VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

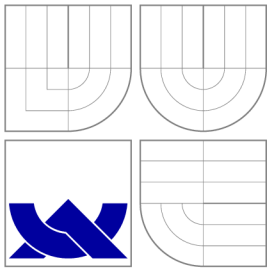
KRYPTOANALÝZA SYMETRICKÝCH ŠIFROVACÍCH ALGORITMŮ S VYUŽITÍM SYMBOLICKÉ REGRESE A GENETICKÉHO PROGRAMOVÁNÍ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

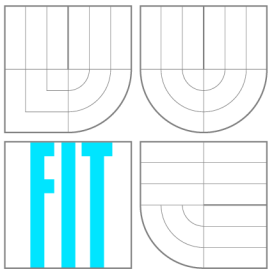
AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ SMETKA

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

KRYPTOANALÝZA SYMETRICKÝCH ŠIFROVACÍCH ALGORITMŮ S VYUŽITÍM SYMBOLICKÉ REGRESE A GENETICKÉHO PROGRAMOVÁNÍ

CRYPTANALYSIS OF SYMMETRIC ENCRYPTION ALGORITHMS USING SYMBOLIC REGRESSION AND GENETIC PROGRAMMING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ SMETKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. IVAN HOMOLIAK

BRNO 2015

Abstrakt

Tato diplomová práce se zabývá kryptoanalýzou symetrických šifrovacích algoritmů. Cílem práce je ukázat jiný úhel pohledu na tuto problematiku. Odlišný způsob oproti současným metodám spočívá ve využití síly evolučních principů, které jsou v kryptoanalytickém systému aplikovány pomocí genetického programování. V teoretické části je popsána kryptografie a kryptoanalýza symetrických šifrovacích algoritmů a genetické programování. Ze získaných informací je dále představen návrh kryptoanalytického systému, který využívá evoluční principy. Praktická část se zabývá implementací symetrického šifrovacího algoritmu, lineární kryptoanalýzou a simulačním nástrojem genetického programování. Závěr práce prezentuje experimenty s navrženým kryptoanalytickým systémem využívající genetické programování a zhodnocuje dosažené výsledky.

Abstract

This diploma thesis deals with the cryptanalysis of symmetric encryption algorithms. The aim of this thesis is to show different point of view on this issues. The dissimilar way, compared to the recent methods, lies in the use of the power of evolutionary principles which are in the cryptanalytic system applied with help of genetic programming. In the theoretical part the cryptography, cryptanalysis of symmetric encryption algorithms and genetic programming are described. On the ground of the obtained information a project of cryptanalytic system which uses evolutionary principles is represented. Practical part deals with implementation of symmetric encrypting algorithm, linear cryptanalysis and simulation instrument of genetic programming. The end of the thesis represents experiments together with projected cryptanalytic system which uses genetic programming and evaluates reached results.

Klíčová slova

Kryptografie, informační bezpečnost, symetrické šifrovací algoritmy, útok hrubou silou, lineární kryptoanalýza, diferenciální kryptoanalýza, umělá inteligence, strojové učení, genetické programování, automaticky definované funkce, fitness funkce, symbolická regrese

Keywords

Cryptography, information security, symmetric encryption algorithms, brute force attack, linear cryptanalysis, differential cryptanalysis, artificial intelligence, machine learning, genetic programming, automatically defined functions, fitness function, symbolic regression

Citace

Tomáš Smetka: Kryptoanalýza symetrických šifrovacích algoritmů s využitím symbolické regrese a genetického programování, diplomová práce, Brno, FIT VUT v Brně, 2015

Kryptoanalýza symetrických šifrovacích algoritmů s využitím symbolické regrese a genetického programování

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Ivana Homoliaka

.....
Tomáš Smetka
27. května 2015

Poděkování

Chtěl bych poděkovat svému vedoucímu panu Ing. Ivanu Homoliakovi za odborné vedení a za cenné rady při tvorbě této práce. Dále bych chtěl poděkovat své rodině za podporu při studiu.

Obsah

Kapitola 1

Úvod

Informace, jejich nalezení a pochopení jsou základní stavební kámen učení a rozhodování ve všech oblastech lidských aktivit. Jsou proto absolutně nezbytné v současné digitální éře a 21. století se díky tomu nazývá také jako století informací.Kdo má přístup k informacím, dokáže je interpretovat a vhodně využít, ten vítězí.

V současné době se proto více než kdy dříve řeší otázka bezpečnosti, se kterou je spojeno utajení informací. Utajení informací prostupuje všemi oblastmi současné společnosti od státní až po civilní sféru. K účelu utajení jsou využívané různé šifrovací algoritmy s rozdílnými vlastnostmi. Kompromitace porušením utajení, integrity nebo dostupnosti informací může mít negativní dopady pro dotčenou stranu. V některých případech může vést ke ztrátě soukromí či konkurenceschopnosti, v jiných případech dokonce ke konfliktu mezi státy. Citlivost informací je tak přímo úměrná škodám vzniklým jejich kompromitací. U šifrovacích algoritmů je tedy kladen hlavní důraz na zajištění jejich bezpečnosti.

Určení, zda je šifrovací algoritmus bezpečný, je složitá procedura. Nelze jednoznačně říct, že jeden šifrovací algoritmus je lepší než druhý, protože neexistuje exaktní ohodnocení jejich kvality. V některých případech dokonce prokáže kvality algoritmu až čas. Existuje však oblast, která se zabývá metodami získávání obsahu šifrovaných informací bez znalosti klíče a tím potažmo hodnotí i kvalitu šifrovacího algoritmu. Tato věda se nazývá kryptoanalýza.

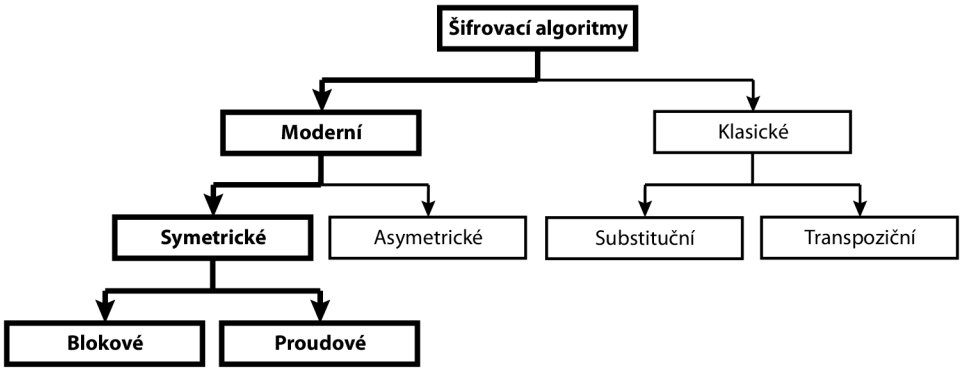
Standardní kryptoanalytické metody symetrických šifrovacích algoritmů jsou založené na principu nalezení klíče, který byl použit pro utajení informace. Využívají hrubé síly, nebo hledají slabiny šifrovacích algoritmů ve snaze zkrátit dobu nalezení klíče. Tato diplomová práce má za cíl ukázat, že jsou i jiné přístupy a metody, kterými lze kryptoanalýzu aplikovat. Práce bude prezentovat využití hnacího motoru vývoje všech živých organismů na Zemi, kterým je evoluce. Evoluční mechanismy budou aplikovány pomocí genetického programování, jehož výsledkem bude nalezení programu, který modeluje chování symetrického šifrovacího algoritmu. Pokud se hledaný model podaří zkonstruovat, měl by být schopen dešifrovat nové zprávy, které byly utajeny modelovaným šifrovacím algoritmem.

Text diplomové práce je rozdělen do sedmi hlavních kapitol. Kapitola č. 2 popisuje teoretické znalosti z oblasti kryptografie a kryptoanalýzy symetrických šifrovacích algoritmů. Kapitola č. 3 popisuje principy genetického programování. Ze získaných teoretických znalostí je v kapitole č. 4 představen možný kryptoanalytický systém, který využívá principů genetického programování. Kapitola č. 5 popisuje návrh a implementaci představeného systému. Jeho součástí je implementace algoritmu DES, lineární kryptoanalýza a simulační nástroj využívající genetické programování. Kapitola č. 6 se věnuje samotné kryptoanalýze algoritmu DES s využitím genetického programování. Poslední kapitola č. 7 představuje závěr, shrnuje dosažené výsledky a uvádí možná rozšíření diplomové práce.

Kapitola 2

Symetrická kryptologie

Kryptologie je matematická vědní disciplína, která se zabývá aplikací vzorců a algoritmů s cílem informace utajit a možnostmi, jak je následně opět získat. Dělí se na kryptografii a kryptoanalýzu. Zájmem kryptografie je utajení, zabezpečení a identifikace informací. Kryptoanalýza je tvořena postupy a principy, které se snaží kryptografií vyvinuté šifrovací algoritmy oslabit nebo prolomit [28, 35]. Kryptologie je tedy oblast vědy, ve které se setkávají 2 skupiny lidí. První skupina se snaží informace utajit a druhá se je snaží získat. Výhodou tohoto přirozeného procesu je, že se oblast bezpečnosti neustále zdokonaluje.



Obrázek 2.1: Rozdělení šifrovacích algoritmů.

V této práci se zaměříme na tučně vyznačenou část rozdělení šifer podle diagramu 2.1, tedy na kryptografii 2.1 a kryptoanalýzu 2.3 moderních symetrických šifrovacích algoritmů.

2.1 Kryptografie symetrických algoritmů

Moderní kryptografie je studium matematických technik s pevnými formálními základy, které jsou spojeny s aspekty informační bezpečnosti. Těmito aspekty jsou důvěrnost, integrita, autenticita a nepopiratelnost [28]. Její počátky sahají do 70. let 20. století, kdy s rozšiřováním výpočetní síly vznikaly požadavky na zajištění bezpečnostních cílů, viz 2.1.1. Prakticky se jedná o matematické modely, které převádí otevřená data do nečitelné podoby na základě utajené informace. Zároveň jsou tyto modely navrhovány tak, aby byly odolné vůči kryptoanalytickým metodám.

Symetrická kryptografie je typem kryptografie, kde komunikující subjekty používají stejný kryptografický klíč $K = K'$ nebo též tajný klíč. Znalost tajného klíče K může kromě důvěrnosti zajišťovat i další aspekt informační bezpečnosti, kterým je důkaz identity [14].

Matematické modely symetrických šifrovacích algoritmů aplikují oproti modelům asymetrickým jednodušší matematické operace, kterými jsou např. nonekvivalence a bitový součet. Díky tomu jsou násobně rychlejší a zároveň je možná jejich jednoduchá implementace na hardwarové úrovni.

Symetrické šifrovací algoritmy se dělí na blokové a proudové [14, 39]:

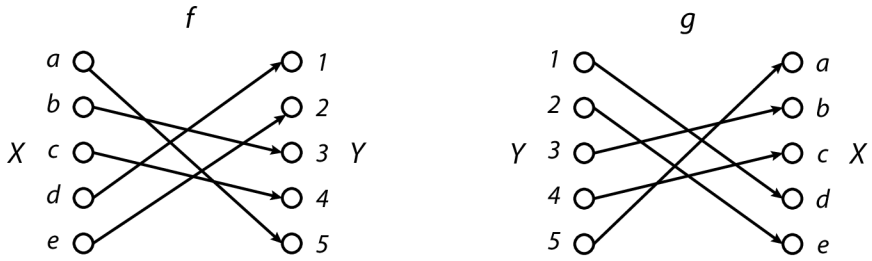
- **Blokové šifry** - zpracovávají postupně otevřený text P po blocích P_i, které mají definovanou velikost a konstruují z nich šifrované bloky C_i o stejné velikosti. Typická velikost bloku je 64 nebo 128 bitů.
- **Proudové šifry** - provádí postupné šifrování dat po bitech popř. jiné předem definované délce (př. byte). Šifrování probíhá s využitím klíčů K_1, K_2, ..., které jsou generovány jako „proud“.

Základem symetrických šifrovacích algoritmů je substituce a permutace, které jsou definovány následovně [28]:

Substituce

Definice 2.1. *Substituce* je funkce f, která je bijektivním zobrazením prvků z množiny X na prvky z množiny Y.

Definice 2.2. Pokud je substituce f bijektivním zobrazením, pak můžeme také definovat bijektivní zobrazení g z Y do X následovně: $\forall y \in Y$, $g(y) = x$, kde $x \in X$ a $f(x) = y$. Pak se funkce g nazývá funkcí inverzní k funkci f a je zapisována jako $g = f^{-1}$.



Obrázek 2.2: Substituční funkce f a její inverzní funkce $g = f^{-1}$.

Příklad 2.1. (*inverzní funkce*) Nechť $X = \{a, b, c, d, e\}$ a $Y = \{1, 2, 3, 4, 5\}$. Funkce f je dána předpisem (pomocí šipek) na obrázku 2.2. Funkce f je bijektivní zobrazení a její inverzní funkce g je pak formována jednoduše pomocí otočení směru šipek na druhé konce.

V kryptografii je bijektivní vlastnost substituční funkce f velice důležitá. Pomocí ní šifrovací algoritmus provádí šifrování a pomocí její inverzní funkce naopak dešifrování [28].

