen jedince s maximální hodnotou fitness. Maximální teoretická hodnota fitness vyvážených 8-vstupých funkcí s vysokou nelinearitou je rovna 118 (viz [22]), ve všech případech však bylo dosaženo maximálně fitness 113.

Průměrné hodnoty získaných fitness pro jednotlivé velikosti populace jsou zároveň s příslušnými rozptyly uvedeny v tabulce 6.3.

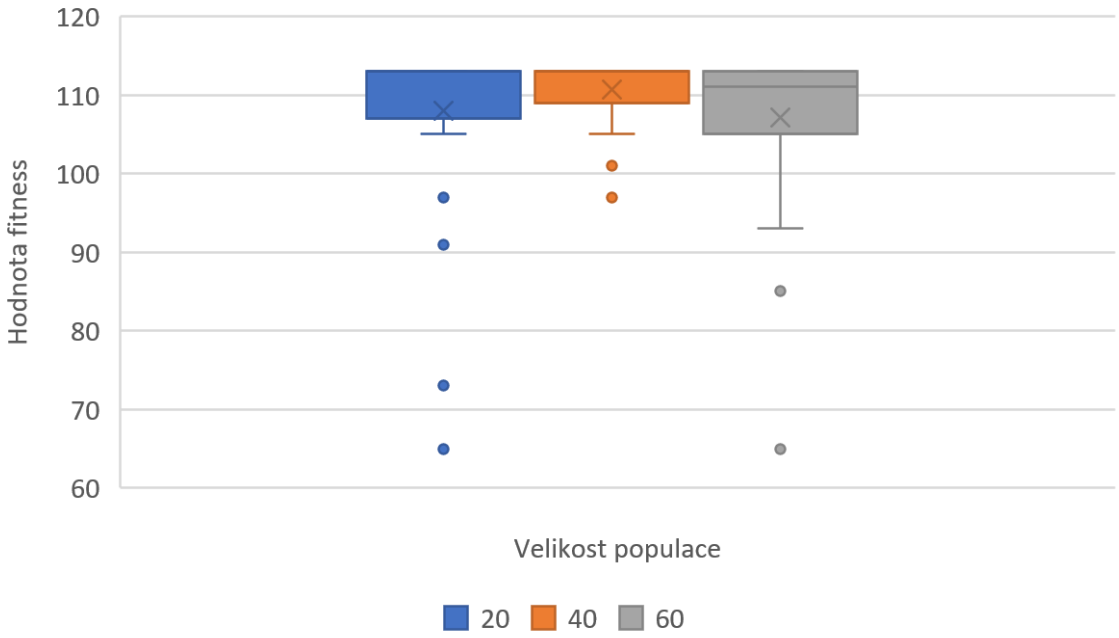
	Velikost populace		
	20	40	60
Průměr	108,9000	108,8667	110,1333
Rozptyl	59,3900	90,6489	17,5156
Úspěšnost	0 %	0 %	0 %

Tabulka 6.3: Průměrná fitness, rozptyl hodnot a úspěšnost jednotlivých velikostí populace

Z tabulky je vidět, že v případě velikosti populace rovné 60 je průměrná hodnota o více. než jeden stupeň vyšší, než průměrné hodnoty pro ostatní velikosti. Navíc rozptyl hodnot pro tuto velikost je několikanásobně nižší, než rozptyl v ostatních případech. Z tohoto důvodu byla použita při celkovém testování.

6.1.4 Vyvážené funkce s vysokou nelinearitou, selekce turnajem

Poslední optimalizační test byl věnován selekci turnajem, použité v případě evoluce ohnutých vyvážených funkcí. Zkoumané velikosti populace jsou stejné jako v ostatních případech. Získané hodnoty jsou zobrazeny grafem 6.4.



Obrázek 6.4: Průměrná fitness jednotlivých velikostí populace v případě ohnutých vyvážených funkcí a selekce turnajem

Získaná data při testování turnajové selekce s různými velikostmi populace dopadla podobně jako v případě selekce ruletou. Žádný vygenerovaný jedinec nedosáhl vyšší fitness, než 113. Na druhou stranu této hodnoty dosáhla, více než polovina všech jedinců napříč všemi testovacími běhy. V tabulce 6.4 jsou uvedeny, roztříděné podle velikosti populace, průměrné hodnoty dosažené fitness a rozptyly těchto hodnot.

	Velikost populace		
	20	40	60
Průměr	107,933	110,7333	107,1333
Rozptyl	111,1289	18,3289	99,4489
Úspěšnost	0 %	0 %	0 %

Tabulka 6.4: Průměrná fitness, rozptyl hodnot a úspěšnost jednotlivých velikostí populace

Už podle průměrné dosažené fitness, uvedené v tabulce 6.4 je vidět, že populace při použití populace o velikosti 40 jedinců, bylo dosaženo podstatně lepších výsledků. Rozdíly fitness jsou podpořeny hodnotou rozptylů příslušných k jednotlivým velikostem populace. V případě populace tvořené ze 40 jedinců je rozptyl hodnot opět násobně menší, než rozptyl u zbylých dvou možností.