Permutace

Definice 2.3. Nechť $\mathcal{S}$ je konečná množina prvků. Pak permutace p na množině $\mathcal{S}$ je bijektivní zobrazení $p : \mathcal{S} \to \mathcal{S}$.

Příklad 2.2. (*permutace*) Nechť $\mathcal{S} = \{1, 2, 3, 4, 5\}$. Permutace $p : \mathcal{S} \to \mathcal{S}$ je definovaná následovně:

$$p(1) = 3, p(2) = 5, p(3) = 4, p(4) = 2, p(5) = 1$$

Permutace může být popsána různými způsoby. Může být zobrazena např. pomocí matice:

$$p = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 4 & 2 & 1 \end{pmatrix}$$

kde horní řádek matice reprezentuje doménu a spodní řádek matice její obraz na základě mapování p.

Vzhledem k tomu, že permutace jsou bijekce, mají své inverzní funkce. Uvažujme předchozí matici, pak inverzní funkce k permutaci p je:

$$p^{-1} = \begin{pmatrix} 3 & 5 & 4 & 2 & 1 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}.$$

V této podkapitole budou následně detailně prezentovány cíle informační bezpečnosti 2.1.1, dále princip symetrického šifrovacího modelu 2.1.2 a také principy činnosti blokových šifer 2.1.3.

2.1.1 Kryptografické bezpečnostní cíle

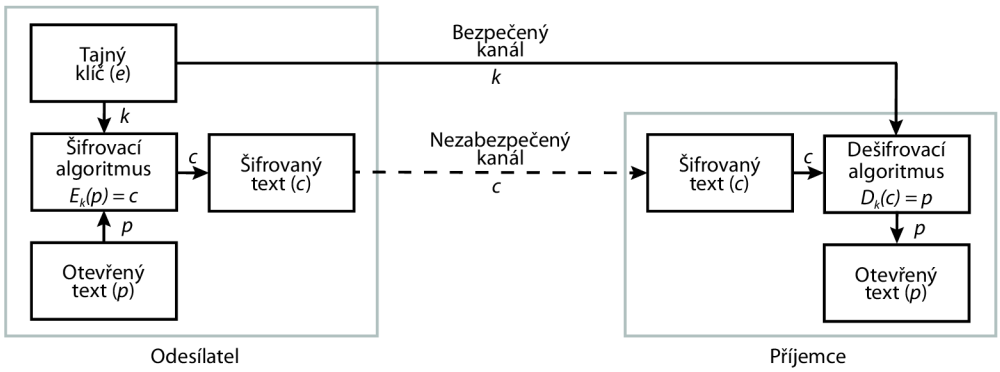
V soudobém chápání bezpečnosti informací je bezpečnost dána zajištěním čtyř základních cílů [28, 39, 14]:

- **Důvěrnost** - je služba, která zajišťuje, že obsah informace bude dostupný pouze oprávněným osobám. Pojem *tajemství* je v našem případě chápáno jako synonymum pro důvěrnost a soukromí. V rámci šifrovacích algoritmů vyjadřuje důvěrnost matematickou ochranu informací, která je aplikací algoritmu činí nečitelné.
- **Integrita** - je služba, která se zaměřuje na neoprávněné změny dat. K zajištění integrity musíme mít schopnost detekovat, zda s daty nemanipuloval neoprávněný subjekt. Za manipulaci s daty považujeme operace vložení, mazání a nahrazení.
- **Autenticita** - je vlastnost, která nám umožňuje ověřit identitu subjektu. Jinými slovy, že subjekty jsou těmi, za které se vydávají a že každá příchozí informace tak pochází z důvěryhodného zdroje.
- **Nepopiratelnost** - je služba, která zabraňuje subjektu popřít předchozí události či akce. Této služby se využívá v rámci sporů, aby bylo možné určit, zda se událost nebo činnost vyskytla či nikoliv.

2.1.2 Symetrický šifrovací model

Schéma symetrického šifrování vyžaduje pro možnou funkcionalitu pět základních složek [39], jejichž použití je zobrazeno na diagramu 2.3.

1. **Otevřený text** - Plaintext P je originální otevřená zpráva nebo data, která jsou vstupem šifrovacího algoritmu.
2. **Šifrovací algoritmus** - je algoritmus, který provádí různé substituce a transformace nad otevřeným textem.
3. **Tajný klíč** - secret Key K je také vstupem šifrovacího algoritmu. Klíč je nezávislá hodnota vzhledem k otevřenému textu a algoritmu. Příslušné substituce a transformace prováděné algoritmem jsou závislé na klíči. Algoritmus tedy produkuje odlišné výstupy na základě specifikace klíče.
4. **Šifrovaný text** - Cyphertext C je výsledkem šifrování otevřeného textu P zvoleným šifrovacím algoritmem a příslušným tajným klíčem K. Pro dva různé klíče K bude šifrovací algoritmus pro stejný otevřený text P generovat dva různé šifrované texty C. Šifrovaný text je z pohledu útočníka náhodný proud dat, který je ve své podobě nesrozumitelný.
5. **Dešifrovací algoritmus** - je v zásadě původní šifrovací algoritmus, který pro dešifrování používá reverzní princip. Vstupem je šifrovaný text C a příslušný tajný klíč K. Výstupem je pak originální otevřený text P.



Obrázek 2.3: Model symetrického kryptosystému.

Na bezpečné používání konvenčního šifrování máme dva základní požadavky [39]:

1. Potřebujeme silný šifrovací algoritmus a to takový, že útočník, který zná algoritmus a má přístup k jednomu nebo více šifrovaným textům C, nebude schopný text dešifrovat nebo získat tajný klíč K. Tento požadavek existuje i v silnější formě. Útočník by neměl být schopný dešifrovat šifrovaný text C nebo získat tajný klíč K i v případě, že vlastní řadu šifrovaných textů C a k nim odpovídající otevřené texty P.
2. Odesílatel i příjemce si musí vyměnit kopie tajného klíč K v bezpečném módu (např. skrz zabezpečený komunikační kanál). Tajný klíč K musí oba subjekty držet v bezpečí. Pokud by útočník získal tajný klíč K a měl znalost o typu šifrovacího algoritmu, všechna komunikace mezi subjekty by pro něj byla čitelná.

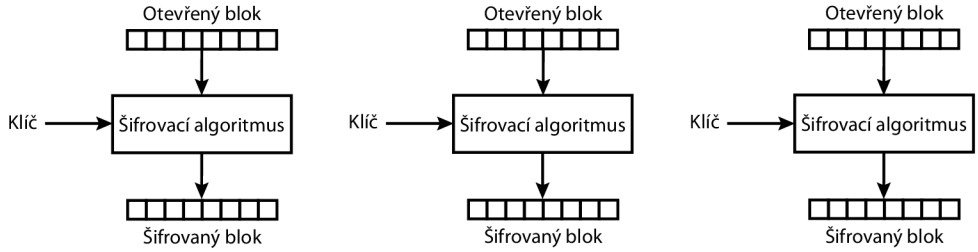
2.1.3 Režimy činnosti blokových a proudových šifer

Symetrické šifrovací algoritmy mohou pracovat v různých režimech [14].

Režim ECB (Electronic Code Book)

Režim tzv. elektronické kódovací knihy dělí otevřený text P (Plaintext) na bloky P_i o délce N, která odpovídá délce bloku šifrovacího algoritmu. Tímto algoritmem je každý blok P_i samostatně šifrován a to za použití stejného klíče. Výsledné zašifrované bloky jsou spojeny do výsledné zprávy C (Ciphertext). Proces dešifrování probíhá opačným způsobem. Zpráva C je rozdělena na bloky a ty jsou stejným algoritmem a klíčem dešifrovány každý samostatně. Výsledky dešifrovaných bloků jsou spojeny do původní otevřeného textu P.

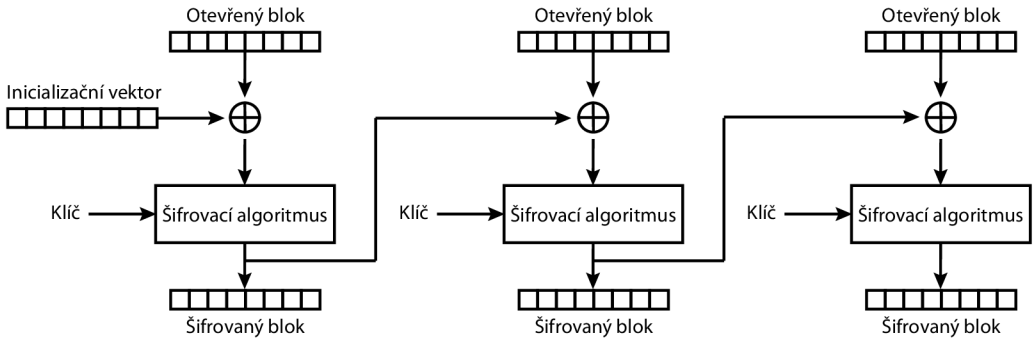
Režim ECB na rozdíl od následujících režimů nezajišťuje dostatečnou kontrolu integrity dat [37]. Díky tomu může útočník do zašifrované zprávy vložit popř. vyjmout bloky dat. Princip činnosti ECB je zobrazen diagramem 2.4.



Obrázek 2.4: Režim Electronic Code Book.

Režim CBC (Cipher Block Chaining)

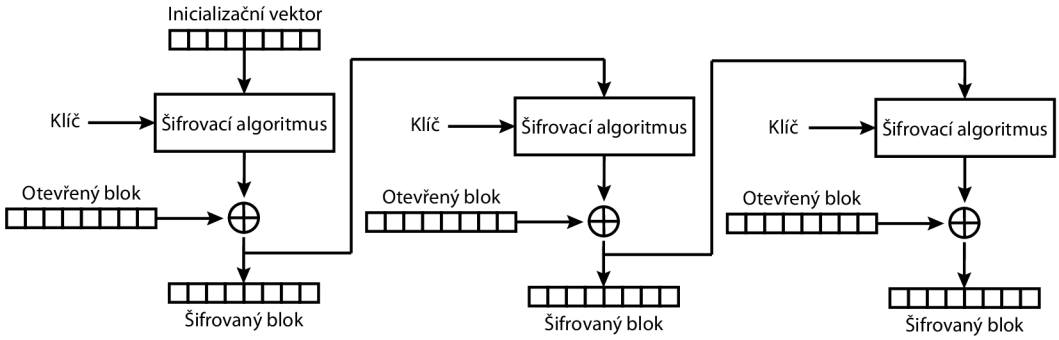
Režim CBC neboli řetězení bloků šifry pracuje podobně jako režim ECB. Otevřený text P je rozdělen na bloky P_i, jejichž velikosti odpovídá délce bloku zvoleného algoritmu. Následně je každý blok zašifrován samostatně. Na rozdíl od ECB je na každý blok P_i před samotným šifrováním aplikována operace XOR, jejímž druhým argumentem je výsledek šifrování předchozího bloku C_{i-1}. Pro první blok zprávy P_1 je operace XOR provedena s pomocí tzv. *inicializačního vektoru* (IV). Režim CBC je nejpoužívanějším režimem při šifrování dlouhých zpráv. Graficky je jeho princip popsán diagramem 2.5.



Obrázek 2.5: Režim Cypher Block Chaining.

Režim CFB (Cyphertext FeedBack)

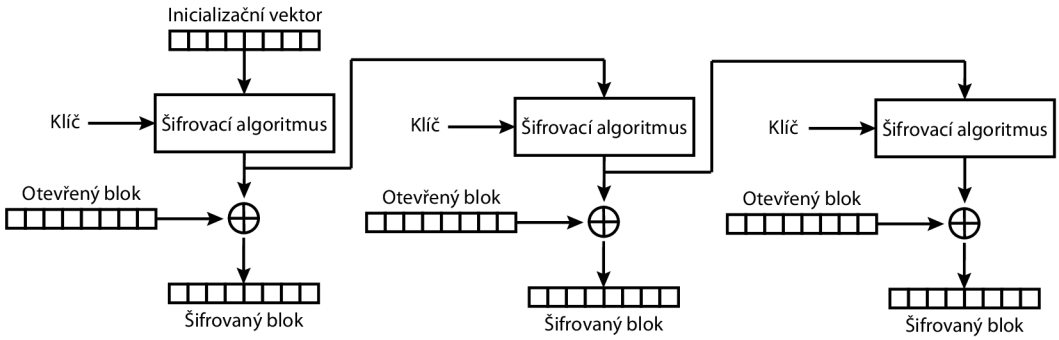
Režim CFB nebo také režim zpětné vazby se od předchozích dvou režimů liší. Otevřený text P se nerozděluje na bloky o velikosti odpovídající blokům šifrovacího algoritmu. Místo toho režim CFB chápe zprávu P jako proud symbolů o předem definované velikosti (např. 1 byte). Tento režim lze proto považovat za režim proudový. Šifrovací algoritmus představuje generátor klíčů o délce symbolu, který je navíc ovlivněn zpětnou vazbou. Šifrování otevřeného symbolu P_i probíhá pomocí operace XOR s klíčem vygenerovaným šifrovacím algoritmem. Na diagramu 2.6 je zobrazen zjednodušený princip režimu CFB.



Obrázek 2.6: Režim Cyphertext FeedBack.

Režim OFB (Output FeedBack)

Mód výstupní zpětné vazby (OFB) je podobný režimu CFB. Otevřený text P chápe jako proud symbolů, které mají předem definovanou velikost. Šifrovací algoritmus opět zastupuje úlohu generátoru klíčů o velikosti symbolu, tentokrát však zpětná vazba nezahrnuje výsledek šifrování C_i, ale je závislá pouze na výstupu samotného generátoru [35]. Šifrování symbolů probíhá stejně jako u režimu CFB pomocí operace XOR, která má za argumenty otevřený symbol P_i a vygenerovaný klíč. Diagram 2.7 zobrazuje zjednodušený princip režimu OFB.



Obrázek 2.7: Režim Output FeedBack.

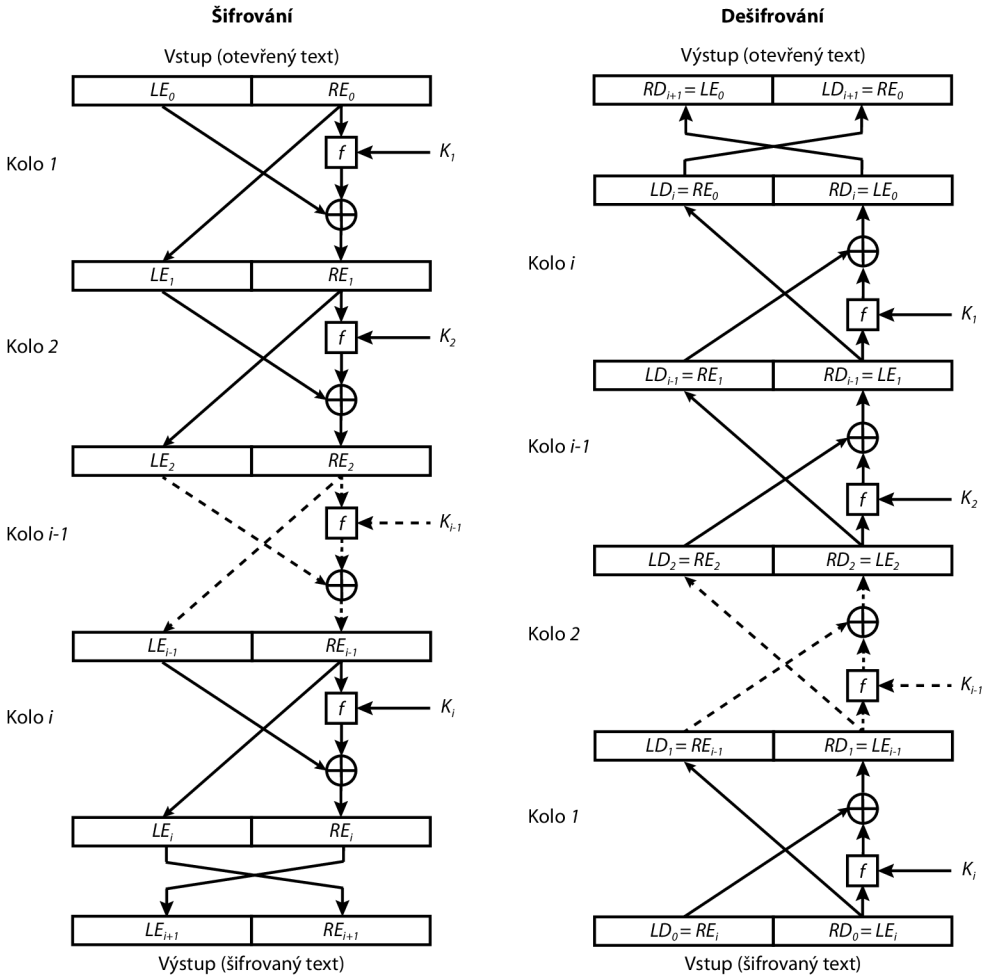
2.2 Vybrané algoritmy

V této části budou probrány detaily a principy vybraných moderních symetrických šifrovacích algoritmů.

2.2.1 Feistelova šifra

Algoritmus Feistelovy šifry byl navržen na základě předpokladu, že můžeme aproximovat ideální blokovou šifru za použití konceptu produkční šifry [28]. Tento koncept kombinuje aplikaci dvou nebo více šifer sekvenčně za sebou, jehož cílem je dosažení kryptograficky silnějšího výsledku, něž kdyby byly jednotlivé šifry aplikovány samostatně. Podstatou tohoto přístupu je vytvořit blokový šifrovací algoritmus, který pracuje s tajným klíčem o délce k bitů, což umožnuje celkově 2^k možných transformací. Princip algoritmu je navržen tak, že postupně střídá substituce a permutace [10].

Struktura mnoha symetrických šifrovacích algoritmů, které pracují v blokovém režimu, je založena na Feistelově šifře [39]. Její princip je popsán diagramem 2.8.



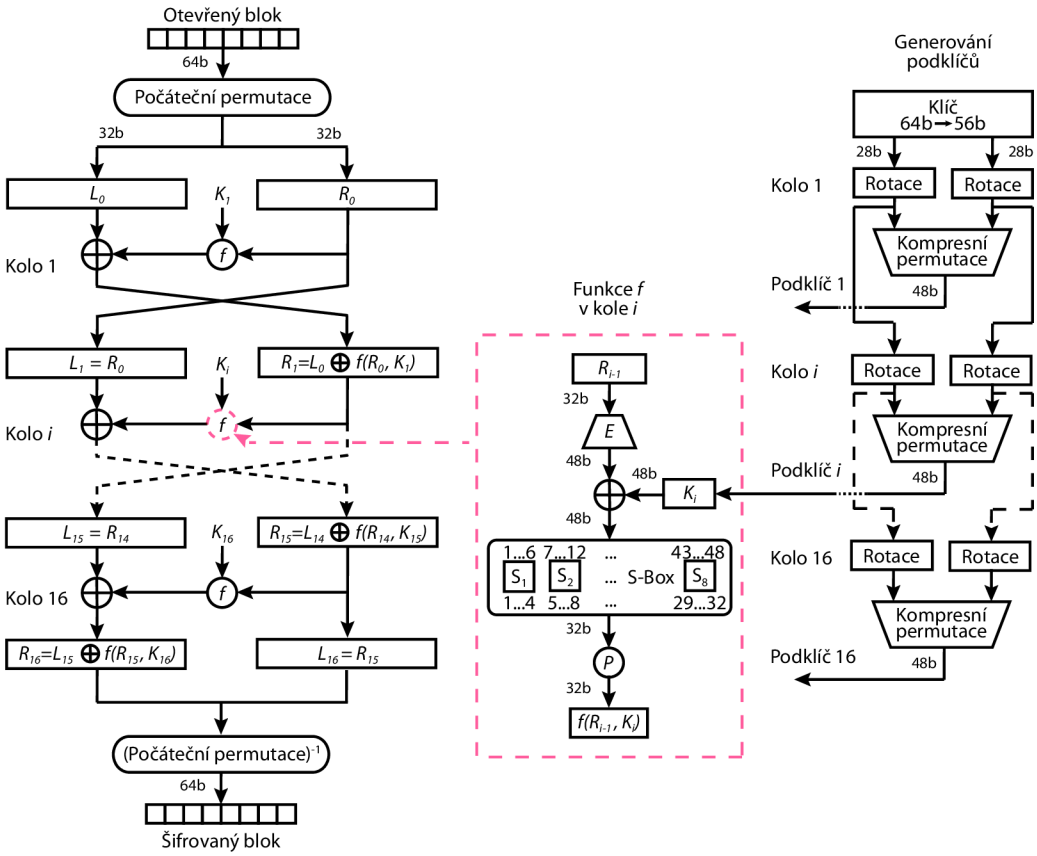
Obrázek 2.8: Princip šifrování a dešifrování pomocí Feistelovy šifry.

Vstupem šifrovacího algoritmu je blok otevřeného textu o délce $2w$ bitů a klíč K. Blok otevřeného textu je rozdělen na dvě poloviny, L_0 a R_0. Obě poloviny procházejí postupně skrz i kol, jejichž výsledkem je šifrovaný blok. Každé kolo i má vstup L_{i-1}, R_{i-1}, které jsou výsledky kola předchozího a dále podklíč K_i, který je odvozen z původního klíče K. Obecně jsou podklíče K_i odlišné od původního klíče K. Algoritmus může být implementován s různým počtem kol.

Každé kolo algoritmu má stejnou strukturu. *Substituce* je provedena s levou polovinou bloku a to tak, že na pravou polovinu bloku je aplikována funkce f a na její výstup je společně s levou polovinou aplikována operace XOR. Funkce f je parametrizovatelná pomocí podklíče K_i. Jinými slovy f je funkce pravého bloku dat o délce w bitů a podklíče K_i o délce y bitů. Tato funkce produkuje výstupní hodnoty o délce w bitů: $f(RE_i, K_{i+1})$. *Permutace* je prováděna pomocí výměny obou polovin dat. Tato struktura je formou tzv. *substitučně-permutační sítě* (SPN), někdy též označovaná jako *Feistelova síť* [39].

2.2.2 Data Encryption Standard

Za vývojem Algoritmus Date Encryption Standard (DES) [31] stojí společnost IBM. Tento šifrovací algoritmus je vylepšenou verzí algoritmu Lucifer [10]. Byl vyvinut v roce 1974 a v roce 1977 byl v USA zvolen jako standard pro šifrování dat ve státní a následně i v civilní sféře. DES je typem Feistelovy šifry a klasickým zástupcem blokových symetrických šifrovacích algoritmů. Jeho mechanismu je popsán diagramem 2.9.



Obrázek 2.9: Princip algoritmu DES.

DES je stejně jako všechny moderní šifry iterační algoritmus. Na každý blok otevřeného textu je aplikováno šifrování, které probíhá v 16 kolech s identickými operacemi. Velikost klíče tohoto algoritmu je 64 bitů, avšak 8 bitů je použito jako paritních. Efektivní délka klíče je tedy 56 bitů. Blok má délku 64 bitů.

Počáteční a koncová permutace

Počáteční permutace IP a koncová permutace IP^{-1} jsou bitové permutace, které si můžeme představit jednoduše jako proházení bitů vstupního otevřeného bloku podle definovaných pravidel (určeno pomocí dvou tabulek). Počáteční ani koncová permutace nezvyšují bezpečnost algoritmu DES a neexistuje rozumné vysvětlení, proč jsou jeho součástí [32]. Mezi počáteční a koncovou permutací platí, že permutace $X = IP(M)$, kde M je 64 bitový vstup a inverzní permutace $Y = IP^{-1}(X) = IP^{-1}(IP(M))$ [39].

Funkce f

Princip funkce f je zobrazen uprostřed diagramu 2.9. Funkce má dva vstupy, prvním je 32 bitová pravá polovinu šifrovaného bloku R_{i-1}, druhým je 48 bitový podklíč K_i viz 2.2.2 v kole i. Vstup R_{i-1} je na začátku rozšířen pomocí expanzní funkce (provádí duplikaci 16 bitů). Výsledných 48 bitů je XORován společně s podklíčem K_i. Výsledek operace XOR o délce 48 bitů je předán substituční funkci, která produkuje 32 bitový výstup. Ten je na závěr permutován. Substituční funkce je zastoupena pomocí tzv. S-boxů, které jsou jádrem a zároveň kryptografickou silou algoritmu DES. S-boxy jsou jediný nelineární prvek algoritmu. Motivace a rozhodnutí proč využívat S-box tabulky nebyla nikdy kompletně objasněna. Algoritmus využívá celkově 8 S-boxů, kdy každý substituuje příslušných 6 vstupních bitů na 4 bity výstupní. Každý S-box je definován tabulkou, kde vstupních 6 bitů určuje pozice do této tabulky (2 bity pro řádky a 4 bity pro sloupce). Na příslušné pozici je pak výstupní 4 bitová hodnota. [4, 32].

Detaily jednoho kola šifrování

Zaměřme se na levou část diagramu 2.9, která reprezentuje jednotlivá kola algoritmu DES. Každé kolo i se skládá ze samostatného zpracování levé L_{i-1} a pravé R_{i-1} poloviny šifrovaného bloku (každá o délce 32 bitů). Jako v každé jiné klasické Feistelově šifře můžeme proces zpracování kola i popsat těmito rovnicemi [32, 35]:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i) \\ f(R_{i-1}, K_i) &= P(S(E(R_{i-1})) \oplus K_i) \end{aligned}$$

Generování klíče

Princip generování podklíčů je znázorněn v pravé části diagramu 2.9. Vstupem tohoto mechanismu je původní 64 bitový klíč K. Klíč je na začátku permutován a výsledkem je 56 bitový klíč, který je rozdělen na dvě 28 bitové poloviny. V každém kole je na obě poloviny samostatně aplikována bitová rotace o 1 nebo 2 bity. Tyto rotované hodnoty slouží jako vstup pro další kolo a zároveň jako vstup pro kompresní permutaci, která produkuje podklíče o délce 48 bitů [28]. Tyto podklíče jsou následně parametrem pro funkci f.

Bezpečnost algoritmu DES

V době uvedení algoritmu DES jako standardu se vyskytly některé obavy a pochybnosti ohledně jeho bezpečnosti a to zejména od pánu Hellmana a Diffieho [15, 7]. Dále pánové Morris, Sloane a Wyner zveřejnily dvě možné slabiny algoritmu DES [30]:

- **Délka klíče** - 56 bitů dlouhý klíč nemusí poskytovat dostatečnou bezpečnost.
- **S-boxy** - mohou mít skryté zadní vrátka.