6.2 Testy zvolených selekčních metod

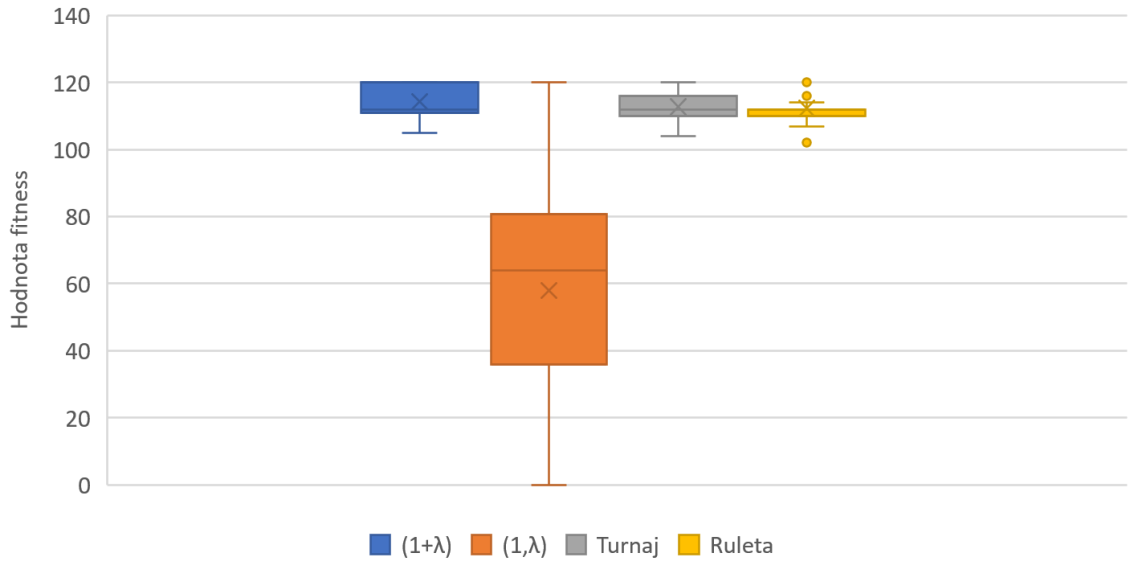
Tyto testy probíhaly velice podobně jako optimalizační testy s jedním rozdílem, a sice při optimalizačních testech bylo provedeno pouze 60 běhů. V případě těchto testů bylo běhů prováděno 100. V sekci 6.2.1 jsou popsány testy ohnutých funkcí a uvedeny výsledky

pro jednotlivé typy selekce. Následující sekce 6.2.2 je věnována testování zaměřenému na vyvážené funkce s vysokou nelinearitou.

6.2.1 Ohnuté funkce

Testování evoluce ohnutých funkcí bylo provedeno se všemi čtyřmi zvolenými druhy selekce. S každou z nich bylo provedeno 100 nezávislých běhů kartézského genetického programování. Pro rekapitulaci, byly tedy použity genotypy s 1500 funkčními uzly a mutační poměr 13 %. V případě evolučních strategií $(1 + \lambda)$ a $(1, \lambda)$ byla λ stanovena na hodnotu 4 a populace se tedy skládala z 5 jedinců. Velikost populace při testování turnajové selekce byla na základě testování (viz 6.1.2) stanovena na hodnotu 40. Ruletová selekce také pracovala s populací skládající se ze 40 jedinců.

V následujícím grafu 6.5 jsou znázorněny výsledky pro všechny typy selekce použité při evoluci ohnutých funkcí. Evoluční strategie $(1, \lambda)$ dosáhla v porovnání s ostatními výsledky velice rozptýlených hodnot. Pro názornější ukázku výsledků ostatních selekčních metod byla z grafu 6.6 tato evoluční strategie vynechána.

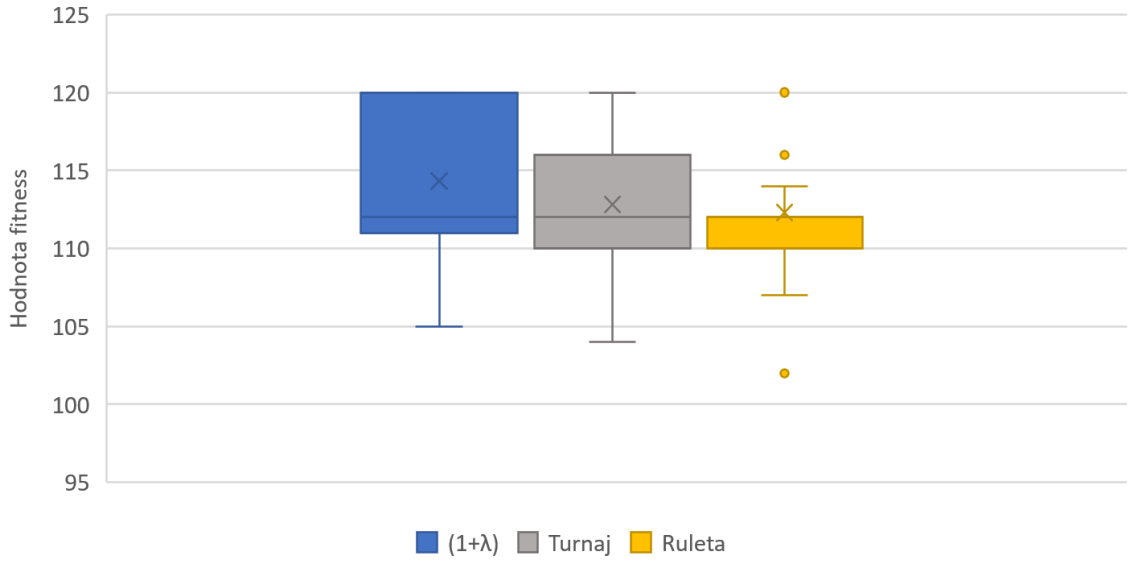


Obrázek 6.5: Výsledky zvolených selekčních metod při evoluci ohnutých funkcí

Z těchto grafů je vidět, že nejlepších výsledků bylo dosaženo při selekci pomocí evoluční strategie $(1 + \lambda)$. Selekce turnajem a ruletou dosáhla velice podobných výsledků. Průměr získaných hodnot fitness i rozptyl těchto hodnot se liší pouze nepatrně (viz tabulka 6.5), ale turnajová selekce dosáhla vyšší úspěšnosti, jinými slovy bylo nalezeno nalezeno více ohnutých funkcí.

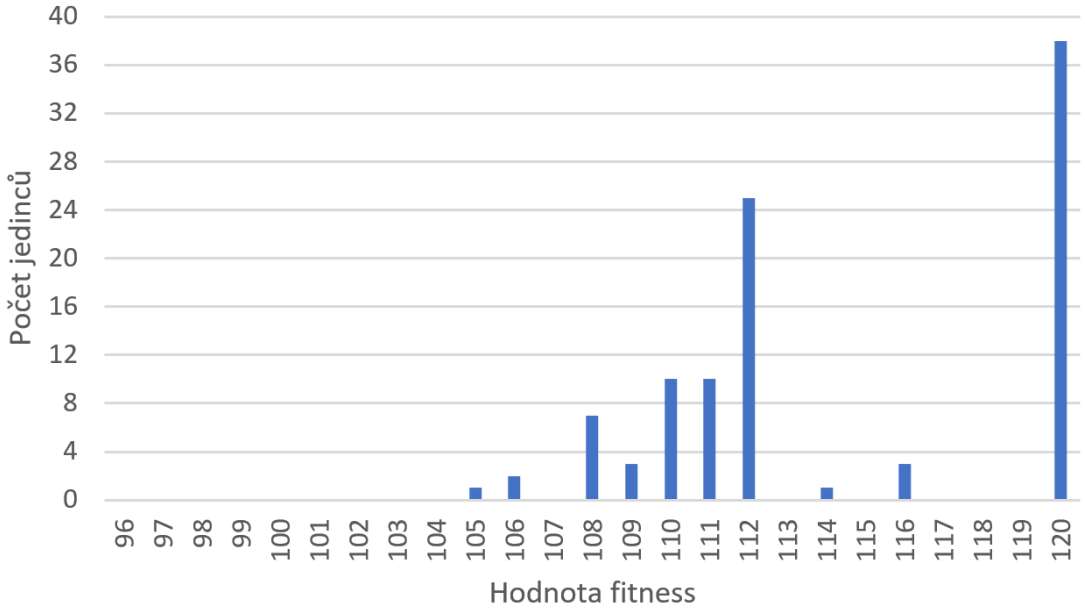
	$(1 + \lambda)$	$(1, \lambda)$	Turnaj	Ruleta
Průměr	114,32	57,91	112,81	112,29
Rozptyl	22,5576	933,3819	17,4339	13,5659
Úspěšnost	38 %	1 %	20 %	13 %

Tabulka 6.5: Průměrná fitness, rozptyl hodnot a úspěšnost jednotlivých velikostí populace

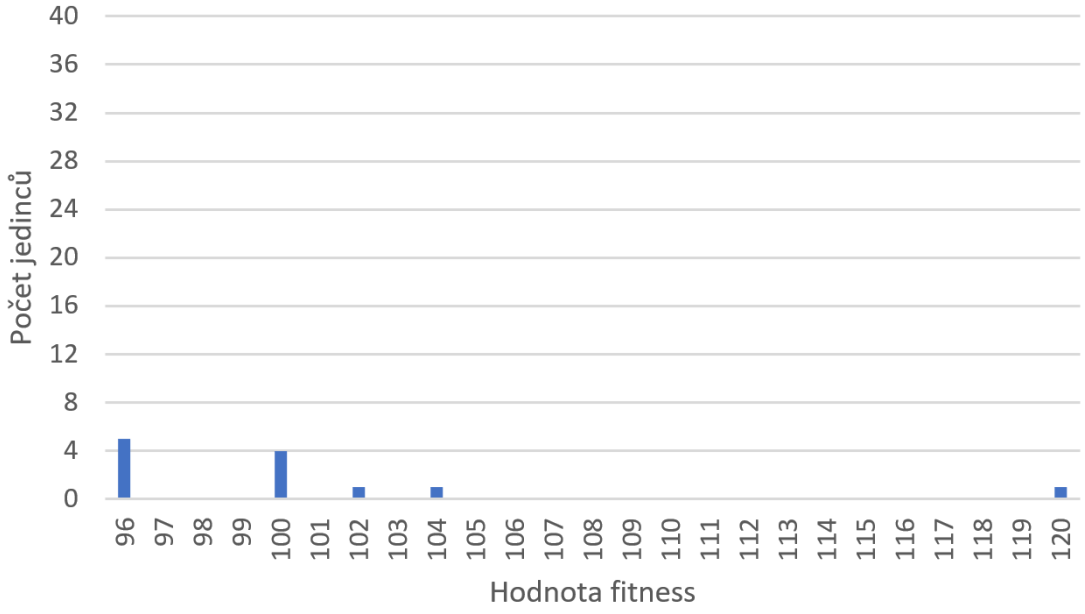


Obrázek 6.6: Výsledky zvolených selekčních metod při evoluci ohnutých funkcí (vynechána $(1, \lambda)$-ES)