Diffie a Hellman následně potvrdili, že DES s 56 bitovým klíčem může být prolomen pomocí útoku hrubou silou (za použití modelu Known plaintext attack). Představili jednoúčelový stroj, který byl složen z miliónů LSI čipů, který byl schopen vyzkoušet všech 2^{56} klíčů během jednoho dne [7].

Triple DES

Triple DES (3DES) je variantou šifrovacího algoritmu DES, která byla vytvořena z důvodu nedostatečné délky klíče původního algoritmu. Princip této modifikace je jednoduchý, algoritmus DES je pro zašifrování jednoho otevřeného bloku puštěn 3x a pokaždé s jiným klíčem. 3DES tedy pracuje s blokem stejné délky jako DES, tedy 64b, délka klíče je 3x56 bitů, tedy 168 bitů. Počet kol algoritmu je 3x16, tedy celkově 48 kol. Princip algoritmu 3DES můžeme popsat jednoduchou rovnicí:

$$3DES(K_0, K_1, K_2) = DES(K_2, DES(K_1, DES(K_0, B)))$$

kde K_0, K_1, K_2 jsou klíče o délce 56 bitů a B představuje otevřený blok dat určený k zašifrování.

2.2.3 IDEA

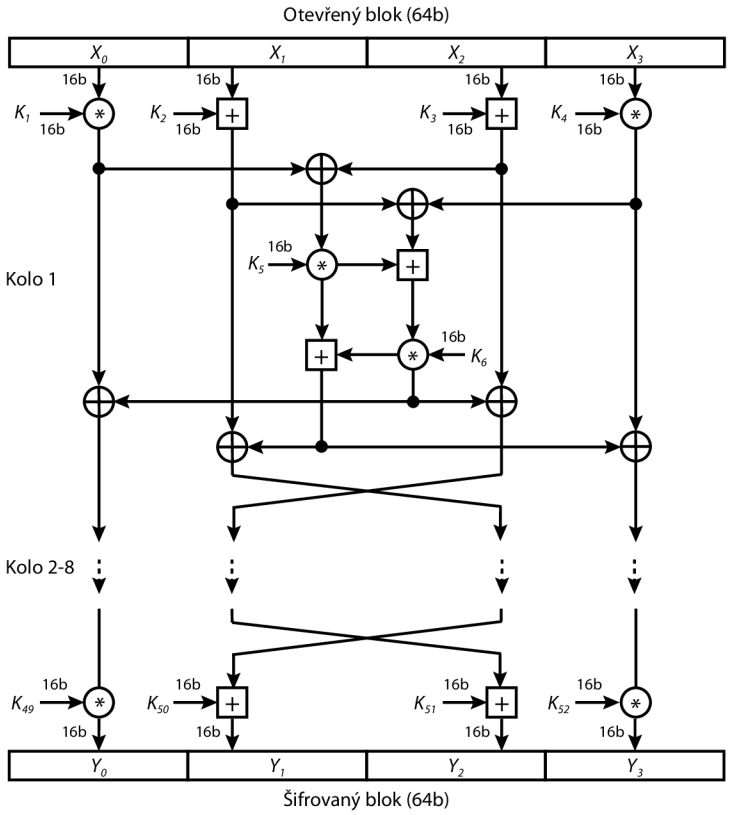
International Data Encryption Algorithm, dále jen IDEA, je šifrovací algoritmus, který byl představen pány Laiem a Masseyem v roce 1990 [24]. Algoritmus pracuje v blokovém režimu. Operuje s bloky otevřeného textu o délce 64 bitů a klíč má délku 128 bitů. IDEA je založena na několika teoretických základech a i když kryptoanalýza dosáhla jistého pokroku v oblasti redukce počtu iteračních kol, algoritmus IDEA nebyl stále prolomen. Je to jeden ze šifrovacích algoritmů, jehož kvalita byla prověřena časem. O jeho kvalitách vypovídá i to, že je součástí PGP [37].

Generování klíčů

IDEA pracuje v každém kole se šesti podklíči, které jsou získané rozdělením původního 128 bitového tajného klíče K na osm samostatných podklíčů o délce 16 bitů. Prvních šest podklíčů je použito pro první kolo a zbylé dva pro kolo druhé. V každém kole je původní tajný klíč K rotován o 25 bitů doleva a opět rozdělen na 8 samostatných podklíčů. První čtyři jsou použity ve druhé kole a zbylé čtyři v kole následujícím. Tento proces běží až do konce algoritmu.

Šifrování otevřeného textu

Algoritmus IDEA pracuje v 8 kolech. Vstupní 64 bitový otevřený blok je rozdělen na čtyři části X_0, X_1, X_2 a X_3, každý o délce 16 bitů. V každém kole jsou na jednotlivé části X_i společně s příslušnými podklíči K_i aplikovány matematické operátory $+$, $*$ a XOR. Mezi každým kolem jsou prohozeny bloky X_1 a X_2. Na závěr jsou jednotlivé bloky kombinovány s posledními čtyřmi podklíči a převedeny na výstup představující zašifrovaný blok [28]. Detailní popis principu algoritmu IDEA je znázorněn na diagramu 2.10.



Obrázek 2.10: Princip algoritmu IDEA.

2.2.4 RC4

RC4 je proudová šifra s variabilní délkou klíče. Byla vyvinuta v roce 1987 Ronem Rivestem pro společnost RSA Data Security, Inc. Detaily algoritmu byly po dlouhou dobu utajované a to až do roku 1994, kdy anonym rozeslal zdrojové kódy prostřednictvím emailu. RC4 je součástí spousty kryptografických produktů jako Lotus Notes či Oracle Secure SQL.

Algoritmus pracuje v módu OFB 2.7, kdy je proud klíčů nezávislý na otevřeném textu P. Obsahuje $8 * 8$ S-boxů: S_0, S_1, …, S_{255}, které u tohoto algoritmu představují generátor klíčů, tzv. *keysteam*. Úlohu jejich inicializačního vektoru zastupuje tajný klíč K s délkou 128 nebo 40 bitů [37].

Inicializace S-boxů

Jak již bylo zmíněno v úvodu, inicializace S-boxů probíhá pomocí tajného klíče K. Na začátku jsou hodnoty S-boxů inicializovány lineárně: $S_0 = 0$, $S_1 = 1$, …, $S_{255} = 255$. Následně je pole o délce 256 bytů naplněno tajným klíčem K a to opakovaně, dokud není pole plné: K_0, K_1, …, K_{255}. Dále jsou použity dva čítače i a j, které jsou inicializované na hodnotu 0 a následuje samotný proces, jehož pseudokód vypadá takto:

for $i = 0$ *to* 255 **do**
  $j = (j + S_i + K_i) \bmod 256$;
  swap(S_i, S_j);
end

Algoritmus 1: Inicializace S-boxů.

Generování klíčů a šifrování otevřeného textu

Samotné šifrování otevřeného textu probíhá po bytech, avšak je možné délku šifrovaného textu upravit např. na 2 byty. Mechanismus pracuje opět se dvěma čítači i a j, které jsou na začátku šifrování inicializovány na hodnotu 0. Každý vygenerovaný klíč o velikosti 1 byte je XORován s bytem otevřeného textu a tím je produkován text šifrovaný. Proces generování klíčů je popsán níže pomocí pseudokódu:

$i = (i + 1) \bmod 256$;
$j = (j + S_i) \bmod 256$;
swap(S_i, S_j);
$t = (S_i + S_j) \bmod 256$;
$K = S_t$;

Algoritmus 2: Proces generování klíčů.

Proces šifrování je díky své jednoduchosti velice rychlý, přibližně 10x rychlejší, něž u algoritmu DES.

2.3 Kryptoanalýza symetrických algoritmů

Moderní kryptoanalýza je skupina matematických metod a technik, které se díky znalosti návrhu šifrovacích algoritmů snaží najít jejich slabiny a tím tyto algoritmy oslabit nebo prolomit. Všechny symetrické šifrovací algoritmy dodržují Kerckhoffův princip [19, 40].

Kerckhoffův princip

V 19. století zmínil Auguste Kerckhoff názor, že pro šifrovací metodu nemusí být vyžadováno její utajení a klidně může padnout do rukou nepřítele. Jinými slovy, šifrovací mechanismus nemusí být držen jako tajemství. Pouze klíč by měl být tajnou informací, která je sdílena mezi komunikujícími stranami.

Tento přístup zajišťuje nezaujatý pohled z venčí na správnost mechanismus algoritmu a umožňuje tak hledat možné bezpečnostní nedostatky. Protože jsou mechanismy šifrovacích algoritmů veřejně známy, jsou tím zároveň zajištěny základní znalosti pro kryptoanalýzu.

Samotné útoky se pak na základě znalostí o mechanismu algoritmu a získaných datech viz 2.3.1 snaží najít různé vztahy mezi otevřeným textem P a odpovídajícím šifrovaným textem C bez znalosti tajného klíče K.

2.3.1 Modely útoků

Modely útoků se v principu liší podle dostupné znalosti informací útočníka. Níže budou probrány čtyři základní modely útoků proti šifrovacím systémům [19, 35, 40]:

- **Ciphertext only attack** - jedná se o nejzákladnější typ útoku. Ve scénáři má útočník přístup pouze k šifrovaným textům C a jeho snahou je bez jakékoliv další znalosti získat otevřený text P.
- **Known plaintext attack** - útočník má k dispozici jeden nebo více párů otevřeného textu P a k nim odpovídající šifrované texty C, které byly zašifrovány pomocí stejného tajného klíče K. Cílem tohoto modelu dešifrovat další šifrované texty C, ke kterým nemá otevřené texty P.
- **Chosen plaintext attack** - útočníkovi je dočasně poskytnut přístup k šifrovacímu systému. Následně si může zvolit otevřený text P a k němu zkonstruovat odpovídající zašifrovaný text C. Cílem je stejně jako u Known plaintext attack dešifrovat další šifrované zprávy C bez znalosti jejich otevřené podoby.
- **Chosen ciphertext attack** - tento model útoku je velmi podobný typu Chosen plaintext attack s rozdílem, že útočník v tomto případě volí libovolné šifrované texty C, ke kterým šifrovací systém konstruuje otevřené texty P.

2.3.2 Útok hrubou silou

Útok hrubou silou (brute force attack) není kryptoanalytickou metodou v pravém slova smyslu, protože kryptoanalytické metody hledají možnosti, jak prolomit šifrovací algoritmus rychleji, než právě za použití útoku hrubou silou. Pro úplnost je však vhodné tuto metodu zmínit, protože jak diferenciální kryptoanalýza 2.3.3 tak lineární kryptoanalýza 2.3.4 využívají tento typ útoku pro dopočet zbylých bitů klíče. Samotný útok je velice jednoduchý, postupně zkouší všechny možné varianty klíčů, dokud nenalezne shodu. V praxi je doba trvání útoku závislá na počtu kombinací klíčů nutných k ověření a na metodě, kterou se ověřuje shoda nalezeného klíče. Ideálním modelem pro tento typ útoku je Known plaintext attack [5].

2.3.3 Diferenciální kryptoanalýza

V této části se zaměříme na kryptoanalýzu z hlediska textových diferencí u iterativních blokových šifer založených na Feistelově síti. Takovou šifrou je například algoritmus DES. Diferenciální kryptoanalýza byla představena kryptology Bihamem a Shamirem v roce 1991 [4]. Tento typ kryptoanalýzy předpokládá model útoku Chosen plaintext attack. Tedy útočník má možnost výběru otevřeného textu, který následně zašifruje.

Diferenciální kryptoanalýza se zaměřuje na analýzu párů šifrovaných textů, jejichž otevřené texty mají částečné diference. Předpokládáme, že při procesu šifrování je používán totožný tajný klíč. Metoda postupně analyzuje vývoj diferencí tak, jak otevřené texty prochází skrz jednotlivá kola šifry DES. Pro algoritmus DES je pojem *diference* definovaná

pomocí operace XOR. Pro jiné algoritmy však může být diference definovaná odlišně. Analyzovaným diferencím výsledných šifrovaných textů jsou v závislosti k různým klíčům přiřazeny různé pravděpodobnosti. S analýzou dalších a dalších šifrovaných párů se na závěr jeví jeden klíč jako nejpravděpodobnější a tím je náš hledaný klíč [37, 39].

Princip útoku

Zaměříme se na algoritmus DES a na úvod upravíme notace. Otevřený blok m je složen ze dvou polovin m_0 a m_1. Každé kolo algoritmu DES mapuje pravou stranu vstupu na levou stranu výstupu a pravou stranu výstupu jako funkci s příslušným podklíčem jako vstup pro levou stranu. Každé kolo je tedy vytvořen pouze 32 bitový nový blok. Pokud označíme každý nový blok jako $m_i (2 \leq i \leq 17)$, pak jsou průběžné bloky polovin ve vztahu:

$$m_{i+1} = m_{i-1} \oplus f(m_i, K_i) \tag{2.1}$$

kde $i = 1, 2, ..., 16$. V diferenciální analýze začínáme se dvěma otevřenými bloky m a m' se známou XOR diferencí $\Delta m = m \oplus m'$ a předpokladem vztahu diferencí mezi průběžnými bloky polovin $\Delta m_i = m_i \oplus m'_i$. Úpravou získáme tento vztah:

$$\begin{aligned} \Delta m_{i+1} &= m_{i+1} \oplus m'_{i+1} \\ &= [m_{i-1} \oplus f(m_i, Ki)] \oplus [m'_{i-1} \oplus f(m'_i, Ki)] \\ &= \Delta m_{i-1} \oplus [f(m_i, Ki) \oplus f(m'_i, Ki)] \end{aligned} \tag{2.2}$$

Nyní předpokládejme, že mnoho párů vstoupí do funkce f se stejnými diferencemi a stejné diference postoupí výstupům za předpokladu, že pro šifrování byl použit stejný podklíč. Pokud s vysokou pravděpodobností známe Δm_{i-1} a Δm_i, pak s vysokou pravděpodobností známe i Δm_{i+1}. Je-li stanoven počet těchto diferencí, je možné určit podklíč funkce f. Celková strategie diferenciální kryptoanalýzy je stanovena na základě zmíněných úvah pro jedno kolo [39].

2.3.4 Lineární kryptoanalýza

Lineární kryptoanalýza byla zveřejněna kryptologem Matsuiem v roce 1992 [27]. Metoda může být principiálně aplikována na jakýkoliv iterační šifrovací algoritmus a jako příklad uveďme, že dokáže najít tajný klíč algoritmu DES za použití 2^{43} otevřených textů s jejich odpovídajícími zašifrovanými texty. V porovnání s diferenciální analýzou algoritmu DES, která vyžaduje k porovnání 2^{47} je tedy efektivnější. Další výhodou oproti diferenciální analýze je použití typu útoku known plaintext attack, protože je jednodušší získat pár otevřeného a šifrovaného textu, než mít možnost si otevřený text zvolit a na poskytnutém systému jej zašifrovat.

Princip útoku

Lineární kryptoanalýza vyžaduje nalezení množiny lineárních aproximací S-boxů, která může být použita pro získání lineární aproximace celé substitučně-permutační sítě. [39, 40].

Pro blok otevřeného a šifrovaného textu o délce n bitů a tajný klíč o délce m bitů označme blok otevřeného textu $P[1]$, $P[2]$, ..., $P[n]$, blok šifrovaného textu $C[1]$, $C[2]$, ..., $C[n]$ a tajný klíč $K[1]$, $K[2]$, ..., $K[m]$. Dále definujme

$$A[i, j, ..., k] = A[i] \oplus A[j] \oplus ... \oplus A[k] \tag{2.3}$$

Cílem lineární kryptoanalýzy je nalézt efektivní lineární rovnici ve formě:

$$P[\alpha_1, \alpha_2, ..., \alpha_a] \oplus C[\beta_1, \beta_2, ..., \beta_b] \quad = \quad K[\gamma_1, \gamma_2, ..., \gamma_c] \tag{2.4}$$

kde $1 \leq a$, $b \leq n$, $c \leq m$. Dále α, β a γ jsou termy reprezentující unikátní bitové pozice a celá rovnice má pravděpodobnost $p \neq 0.5$. Čím více je pravděpodobnost p odlišná od hodnoty 0.5, tím je rovnice efektivnější.

Jakmile je navrhovaný vzor určený, úkolem je počítat levou stranu rovnice pro velké množství párů otevřeného a šifrovaného textu. Pokud je ve více než polovině případů výsledek 0, předpokládáme $K[\gamma_1, \gamma_2, ..., \gamma_c] = 0$. Pokud je ve více jak polovině případů výsledek 1, předpokládáme $K[\gamma_1, \gamma_2, ..., \gamma_c] = 1$. Tímto získáme lineární rovnici pro bity klíče. Snahou je získat více takových vztahů, které můžeme řešit pro získání bitů klíče. Protože pracujeme s lineárními rovnicemi, k problému můžeme přistupovat po jednotlivých kolech algoritmu a výsledky kombinovat [39].

Kapitola 3

Genetické programování

V roce 1959 položil Arthur Samuel otázku „Jak se mohou počítače učit řešit problémy, aniž by byly explicitně programovány? Jinými slovy, jak mohou počítače dělat to, co potřebujeme, aniž bychom jim řekli jak to udělat?“.

Na tuto otázku odpovídá oblast počítačové vědy, kterou dnes nazýváme evoluční algoritmy. Práce na této oblasti započaly v 60. letech 20. století ve Spojených státech amerických a Německu. Využití evolučních mechanismů můžeme rozdělit na dvě samostatné části: Evoluční strategie (ES) a Genetické algoritmy (GA). Ideu genetických algoritmů jako adaptivní systém umělé inteligence (účinný prohledávací mechanismus) poprvé prezentoval J. Holland. Definoval operátor křížení a operátor inverze [16].

Evoluční algoritmy hledají nejlepší možné řešení prostřednictvím populace jedinců v generacích (cyklech), podle daných pravidel [23]. Jsou inspirovány dílem „O vzniku druhů“, nejvýznamnější prací Charlese Darwina. Evoluce z hlediska evolučních algoritmů je postupný proces, kdy dochází k adaptaci jedinců pomocí fitness funkce. Stejně jako v přírodě se v průběhu generací jednotlivé druhy života zdokonalují, analogicky se v průběhu evolučního algoritmu zdokonalují generace řešení problému. Jako v přírodě totiž platí, že zdatnější jedinci (řešení problému) mají větší šanci přenést své vlastnosti do dalších generací [6]. Nové generace jedinců v sobě akumulují právě takové genetické změny, které jim umožňují lepší adaptaci, možnost přežít a možnost reprodukovat se. Větší fitness hodnota jedince znamená lepší adaptaci a tím i větší pravděpodobnost, že bude vybrán pro reprodukci do další generace a tím promítne svůj genotyp do dalších potomků [38].

Hlavní rozdíl mezi evolučními strategiemi a genetickými algoritmy je v použití operátorů (mutace, reprodukce a křížení). Mutace je hlavní operátor pro evoluční strategie. Za to genetické algoritmy považují za primární rekombinační operátor křížení.

Genetické programování (GP) vzniklo koncem 80. let 20. století jako rozšíření genetických algoritmů, jsou to biologicky inspirované metody, které jsou schopny vytvořit počítačové programy, které řeší vysokoúrovňové problémy [1]. Reálně je problém umělé inteligence, strojového učení, adaptivních systémů a automatického učení převeden na hledání počítačového programu; genetické programování nám poskytuje cestu, jak najít počítačový program řešící daný problém v prostoru počítačových programů [20].

Podstatným rozdílem mezi evolucí v živé přírodě a genetickým programováním je, že v živé přírodě probíhá evoluce trvale podle změn prostředí a bez cíle (nemá žádný koncový stav), zatímco cíl evoluce genetického programování je definován fitness funkcí [33, 38].

Genetické programování imituje aspekty přirozené evoluce, avšak kde genetické algoritmy hledají pole znaků nebo čísel, účel procesu genetického programování je najít počítačový program, který řeší zadaný problém. Na rozdíl od genetických algoritmů nemá tak

genetické programování při hledání vhodného řešení daného problému omezenou strukturu (pevnou délkou chromozomu) [8]. Jedinci/programy genetického programování tak mohou nabývat potenciálně neomezené komplexity [41].

Tato kapitola je rozdělena na 6 částí. V první části 3.1 budou zmíněny biologické pojmy týkající se genetického programování. V 3.2 jsou detailně probrány základy genetického programování od reprezentace jedince, přes fitness funkci až po určení výsledků a ukončení evolučního procesu. V části 3.3 budou podrobně rozebrány principy genetických operátorů, které transformují aktuální generaci jedinců na generaci novou. Dále bude prezentován algoritmus genetického programování 3.4 a popis symbolické regrese 3.5 jako metody využití genetického programování v praxi. Na závěr budou v části 3.6 zmíněny problémy genetického programování.

3.1 Biologické pojmy

V krátkosti se zmíníme o některých biologických pojmech, které se často vyskytují v oblasti genetického programování [8].

Jedinec

Někdy označován jako chromozóm, je program, který je potenciálním kandidátem na řešení příslušného problému. Představuje prvek prohledávaného prostoru všech řešení daného problému a je vytvářen z existujících jedinců pomocí genetických operátorů.

Populace

Je reprezentována množinou jedinců (chromozómů). Počáteční populace je obvykle definována daným počtem jedinců, kteří jsou následně generování náhodně.

Generace

Představuje aktuální populaci jedinců.

Genotyp

Je genetická výbava jedince zakódovaná do vhodné datové struktury. V případě genetických algoritmů je to pole hodnot, v případě genetického programování je to graf typu strom, který reprezentuje nějaký program.

Fenotyp

Je interpretací genotypu. V přírodě určuje fenotyp konkrétní projevy genu (např. barva očí), u genetických algoritmů představuje fenotyp dekódování genotypu (např. dekódování pole dekadických čísel). V genetickém programování představuje fenotyp interpretaci genotypu (výsledek provedení programu).

3.2 Parametry genetického programování

Pod parametry GP jsou v této části myšleny všechny vlastnosti genetického programování, které je nutné definovat za účelem jeho spuštění. Bude probrán jazyk reprezentace jedince, ohodnocující funkce určující kvalitu jedince, která je hnacím motorem evoluce. Dále bude probráno generování výchozí populace, která je nezbytná pro běh GP. Na závěr budou zmíněny podmínky nutné pro ukončení evolučního procesu.

3.2.1 Jazyk reprezentace

Kód každého jedince je v GP tvořen pomocí množiny funkcí a množiny terminálů [3, 17]. Tyto dvě množiny budou detailně probrány níže. Následovat bude popis celkové struktury chromozomu a také budou zmíněny automaticky definované funkce.

Množina terminálů

Vstupy do programů, které jsou evolvovány pomocí GP, jsou reprezentovány pomocí množiny terminálů. Tato množina obsahuje konstanty, 0-aritní funkce (funkce bez argumentů) a proměnné, které zastupují vstupy do programu. Pojem terminál je použit z důvodu, že všechny symboly z množiny tvoří ve stromové struktuře jedince listy stromu (jsou to ukončující symboly). Terminály při interpretaci jedince předávají programu svoji hodnotu.

Konstanty jsou v klasické variantě GP zvoleny pro celou populaci jako množina hodnot (např. určitá podmnožina celých čísel).

Funkce bez argumentů zastupují takové funkce, které nejsou závislé na žádném vstupu a při jejich interpretaci předávají programu nějakou hodnotu. Takovou funkcí může být např. funkce `rand()`.

Proměnné umožňují předávání dat z trénovací množiny do programu a hrají tak důležitou roli v procesu učení.

Množina terminálů	
Typy primitiv	Příklady
Proměnné	x, y
Konstanty	3, 0.45
0-aritní funkce	`rand()`, `go_left()`

Množina funkcí

Množina použitých funkcí je závislá na oblasti problému, který má GP řešit. Součástí množiny mohou být aritmetické a logické operace, dále klasické konstrukce běžné z programovacích jazyků jako podmínky, cykly a podprogramy. Není vhodné použít příliš rozsáhlou množinu funkcí, protože se tím zvětšuje prohledávací prostor a požadované řešení pak nemusí být nalezeno [38]. Množina funkcí by tedy měla být volena tak, aby s její pomocí bylo možné co nejlépe reprezentovat řešení zadaného problému.

Aby mohlo GP pracovat efektivně, u většiny množin funkcí je vyžadováno, aby splňovaly důležitou vlastnost známou jako *uzavřenost* [20]. Tato vlastnost znamená, že všechny funkce v množině musí akceptovat jakýkoliv vstup z množiny terminálů a dále všechny hodnoty, které vznikly jejich vyhodnocením množinou funkcí. Tím je zamezen výskyt sémantických chyb v kódu. Jako příklad můžeme uvést chráněné (protected) varianty některých funkcí jako např. dělení, které není definováno pro hodnotu 0. V tomto případě je nutné vracet nějakou bezpečnou hodnotu např. velmi vysoké číslo.

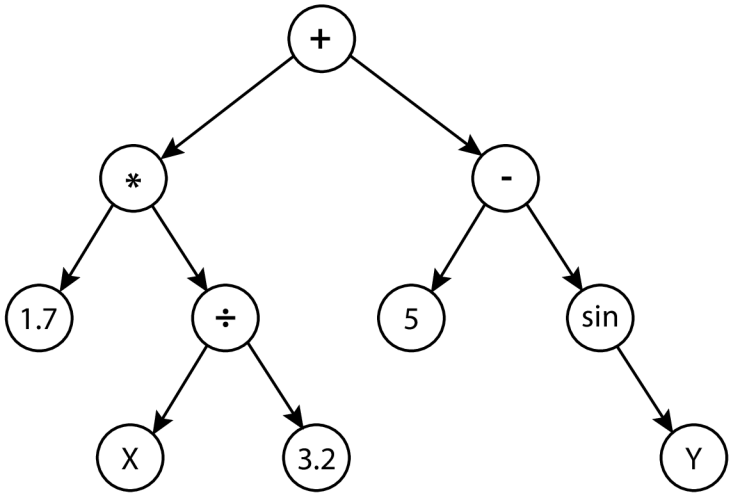
Množina funkcí	
Typy primitiv	Příklady
Aritmetické	+, -, *, /
Matematické	sin, cos, exp
Logické	AND, OR, NOT
Podmíněné	IF-THEN-ELSE
Cykly	FOR, REPEAT
...	...