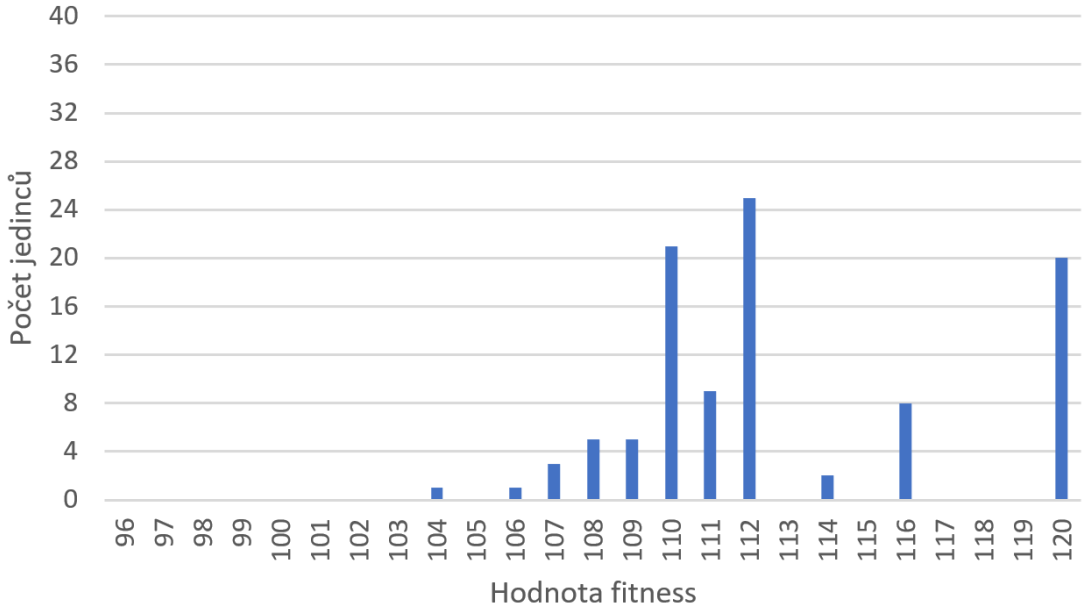
Na obrázcích 6.7, 6.8, 6.9 a 6.10 jsou zobrazeny grafy četností maximálních dosažených hodnot fitness pro všechny testované typy selekce. Z nasbíraných dat lze říci, že nejlepších výsledků bylo dosaženo s použitím selekce pomocí evoluční strategie $(1 + \lambda)$. Druhým nejlepším selekčním mechanismem (z testovaných typů selekce), se při evoluci ohnutých funkcí ukázala selekce turnajem. Na dalším místě skončila selekce ruletou, ale pouze mírným rozdílem. Je možné, že při vyšším počtu běhů nebo vyšším počtu evaluací fitness, by selekce pomocí rulety dosáhla lepších výsledků, než selekce turnajem. Pomocí evoluční strategie $(1, \lambda)$ byly nalezeny jedinci s fitness rovnou 0, ale i jeden jedince s fitness maximální. Nicméně průměrná hodnota fitness získaná při testování $(1, \lambda)$-ES nedosáhla ani poloviny v porovnání s ostatním metodami.



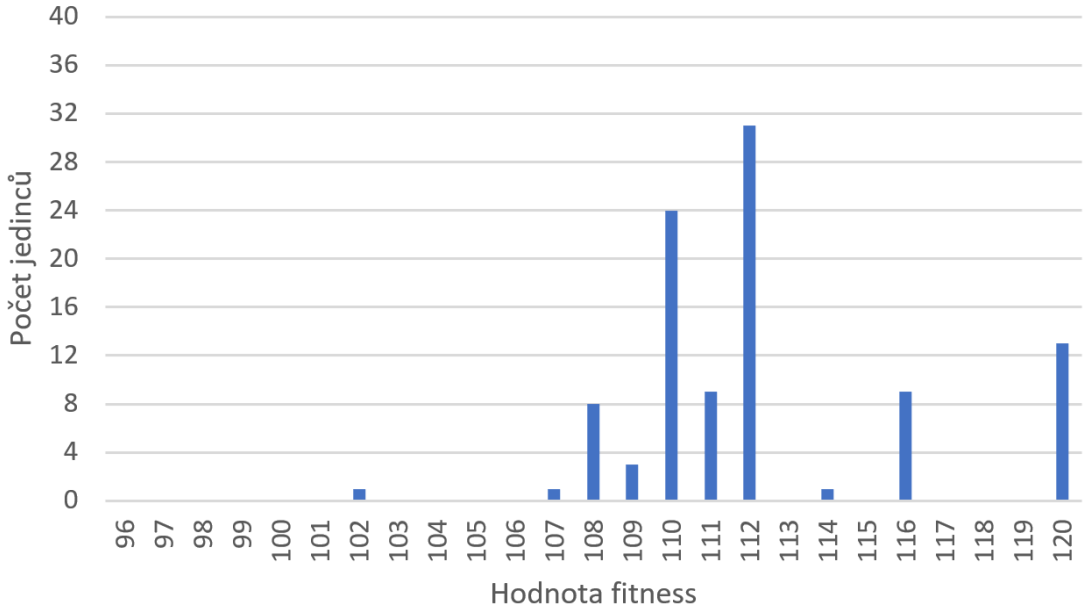
Obrázek 6.7: Rozložení fitness jedinců, ohnuté funkce, selekce $(1 + \lambda)$



Obrázek 6.8: Rozložení fitness jedinců, ohnuté funkce, selekce $(1, \lambda)$



Obrázek 6.9: Rozložení fitness jedinců, ohnuté funkce, selekce turnajem



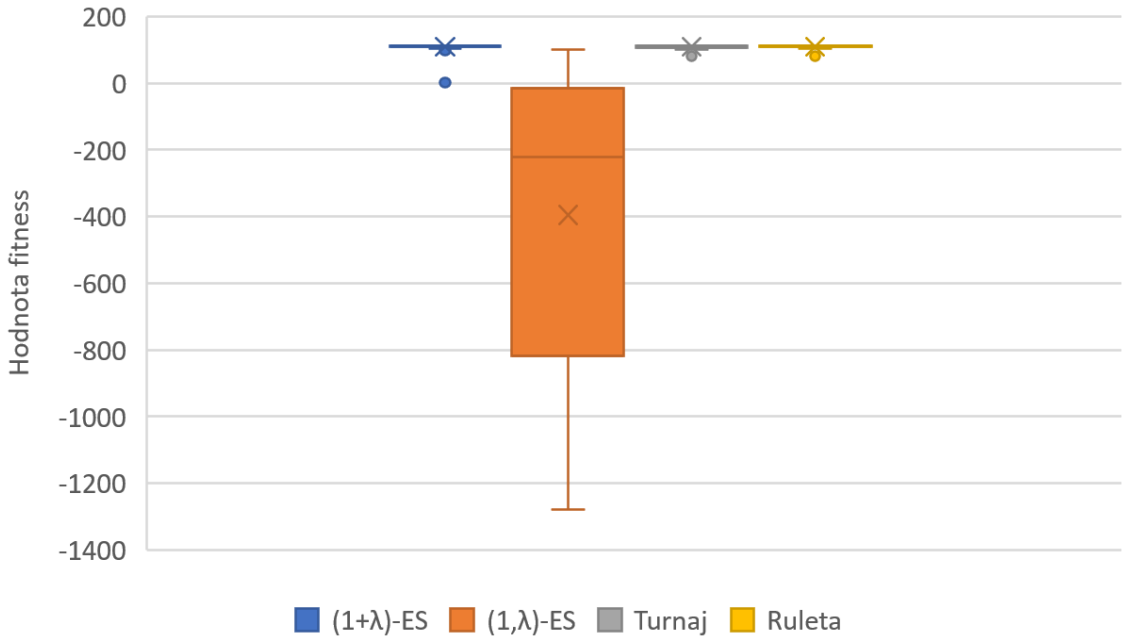
Obrázek 6.10: Rozložení fitness jedinců, ohnuté funkce, selekce ruletou