Reprezentace chromozomu

Programy v GP pracují s tzv. spustitelnými strukturami, které se sestávají z funkcí a terminálů. Základními strukturami jsou lineární, stromové a grafové struktury a typicky mají proměnnou délku. Nejčastěji používané struktury jsou syntaktické stromy, protože jsou vhodné pro strojové zpracování a lze je použít téměř v kterémkoliv programovacím jazyce [20]. Reprezentace stromovou strukturou je také vhodná při aplikaci genetických operátorů 3.3.

Průchodem stromové struktury dojde k interpretaci programu, který strom reprezentuje. V pokročilejších formách GP mohou být programy skládány pomocí různých komponent (podprogramů) viz ADF 3.2.1, podmíněných výrazů, cyklů či rekurze, čímž dostáváme programovací jazyk, který je schopen generovat programy, které řeší i komplexnější problémy.

Na následujícím obrázku 3.1 je jednoduchý příklad stromové struktury programu, kdy interpretací stromu dojde k vykonání programu $(1.7 * \frac{X}{3.2}) + (5 - sin(Y))$.



Obrázek 3.1: Reprezentace chromozomu stromovou strukturou.

Automaticky definované funkce

Ve většině reálných algoritmů lze identifikovat takové části kódu, které se v řešení opakují a mohou tak tvořit logický podcelek hledané funkce/programu. Tyto sekvence kroků by měly přirozeně snížit nároky na nalezení řešení v GP. V klasickém programování jsou tyto sekvence baleny do znovupoužitelných komponent jakými jsou například podprogramy, funkce a třídy. Mohou být opakovaně volány a to typicky s různými vstupy. Z definice množiny

funkcí a terminálů GP je zřejmé, že se tyto množiny nemohou během experimentu měnit a nemáme tak žádný nástroj, jak evolvovaný program rozdělit na hierarchicky vyšší celky.

Tento problém řeší právě Automaticky definované funkce (ADF), které jsou široce používanou metodou evolučních opakovaně použitelných komponent v GP [21]. Opakované použití komponent eliminuje nutnost znovu objevovat již jednou nalezené funkce/podprogramy. Také umožňuje nalezení a využití modularit problému, jeho symetrické závislosti a zákonitosti a tím potenciálně urychlit proces řešení problému. Ke zvýšení efektivity evoluce (snížení počtu kroků výpočtu) v porovnání s případem GP bez ADF je nutné, aby ADF obsahovaly části kódu, které jsou „užitečné“ pro řešení daného problému. Je tedy nezbytné, aby experimentátor kvalitně porozuměl problematice řešeného úkolu.

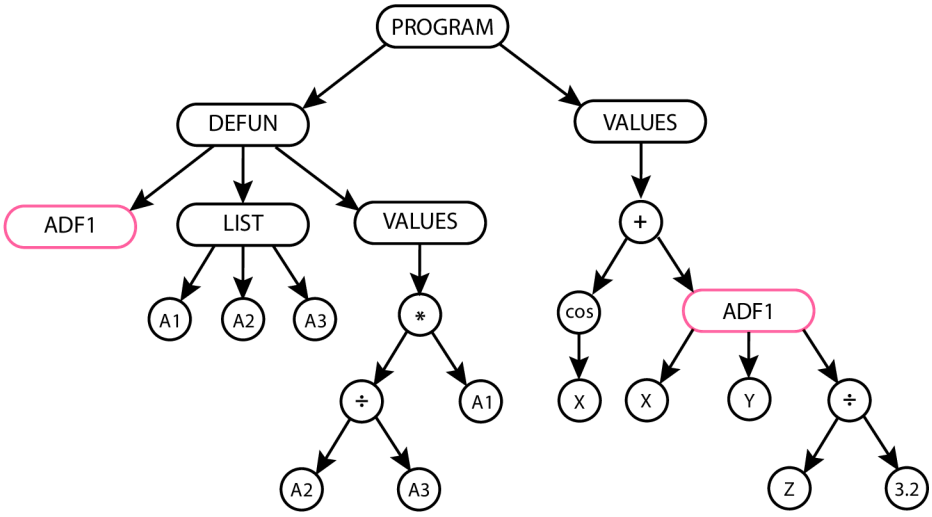
Principem ADF je rozšíření množiny funkcí o pomocné funkce a další parametry:

- počet ADF, které může jedinec volat
- počet argumentů pro každou ADF
- volba množiny terminálů a funkcí, ze kterých budou ADF tvořeny

Tyto pomocné funkce mohou jedinci napříč populací volat v rámci svého kódu a během evoluce se kód pomocných funkcí vyvíjí stejně jako kód každého jedince. Jedinec obsahující ADF je reprezentován syntaktickým stromem jako doposud s tím rozdílem, že je strom rozdělen na 2 části:

- hlavní větev, která je vyhodnocena během evoluce a může obsahovat volání ADF
- větve pomocných funkcí, která definuje jednu nebo více ADF

Na obrázku 3.2 je znázorněno použití ADF v praxi. Hlavní interpretovaná větev programu je reprezentovaná pravým podstromem od uzlu VALUES, ve kterém se nachází i volání pomocné funkce ADF1. V levém podstromu jedince je od uzlu DEFUN definována pomocná funkce ADF1. Její argumenty A1, A2 a A3 se nacházejí v podstromu LIST. Samotné tělo pomocné funkce ADF1 je konstruováno v podstromu VALUES.



Obrázek 3.2: Struktura stromu jedince s ADF.

Kromě ADF byly navrženy další typy automaticky definovaných komponent, kterými jsou Automaticky definované cykly (ADLs) a automaticky definované rekurze (ADRs), které poskytují prostředky k opakovanému použití kódu. Dále Automaticky definované uložiště (ADSs), které poskytuje možnosti jak znovu použít výsledky již spuštěného kódu [22].

3.2.2 Fitness funkce

Množina terminálů a funkcí (popřípadě i ADF) definuje primitiva genetického programování a tím nepřímo definuje i prostor možných programů určený k prohledávání. Tím jsou zahrnuty všechny programy, které mohou být zkonstruovány skládáním primitiv všemi dostupnými způsoby. V tento okamžik však nejsme schopni určit, které programy nebo části prohledávaného prostoru programů jsou dobré, resp. které řeší nebo aproximují řešení našeho problému. K tomuto účelu slouží právě fitness funkce, jejíž vyhodnocení vyjadřuje kvalitu jedince a je stěžejním mechanismem navigace v prohledávaném prostoru a konvergence k přijatelnému výsledku [33].

Protože struktury evolvované genetickým programování jsou počítačové programy, výpočet fitness funkce vyžaduje interpretaci všech programů v populaci a ze získaných výsledků je každému jedinci/programu přiřazena pomocí fitness funkce jeho ohodnocení kvality (fitness). Fitness funkce je předem určena programátorem a sama o sobě nepodléhá procesu evoluce. Měření kvality jedince lze provádět mnoha způsoby a konkrétní podoba fitness funkce záleží na zadání příslušného problému. Základní přístupy k měření kvality jedinců jsou sémantická analýza a shoda na trénovací množině. [8]. S následujícími termíny pro fitness hodnoty se můžeme setkat v literatuře [20]:

Hrubá fitness

Ohodnocení každého jedince se obvykle provádí na trénovací množině, kterou si můžeme představit jako množinu dvojic, kde každá dvojice reprezentuje vstupní hodnotu a hodnotu očekávanou na výstupu. Výsledkem interpretace každého jedince je hodnota (číslo popř. pravdivostní hodnota). Hrubá (raw) fitness definuje výsledek jako chybu/odchylku od očekávaného řešení a je určena následujícím vztahem:

$$r(i) \quad = \quad \sum_{j=1}^{N} |S\ (i,j) - C(j)| \tag{3.1}$$

kde $S(i,j)$ je výsledek vyhodnocení (interpretace) jedince i pro prvek j trénovací množiny, která má mohutnost o velikosti N prvků. $C(j)$ je očekávaná hodnota výstupu pro odpovídající vstupní prvek j trénovací množiny.

Standardizovaný fitness

Přepočtem čistého fitness na určitou referenční hodnotu získáme fitness standardizovaný. Příklad: mějme referenční (maximální, tedy ideální výsledek) fitness určený hodnotou 100, čistý fitness aktuálně počítaného jedince je 67, pak standardizovaný fitness $= 100 - 67 = 33$. Standardizovaný fitness je dán tímto vztahem:

$$s(i) \quad = \quad r_{max} - r(i) \tag{3.2}$$

kde $r(i)$ je čistý fitness a r_{max} je referenční hodnota fitness.

Upravený fitness

Vyjadřuje převedení standardizovaného fitness na interval $\langle 0, 1\rangle$. Pro převod je použit tento vzorec:

$$a(i) = \frac{1}{1 + s(i)} \tag{3.3}$$

Výsledek 1 upraveného (adjustment) fitness představuje ideálního jedince, naopak hodnota 0 jedince nejhoršího.

Normalizovaný fitness

Vychází ze vztahu pro upravený fitness. Vypočítá se podle tohoto vztahu:

$$n(i) = \frac{a(i)}{\sum_{k=1}^{M} a(k)} \tag{3.4}$$

a představuje vůči němu proporční přepočet, pro který platí tyto vlastnosti: $n(i) \in \langle 0, 1\rangle$, kvalitnější jedinec má vyšší n_i a $\sum_{i=1}^{M} n(i) = 1$.

3.2.3 Generování výchozí populace

Prvním krokem běhu algoritmu GP je inicializace počáteční populace. Jako v jiných genetických algoritmech je i v genetickém programování generována počáteční množina jedinců náhodně [33]. Proces inicializace populace náhodně vybírá symboly z množiny terminálů 3.2.1 a množiny funkcí 3.2.1 a podle zadaných pravidel z nich vytváří programy reprezentované stromovou strukturou. Protože je počáteční populace generovaná náhodně bez jakékoliv znalosti o řešeném problému, je průměrná fitness hodnota jedinců nízká.

V následující části budou probrány 3 různé přístupy ke generování výchozí populace, kterými jsou metoda Grow, Full a Ramped Half and Half.

Metoda Grow

V této metodě jsou jedinci generování náhodným výběrem z množiny terminálů a funkcí. Startující uzel, který přestavuje kořen stromu je generován pouze z množiny funkcí. Protože je následný výběr náhodný, nemusí generovaný strom dosáhnout uživatelem definované maximální hloubky. Jakmile je zvolen terminální symbol, je daná větev stromu ukončena. Výsledkem této metody jsou jedinci nepravidelných tvarů a velikostí.

Metoda Full

Metoda Full vybírá na rozdíl od metody Grow za uzly pouze symboly z množiny funkcí a to do okamžiku, kdy větev dosáhne maximální hloubky stromu. Jakmile je uživatelem definované hloubky dosaženo, je vybrán terminální symbol a daná větev stromu končí.

Hloubka uzlu je hodnota vyjadřující počet hran vedoucí od daného uzlu ke kořeni stromu (startující neterminál). Hloubka stromu je hodnota vyjadřující počet hran vedoucí k nejhlubšímu z jejich koncových uzlů (listů). V metodě Full jak už její název napovídá, jsou

generovány plné stromy, jejichž listy jsou ve stejné (maximální) hloubce. Neznamená to však, že by měli jedinci generovaní touto metodou stejnou velikost (stejný počet uzlů). Tato situace by mohl nastat jen v případě, pokud by množina funkcí obsahovala funkce o stejné aritě.

Metoda Ramped Half-and-Half

Protože ani jedna z metod Grow a Full negeneruje populaci dostatečně rozmanitou ve smyslu velikosti stromů, byla vytvořena metoda Ramped Half-and-Half, která kombinuje vlastnosti obou výše zmíněných metod [20]. Navíc zavádí princip *Ramped*, který zajišťuje vytvoření různých variací hloubek stromů. Pro příklad předpokládejme maximální definovanou hloubku 6. Potom je celá populace rozdělena na skupiny jedinců o hloubce 2, 3, 4, 5, a 6. Následně je na každou skupinu jedinců použit princip *Half-and-Half*, kdy je na jednu polovinu stromu každého jedince aplikována metoda Grow a na druhou polovinu metoda Full. Výsledkem této metody je pak mnohem rozmanitější výchozí populace.

3.2.4 Určení výsledku a podmínek ukončení evoluce

Evoluce v živé přírodě probíhá jako nikdy nekončící proces. Avšak k vyřešení reálného problému v praxi za použití síly evoluce to není chtěná vlastnost. Proto k zastavení evolučního procesu genetického programování používáme různá kritéria. Nejčastěji používaná kritéria jsou popsána dále [36].

Počet generací

Kritérium ukončí evoluční proces na základě dosažení určené generace. Ukončující generaci předem definuje uživatel. Pro zajímavost, pokud se nenalezne řešení v 50 generacích, už se zpravidla nenalezne nikdy [20].

Výpočetní čas

K zastavení evoluce dojde v okamžiku, kdy vyprší doba určená k výpočtu. Opět je předem definovaná uživatelem a algoritmus ukončí činnost v době, kdy dojde k dokončení výpočtu aktuálního cyklu (generace).