6.2.2 Vyvážené funkce s vysokou nelinearitou

Testy věnovány evoluci vyvážených funkcí s vysokou nelinearitou probíhaly stejně jako testy evoluce ohnutých funkcí. Se všemi zvolenými způsoby selekce bylo opět provedeno 100 nezávislých běhů a na základě jejich výsledků byly jednotlivé metody vyhodnoceny. Testy

byly spuštěny se stejnými velikostmi populace jako v případě ohnutých funkcí, s výjimkou selekce pomocí rulety, která byla testována s populací o velikosti 60 jedinců namísto 40.

V grafu 6.11 jsou znázorněny výsledky běhů kartézského genetického programování se zmíněnými způsoby selekce. Z tohoto grafu je viditelné, že selekce pomocí evoluční strategie, podobně jako v případě ohnutých funkcí, dosáhla nejhorších výsledků.



Obrázek 6.11: Výsledky zvolených selekčních metod při evoluci ohnutých funkcí

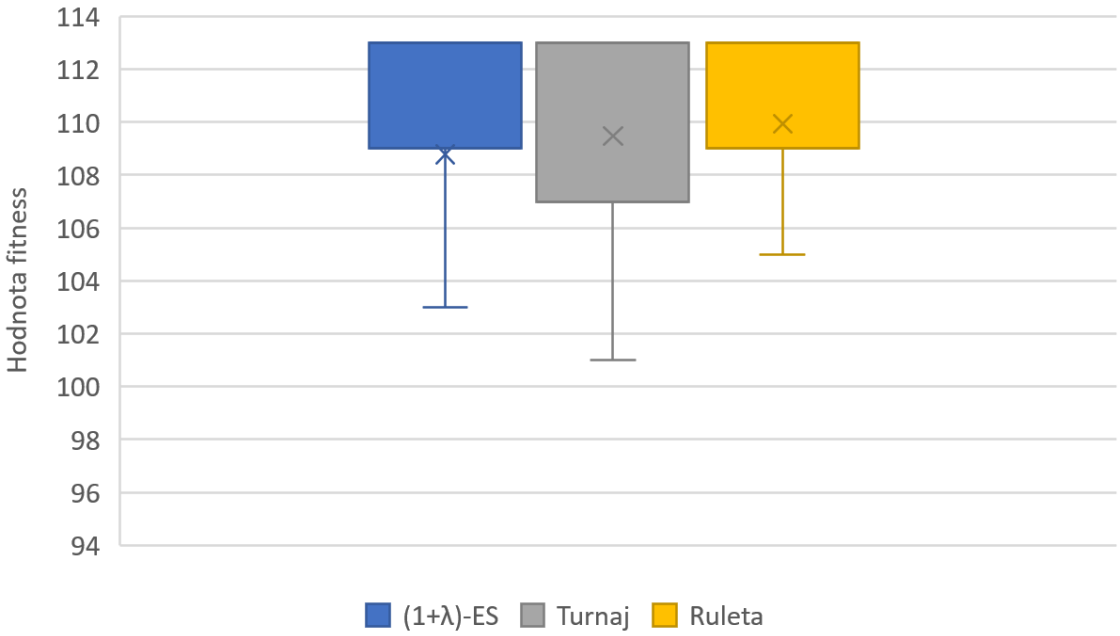
Graf 6.12 obsahuje stejná data jako graf 6.11 s tím rozdílem, že byla zanedbána výsledná data $(1, \lambda)$-ES a zároveň nejsou zobrazeny vzdálené body, aby byly výsledky ostatních selekčních metod čitelné. Z tohoto grafu již lze vyčíst, že selekce evoluční strategií $(1 + \lambda)$, turnajem i ruletou v žádném z běhů nenalezli funkci, která by dosahovala maximální hledané fitness. Nicméně každá z těchto metod dosáhla alespoň v 25 % případů fitness rovné hodnotě 113.

V tabulce 6.6 jsou uvedeny statistické údaje získané zpracováním výstupních dat provedených experimentů. Tato tabulka potvrzuje a upřesňuje výsledky uvedené v grafu 6.11.

	$(1 + \lambda)$	$(1, \lambda)$	Turnaj	Ruleta
Průměr	108,78	-395,83	109,48	109,94
Rozptyl	140,0716	201717,2	34,8896	25,1564
Úspěšnost	0 %	0 %	0 %	0 %