Dosažení požadované fitness hodnoty

Toto kritérium ukončí výpočet, pokud fitness hodnota jedince v aktuální generaci je nižší než hodnota definovaná uživatelem. Kritérium zastavení podle fitness hodnoty vyžaduje dobrou znalost řešené problematiky.

Konvergence populace

Konvergence populace je kritérium, které ověřuje, zda v populaci dochází ke změnám jedinců. Pojem *zkonvergovaná populace* znamená, že v populaci již nedochází k žádným výrazným změnám a to např. na základě porovnání oscilace fitness hodnot jedinců vzhledem k průměrné fitness hodnotě populace popř. k fitness hodnotě nejlepšího jedince (parametr oscilace může být definována např. jako procentuální rozdíl).

3.3 Genetické operátory

V genetickém programování jsou definovány tyto základní operátory: křížení, reprodukce a mutace [3, 18]. Z vybraných jedinců se aplikací genetických operátorů vytváří v průběhu evolučního procesu nová populace [18]. Počáteční populace je tedy transformována pomocí genetických operátorů na populaci konečnou.

3.3.1 Reprodukce

Je typem genetického operátoru, který podle stanoveného selekčního mechanismu vybírá vhodné jedince k reprodukci do nové generace [18]. Selekční mechanizmus napodobuje přirozený výběr podle teorie Charlese Darwina, kdy lépe adaptovaní jedinci mají větší pravděpodobnost se reprodukovat [6]. Tento mechanismus zajišťuje, že průměrná kvalita populace s počtem generací roste.

Selekce jedinců je významnou částí genetických algoritmů. Výběr jedinců pro reprodukci do další generace musí dostatečně upřednostňovat kvalitní jedince (s vyšší hodnotou fitness), ale zároveň musí novou generaci vytvořit dostatečně různorodou. Tím má selekční algoritmus velký vliv na výkon genetického programování. Jestliže nesplňuje jeden ze dvou zmíněných požadavků, vede to v případě upřednostnění kvalitních jedinců k tzv. předčasné konvergenci (do lokálního maxima řešení), v druhém případě k pomalé konvergenci algoritmu [38].

Zvýhodnění kvalitních a potlačení podprůměrných jedinců je vlastnost selekčního mechanismu, která se nazývá *selekční tlak*. Selekční tlak nebo také selekční intenzita je vyjádřena následujícím vztahem:

$$I = \frac{\overline{M^*} - \overline{M}}{\overline{\sigma}} \tag{3.5}$$

kde $\overline{M}$ značí průměrnou fitness hodnotu v generaci před selekcí, $\overline{M^*}$ průměrnou fitness hodnotu po selekci a $\overline{\sigma}$ značí rozptyl fitness hodnot před selekcí. Čím vyšší je selekční tlak, tím rychleji algoritmus konverguje – zároveň však vzrůstá nebezpečí předčasné konvergence.

Selekčních operátorů pro výběr vhodného jedince k následné reprodukci do nové generace existuje velké množství [12].

Proporcionální selekce (roulette wheel selection)

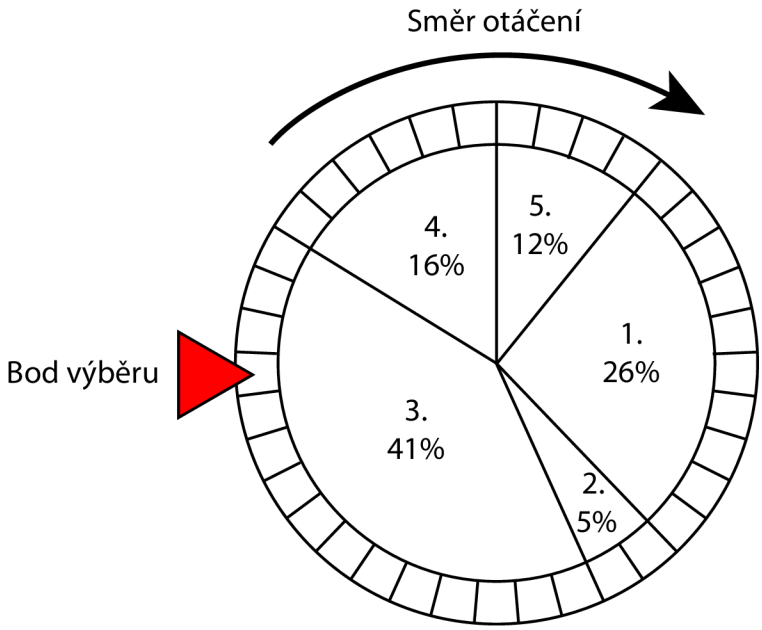
Je prvním algoritmem určeným k selekci jedinců a bývá označována jako tzv. ruletový mechanismus selekce [29]. Mechanismus proporcionální selekce vybírá jedince na základě pravděpodobnosti, která je přímo úměrná jejich fitness hodnotě. Můžeme proto tento mechanismus analogicky přirovnat ke klasické ruletě, kde jedince reprezentuje výseč rulety, jejíž obsah odpovídá poměru jeho ohodnocení vůči ohodnocení celé populace. Jedinci s vyšší hodnotou fitness tak zabírají větší výseč rulety (vyšší pravděpodobnost výběru) a naopak méně zdatní jedinci mají pravděpodobnost k výběru pro následující reprodukci menší, viz grafické zobrazení ruletového mechanismu 3.3.

Nevýhodou proporcionální selekce je problém předčasné konvergence. V aktuální generaci se může vyskytnout jedinec, jehož fitness hodnota je mnohem vyšší než průměrné ohodnocení ostatních jedinců. Tento silný jedinec svojí pravděpodobností výběru zabrání slabším jedincům v reprodukci. Tímto procesem se z populace postupně vytratí rozmanitost genetické výbavy a řešení bude předčasně konvergovat.

Pravděpodobnost výběru jedince i je dána následující pravděpodobností:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{3.6}$$

kde f_i je standardizovaný fitness jedince i v populaci o velikosti M. Platí tedy, že jedinec se reprodukuje s pravděpodobností, která odpovídá podílu jeho fitness k celkovému součtu fitness hodnot celé populace.



Obrázek 3.3: Selekce formou ruletového mechanismu.

Lineární uspořádání

Mechanismus lineárního uspořádání provádí úpravu selekčního tlaku, kterým řeší problém předčasné konvergence. Principem je výběr jedinců podle pořadí jejich ohodnocení [2], kdy se populace jedinců setřídí tak, že nejlepší jedinec je umístěn na index N a nejhorší na index 1. Touto úpravou se smaže velikost rozdílů jejich fitness hodnot a slabší jedinci tak získají větší šanci k výběru a následné reprodukci do nové generace. Díky tomu je také zachována dostatečná rozmanitost populace.

Pravděpodobnost výběru P_i jedince i v množině sestupně uspořádaných jedinců získáme tímto vztahem:

$$p_{lin-rank}(i) = \frac{2-s}{N} + \frac{2i(s\ -1)}{N(N-1)} \tag{3.7}$$

kde s je hodnota selekčního tlaku a $i \in \{1, 2, ..., N\}$.

Exponenciální uspořádání

Používá stejně jako mechanismus lineárního uspořádání seřazenou populaci jedinců, kde nejlepší jedinec je opět umístěn na index N a jedinec nejslabší na index 1. Liší se však

pravděpodobností výběru, která již není rozložena lineárně, ale s exponenciální závislostí. Z výše uvedených mechanismů selekce patří exponenciální uspořádání k nejlepší.

Pravděpodobnost výběru jedince na indexu i je dána tímto vztahem:

$$P_{exp-rank}(i) = \frac{1 - e^{-i}}{c} \tag{3.8}$$

kde volba vhodného selekčního tlaku závisí na parametru c, který může nabývá hodnot v rozsahu $0 < c < 1$, $i \in \{1, 2, ..., N\}$.

Turnajová selekce

Další metodou výběru jedince pro reprodukci je turnajová selekce. Typicky jsou náhodně vybráni 2 jedinci, kdy jedinec s vyšší fitness hodnotou je vybrán jako vítěz. Tato metoda simuluje biologický proces rozmnožování, kde dva jedinci se stejným pohlavím soupeří o dalšího jedince s odlišným pohlavím a silnější dostává možnost účastnit se reprodukčního procesu [20].

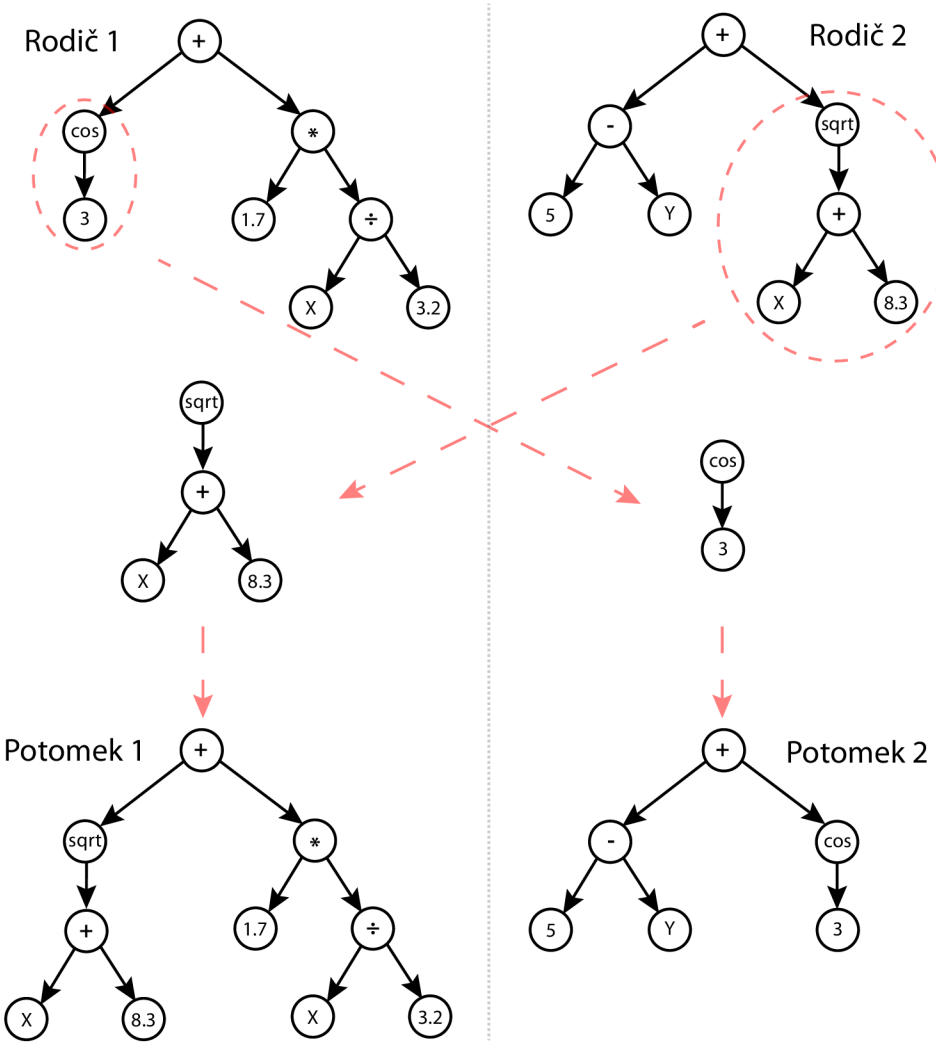
Turnajová selekce dosahuje podobných výsledků jako selekce exponenciální. Její výhodou je, že k aplikaci nevyžaduje setříděnou populaci jedinců, vlastní selekce je velmi jednoduchá a zároveň zachovává dostatečnou diverzitu populace. Díky těmto vlastnostem je turnajový mechanismus nejpoužívanější selekční metodou v reálných aplikacích [26].

I v turnajové selekci můžeme zvýšit selekční tlak a to zvýšením počtu jedinců v turnaji. Čím více jedinců zapojíme do turnaje, tím vyšší je selekční tlak. Se zvýšeným selekčním tlakem však roste pravděpodobnost, že vítězný jedinec je zároveň tím nejsilnějším z celé populace. Zvyšování selekčního tlaku může proto i v této metodě způsobovat problém předčasné konvergence. Na druhou stranu, při nízké velikosti turnaje bude evoluce probíhat pomalu.

3.3.2 Křížení

Je rekombinačním operátorem, který slouží k vytváření nových jedinců. Operátor křížení pracuje se dvěma jedinci (rodiči) a kombinuje jejich genetický materiál prohozením části jednoho rodiče s částí rodiče druhého. Vzniknou tak dva noví jedinci (potomci), kteří jsou umístěni do nové generace [20]. Genetické programování se od evolučních algoritmů významně liší v implementaci operátoru křížení a to výhodou, že je možné vytvořit pomocí křížení dva odlišné potomky dvou identických rodičů. Operátorů křížení existuje řada variant.

Základní varianta se nazývá *křížení podstromu* a aplikuje se následovně. Metodou selekce se vyberou dva jedinci jako rodiče. V každém rodiči se vybere bod křížení (uzel). Vybrané podstromy, které se nacházejí pod body křížení, se mezi oběma rodiči vymění. Výsledkem jsou dva noví jedinci (potomci), kteří jsou umístěni do nové generace. Operátor křížení je graficky zobrazen na 3.4.