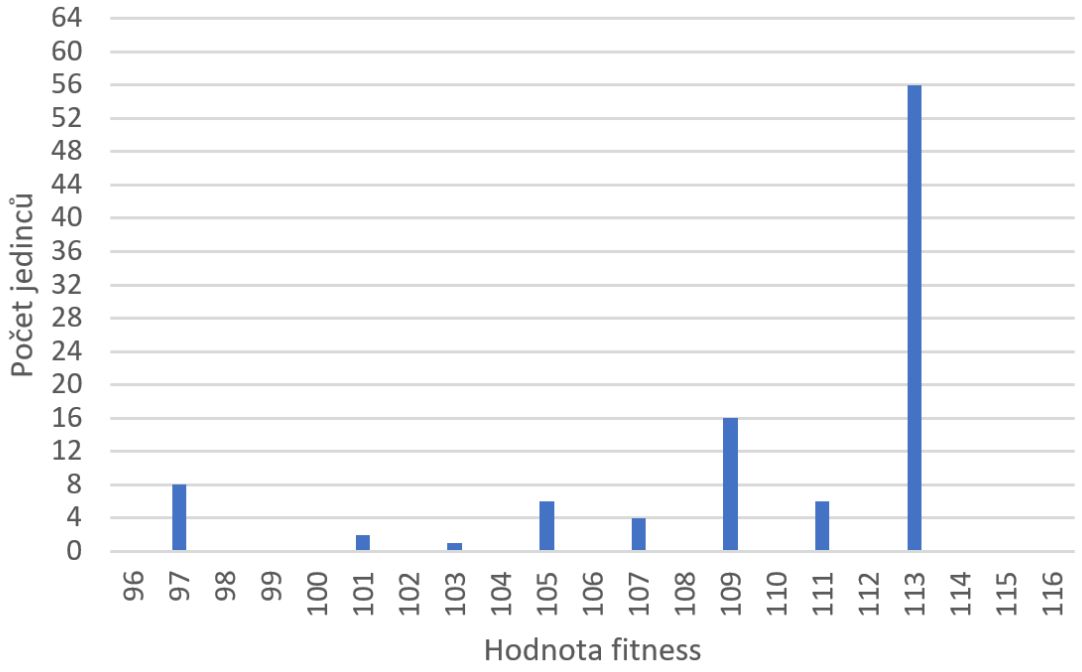
Tabulka 6.6: Průměrná fitness, rozptyl hodnot a úspěšnost jednotlivých velikostí populace

Histogramy 6.13, 6.14, 6.15 a 6.16 ukazují počet běhů, ve kterých nejlepší vyvinutý jedinec dosahoval příslušné hodnoty fitness. Jelikož v případě vyvážených funkcí s vysokou nelinearitou byly dosahované fitness jedinců více rozptýlené, než v případě ohnutých funkcí, jsou v histogramech nějaké hodnoty zanedbány.

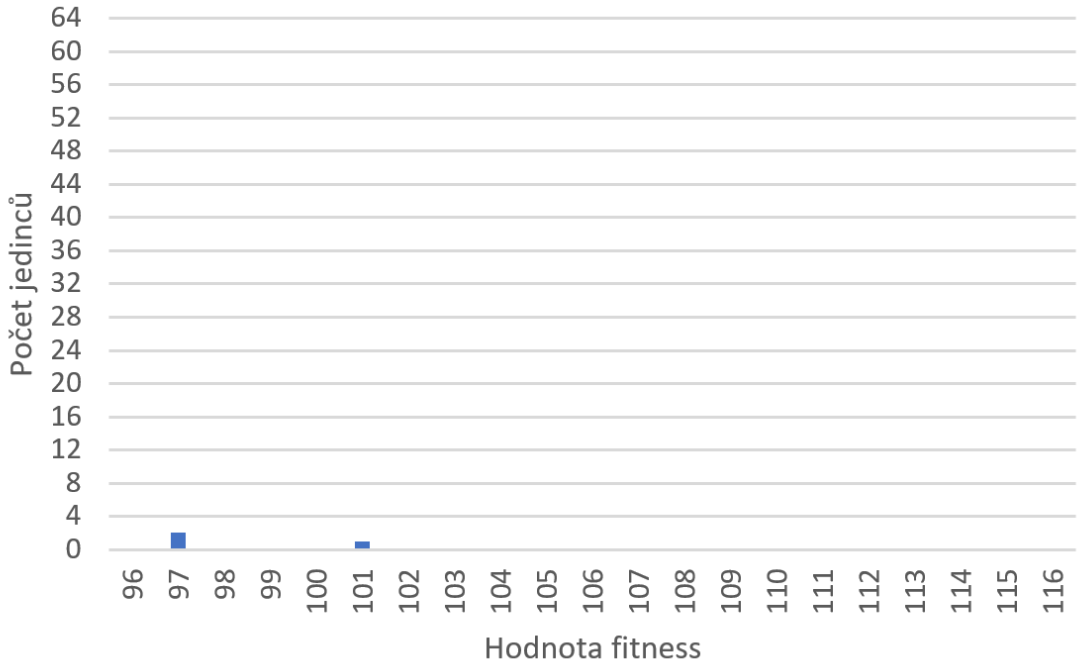


Obrázek 6.12: Výsledky zvolených selekčních metod při evoluci ohnutých funkcí (vynechána $(1, \lambda)$-ES)

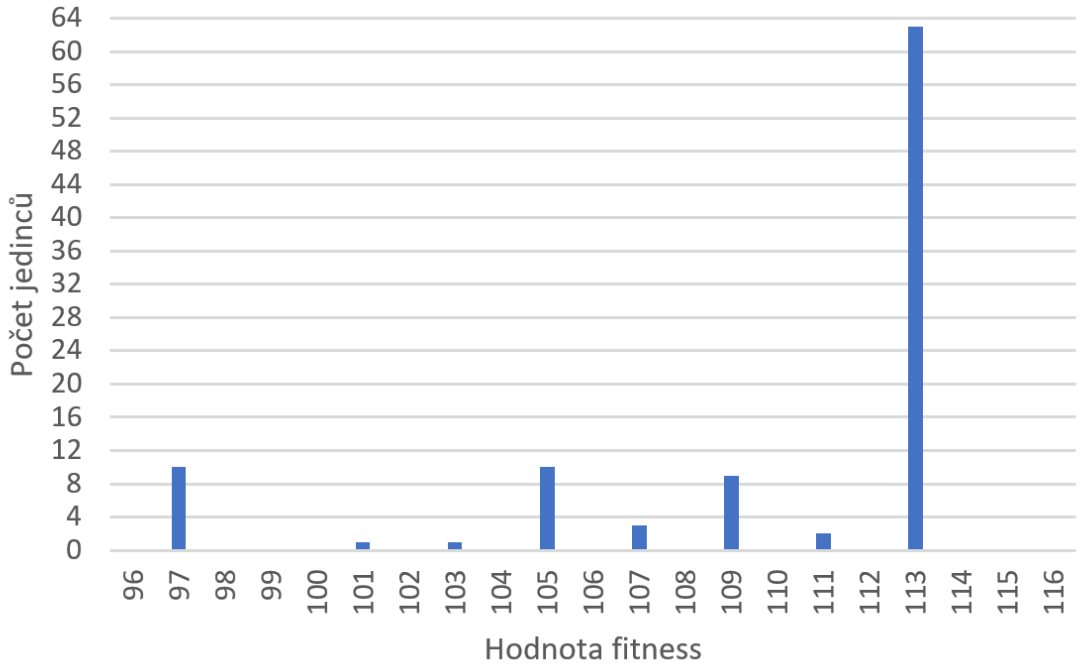
Na základě získaných dat lze tvrdit, že při aplikaci kartézského genetického programování na problému evoluce vyvážených funkcí s vysokou nelinearitou se jako nejlepší selekční metoda ukázala selekce pomocí rulety. Při použití turnajové selekce bylo dosaženo velice podobných výsledků. Na třetím místě se umístila evoluční strategie $(1 + \lambda)$, jejíž průměrná dosažená hodnota fitness byla mírně nižší a zároveň rozptyl těchto hodnot byl násobně vyšší, než v případě turnaje a rulety. Jako nejhorší volba se ukázala, stejně jako v případě ohnutých funkcí, evoluční strategie $(1, \lambda)$. V tomto případě byly nalezeny pouze tři funkce dosahující fitness vyšší než 96.



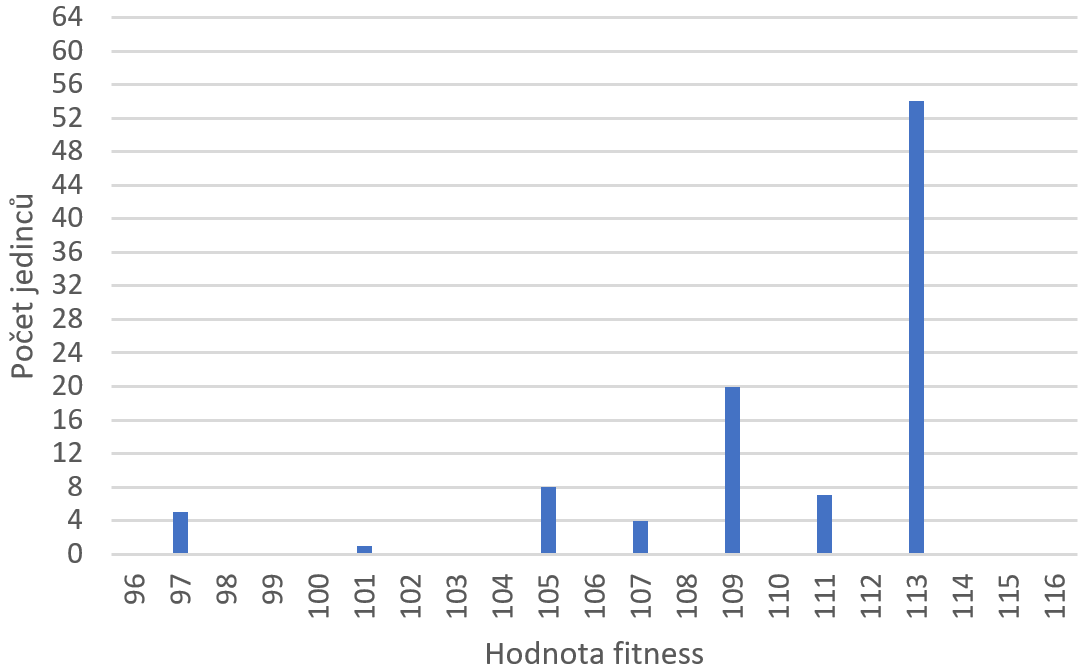
Obrázek 6.13: Rozložení fitness jedinců, vyvážené funkce s vysokou nelinearitou, selekce $(1 + \lambda)$



Obrázek 6.14: Rozložení fitness jedinců, vyvážené funkce s vysokou nelinearitou, selekce $(1, \lambda)$



Obrázek 6.15: Rozložení fitness jedinců, vyvážené funkce s vysokou nelinearitou, selekce turnajem



Obrázek 6.16: Rozložení fitness jedinců, vyvážené funkce s vysokou nelinearitou, selekce ruletou

Kapitola 7

Závěr

Tato práce byla zaměřena na evoluční návrh kryptograficky významných funkcí. Konkrétně se jednalo o využití kartézského genetického programování na problému evoluce ohnutých funkcí a vyvážených funkcí s vysokou nelinearitou. Cílem této práce bylo porovnat různé selekční mechanismy, konkrétně evoluční strategie $(1 + \lambda)$ a $(1, \lambda)$ a selekce pomocí turnaje a rulety, použité při aplikaci kartézského genetického programování na problému evoluce ohnutých funkcí a vyvážených funkcí s vysokou nelinearitou.

V kapitole 2 byly popsány booleovské funkce, jejich kryptograficky významné vlastnosti a také jejich použití při generování bezpečných pseudonáhodných posloupností pomocí LFSR. V následující kapitole 3 byly uvedeny informace k evolučním algoritmům a následně i detailnější popis kartézského genetického programování. Kapitola 4 obsahuje návrh systému schopného generovat různé druhy kryptograficky významných booleovských funkcí. Navržený systém ke generaci funkcí používá kartézské genetické programování s různými druhy selekce, které jsou v této kapitole rovněž specifikovány. V kapitole 5 je popsána implementace navrženého systému v jazyce C++. Popsány jsou především stěžejní části navrženého systému jako je reprezentace jednotlivých uzlů, ze kterých se skládá genotyp, generování nových jedinců, způsob dekódování chromozomu z genotypu, mutace jedinců a nakonec implementace zvolených selekčních mechanismů. V kapitole 6 jsou popsány experimenty provedené s implementovaným systémem, konkrétně se jedná o optimalizační testy turnajové a ruletové selekce a rozsáhlejší testy provedené se všemi zvolenými typy selekce. Výsledky těchto testů a jejich vyhodnocení se nachází také v poslední uvedené kapitole. Tím byly splněny všechny body zadání.

V práci by mohl ještě někdo pokračovat rozšířením implementovaného systému o možnost generování booleovských funkcí s pokročilejšími kryptografickými vlastnostmi např. funkce s korelační imunitou nebo odolné funkce. Dále se nabízí možnost zefektivnění a zrychlení evoluce nových generací.

Literatura

[1] Bidlo, M.: Aplikované evoluční algoritmy, Základní pojmy a techniky EA. přednáška, 2018.

[2] Carlet, C.: Boolean functions for cryptography and error correcting codes. *Boolean Methods and Models*, 2006.

[3] Chowdhury, S.; Maitra, S.: Efficient software implementation of LFSR and boolean function and its application in nonlinear combiner model. In *International Conference on Applied Cryptography and Network Security*, Springer, 2003, s. 387–402.

[4] Courtois, N. T.: Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology - CRYPTO 2003*, editace D. Boneh, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, ISBN 978-3-540-45146-4, s. 176–194.

[5] cplusplus.com: std::stable_sort. Accessed: 2019-05-08.
URL https://en.cppreference.com/w/cpp/algorithm/stable_sort

[6] cplusplus.com: std::vector::erase. Accessed: 2019-05-08.
URL https://en.cppreference.com/w/cpp/container/vector/erase

[7] Darwin, C.: *On the origin of species, 1859*. Routledge, 2004.

[8] George, M.; Alfke, P.: Linear feedback shift registers in virtex devices. *Xilinx apprication note XAPP210*, 2007.

[9] Goldreich, O.: *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[10] Horký, M.: Minimalizace logických funkcí. 2009.

[11] Hynek, J.: *Genetické algoritmy a genetické programování*. Průvodce, Praha: Grada, první vydání, 2008, ISBN 978-80-247-2695-3.

[12] Koza, J. R.: *Genetic programming : On the programming of computers by means of natural selection*. Complex adaptiev systems, Cambridge : London,: Bradford Book, MIT Press, 1992, ISBN 0-262-11170-5.

[13] Maitra, S.; Sarkar, P.: Highly nonlinear resilient functions optimizing Siegenthaler's inequality. In *Annual International Cryptology Conference*, Springer, 1999, s. 198–215.

[14] Mesnager, S.: *Boolean Functions and Cryptography*. Cham: Springer International Publishing, 2016, ISBN 978-3-319-32595-8, doi:10.1007/978-3-319-32595-8_3.
URL https://doi.org/10.1007/978-3-319-32595-8_3

[15] Miller, J. F.: *Cartesian Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, ISBN 978-3-642-17310-3, doi:10.1007/978-3-642-17310-3_2.
URL https://doi.org/10.1007/978-3-642-17310-3_2

[16] Murdock, J.: Normal forms. *Scholarpedia*, ročník 1, č. 10, 2006, ISSN 1941-6016.

[17] Paar, C.; Pelzl, J.: *Stream Ciphers*, kapitola 2. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ISBN 978-3-642-04101-3, s. 29–54, doi:10.1007/978-3-642-04101-3_2.
URL https://doi.org/10.1007/978-3-642-04101-3_2

[18] Petrowski, A.; Ben Hamida, S.: *Evolutionary Algorithms*, kapitola 6. Cham: Springer International Publishing, 2016, ISBN 978-3-319-45403-0, s. 115–178, doi:10.1007/978-3-319-45403-0_6.
URL https://doi.org/10.1007/978-3-319-45403-0_6

[19] Pfahringer, B.: *Conjunctive Normal Form*, kapitola C. Boston, MA: Springer US, 2017, ISBN 978-1-4899-7687-1, s. 260–261, doi:10.1007/978-1-4899-7687-1_158.
URL https://doi.org/10.1007/978-1-4899-7687-1_158

[20] Pfahringer, B.: *Disjunctive Normal Form*, kapitola D. Boston, MA: Springer US, 2017, ISBN 978-1-4899-7687-1, s. 371–372, doi:10.1007/978-1-4899-7687-1_223.
URL https://doi.org/10.1007/978-1-4899-7687-1_223

[21] Picek, S.; Carlet, C.; Jakobovic, D.; aj.: Correlation immunity of boolean functions: an evolutionary algorithms perspective. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ACM, 2015, s. 1095–1102.

[22] Picek, S.; Jakobovic, D.; Miller, J. F.; aj.: Cryptographic Boolean functions: One output, many design criteria. *Applied Soft Computing*, ročník 40, 2016: s. 635–653.

[23] Poli, R.; Langdon, W. B.; McPhee, N. F.; aj.: *A field guide to genetic programming*. S.l.: Lulu Press], 2008, ISBN 978-1-4092-0073-4.

[24] Sarkar, P.; Maitra, S.: Construction of nonlinear Boolean functions with important cryptographic properties. In *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2000, s. 485–506.

[25] Sekanina, L.; Vašíček, Z.; Růžička, R.; aj.: *Evoluční hardware: Od automatického generování patentovatelných invencí k sebemodifikujícím se strojům*. Edice Gerstner, Academia, 2009, ISBN 978-80-200-1729-1, 328 s.
URL http://www.fit.vutbr.cz/research/view_pub.php.cz.iso-8859-2?id=9123

[26] Siegenthaler, T.: Correlation-immunity of nonlinear combining functions for cryptographic applications (Corresp.). *Information Theory, IEEE Transactions on*, ročník 30, č. 5, 1984: s. 776–780, ISSN 0018-9448.

[27] Vaudenay, S.: *Prehistory of Cryptography*, kapitola 1. Boston, MA: Springer US, 2006, ISBN 978-0-387-25880-5, s. 1–20, doi:10.1007/0-387-25880-9_1.
URL https://doi.org/10.1007/0-387-25880-9_1

[28] Wu, C.-K.; Feng, D.; aj.: *Boolean functions and their applications in cryptography*. Springer, 2016.