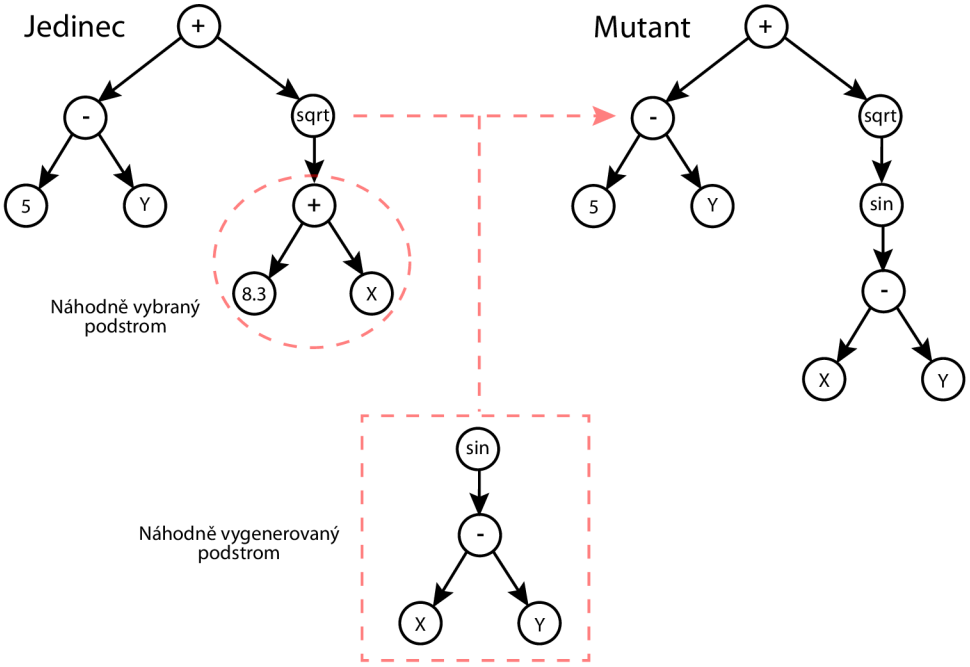
Obrázek 3.4: Ukázka použití operátoru křížení.

3.3.3 Mutace

Je základní rekombinační operátor, který pracuje s jedním jedincem. Na rozdíl od operátoru křížení je operátor mutace schopen zanést do systému nové informace. Mutace je obvykle náhodná a její pravděpodobnost je velmi nízká, pro zajímavost Koza demonstroval, že mutace není nutná, tedy P_m=0 [20] nebo je doporučena velmi malá, např. P_m=0,05 [3]. Operátory mutace existují opět ve velkém množství variant, které se liší problém od problému.

Nejčastěji používaný typ mutace se nazývá *mutace podstromu*, který pracuje následovně. Náhodně vybere jeden uzel ve stromu jedince (tzv. bod mutace) a tento uzel nahradí nově vygenerovaným stromem. Generování nového stromu probíhá stejným způsobem jako generování stromů v počáteční populaci. Mutovaný jedinec je následně vložen do nové generace. Příklad mutace podstromu je uveden na obrázku 3.5.

Jiným příkladem může být *bodová mutace*, kdy je náhodně vybraný uzel nahrazen jiným primitivem o stejné aritě [12].



Obrázek 3.5: Ukázka použití operátoru mutace podstromu.

3.4 Algoritmus genetického programování

Detailní popis principu genetického programování je zobrazen diagramem C.1 v příloze.

```
procedure genetic_algorithm()
begin
  inicializace = náhodné vytvoření počáteční populace;

  repeat {
    interpretace programu každého jedince a~vyhodnocení fitness;

    výběr jednoho nebo dvou jedinců z~generace s~pravděpodobností
    založenou na výsledcích fitness funkce;

    vytvoření nových jedinců aplikací genetických operátorů
    vybraných na základě určené pravděpodobnosti;
  }
  until = není nalezeno akceptovatelné řešení nebo jiná podmínka pro
  ukončení (př. dosažení maximálního počtu generací);

  return = výsledkem je nejlepší nalezený jedinec;
end
```

Algoritmus 3: Algoritmus genetického programování

3.5 Symbolická regrese

Genetické programování lze z principu aplikovat na spoustu oblastí. Nicméně u mnoha problémů je cílem najít funkci, jež má nějaké požadované vlastnosti, např. funkce odpovídá cílovým hodnotám. Tento postup je všeobecně znám jako problém symbolické regrese a je to jedna z prvních aplikací genetického programování v praxi [20].

Pojem regrese znamená nalezení náhodné veličiny (tzv. závislá proměnná) na základě znalosti jiných veličin jako například odhadujeme-li ráno, jaké bude přes den počasí na základě aktuálního počasí venku a znalosti předpovědi počasí pro aktuální den.

Problém regresní analýzy je, že pokud shoda není dostatečná, experimentátor musí zkoušet použít rozdílné funkce do té doby, dokud není nalezen vhodný model pro zadaná data. Nejen, že je to pracné, ale i výsledky analýzy závisí do velké míry na dovednostech a vynalézavosti experimentátora. Například v mnoha aplikačních oblastech jsou tradičně použity pouze lineární a kvadratické modely, přičemž výsledky by mohly být lepší při použití komplexnějších modelů.

Symbolická regrese se snaží proniknout dál. Je založena na hledání takové funkce či programu, která ze zadaných vstupních hodnot vytvoří požadované výstupní hodnoty, aniž by měla jakékoliv předpoklady o struktuře této funkce [8, 20]. Vzhledem k tomu, že genetické programování žádné takové předpoklady nemá, je ideální adept pro tento typ „objevování“.

Příklad použití symbolické regrese

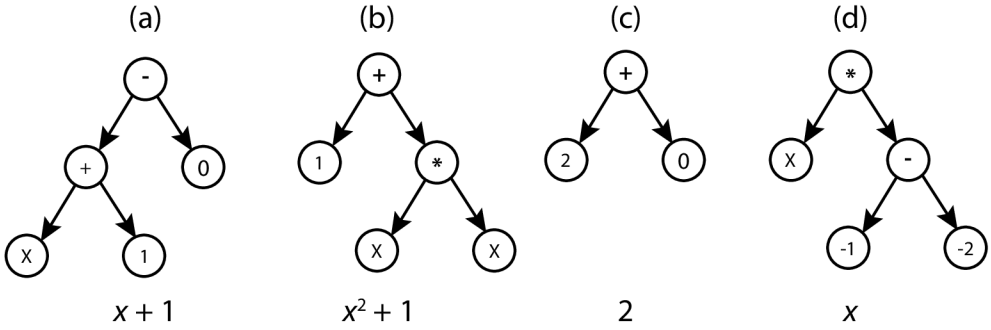
Najděte program (výraz), který nejlépe aproximuje zadané hodnoty (přejato z [33]):

x	y
-1.00	1.00
-0.80	0.84
-0.60	0.76
-0.40	0.76
-0.20	0.84
0.00	1.00
0.20	1.24
0.40	1.56
0.60	1.96
0.80	2.44
1.00	3.00

Hledáme program/matematický výraz, který odpovídá průběhu funkce v intervalu $\langle -1, 1 \rangle$. Trénovací množina představuje 11 dvojic (x, y), kde x (vstup) je hodnota proměnné a y (očekávaný výstup) je odpovídající funkční hodnota. V tomto demonstračním příkladu uvedu, že výsledkem je funkce/program $x^2 + x + 1$. V reálné aplikaci symbolické regrese však výsledný program není znám.

Cíl:	Nalezení programu, který podle zadané trénovací množiny generuje ze vstupní proměnné x očekávané výstupy y.
Množina terminálů:	T = $\{x$, náhodné konstanty $\langle -5, 5\rangle\}$
Množina funkcí:	F = $\{+, -, *, /\}$
Fitness hodnota:	Suma absolutních hodnot rozdílu (výstupy jedince/programu a odpovídající očekávané výstupy y)
Selekce:	Proporcionální selekce (roulette wheel selection)
Inicializace:	Metoda Ramped Half-and-Half (hloubka 1 - 2, 50% terminálů jsou konstanty)
Parametry:	Populace o velikosti $M = 4$, 50% křížení podstromu, 25% reprodukce, 25% mutace podstromu, není omezena velikost stromu
Ukončení:	Nalezení jedince, jehož suma absolutních odchylek je menší než 1

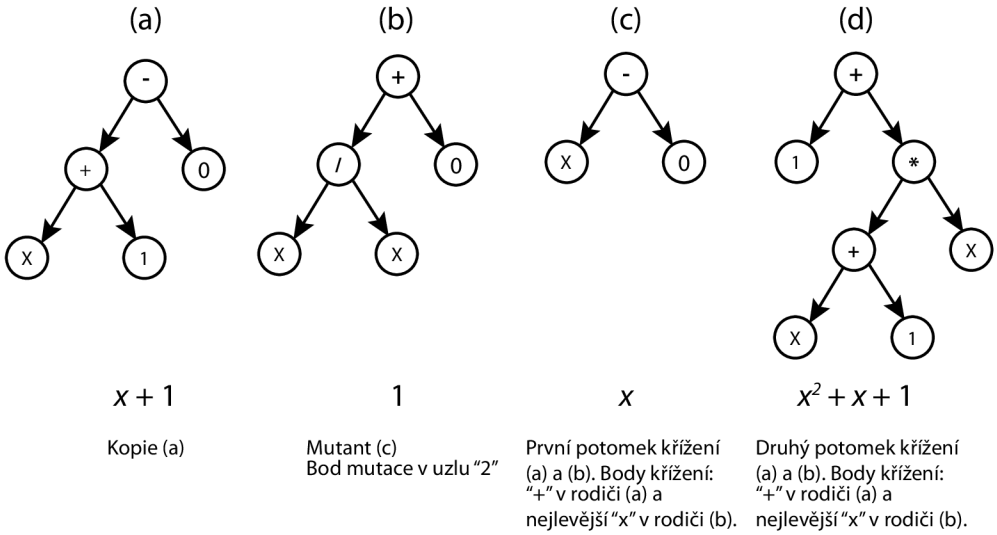
Tabulka 3.1: Definice problému pro GP



Obrázek 3.6: První náhodně vygenerovaná populace.

Jedinec	Reprezentace	Fitness
(a)	$x + 1$	0.67
(b)	$x^2 + 1$	1.00
(c)	2	1.70
(d)	x	2.67

Tabulka 3.2: Výpočet fitness pro vygenerovaná řešení



Obrázek 3.7: Druhá populace - nalezeno řešení (d).

3.6 Problémy genetického programování

Genetické programování může být z principu aplikováno na spoustu oblastí, avšak v oblastech kde známe analytické řešení, je stále výhodnější aplikovat klasické programování. V současné době díky výpočetním nárokům není GP schopno konstruovat složitější počítačové programy, natož programy, které by splnily Turingův test strojové inteligence. Pro tento obrovský úkol nemusí být GP připraveno po celé století [33].

Principy GP popírají základní vlastnosti klasického programování:

- **Správnost**: Řešení může být „dostatečně dobré".
- **Stálost**: Může být nalezeno více velmi rozdílných řešení.
- **Oprávněnost**: Řešení může být velmi nečitelné - jak a proč funguje.
- **Určitost**: Perfektní řešení nemusí být nikdy nalezeno.
- **Uspořádanost**: Řešení může být velmi nesystematické.
- **Stručnost**: Velké části řešení nemusí provádět žádnou činnost.
- **Rozhodnost**: Nemůžeme nikdy vědět, zda bylo nalezeno nejlepší řešení.

Genetické programování má také své problémy, některé z nich zmíníme dále:

Introny a bloat

Introny jsou takové části programu, které neplní žádnou funkci, avšak syntakticky jsou správné. Takovým příkladem může být např. výraz: $x = x + 0$. V programu jedince dochází v průběhu evoluce k nárůstu intronů, čímž narůstá počet uzlů programu, spotřeba paměti a doba výpočtu ohodnocení kandidátních řešení. Tento jev se označuje jako *bloat* [25].

Škálovatelnost

Nejsložitější programy vyevolvované pomocí GP obsahují řádově stoky uzlů. V porovnání s klasickými metodami programování je to málo, díky čemuž nebyl doposud pomocí GP zkonstruován příliš složitý program. Při evolvování složitých programů potřebujeme jedince o rozsáhlých strukturách, které implikují velké prostory, ve kterých je nutné prohledávat. Díky současným výpočetním nárokům GP v této situaci naráží na hranice svých možností a stává se neefektivní.

Fitness funkce

V případě genetických algoritmů (optimalizace parametrů) obvykle víme jakým způsobem porovnat nalezené a zadané hodnoty parametru. V případě GP kdy dochází k evolvování programu je tento proces složitější. Musíme totiž rozhodnout, zda je jeden algoritmus lepší než jiný, resp. zda se přibližuje k řešení problému. Obvyklým způsobem je porovnání výsledků programu s očekávanou hodnotou trénovací množiny. Ohodnocení jedince podle kvality jeho výstupu snadno umožní jedince prohlásit za kvalitního, i když je ze sémantického hlediska naprosto nevhodný. Nejvhodnějším způsobem ohodnocení by byla sémantická analýza kódu jedince, ale tento postup není obecně strojově možný.

Generalizace

Generalizace je v oblasti umělé inteligence obecně velice žádaná, avšak obvykle nedosažitelná vlastnost. Problém generalizace označuje situaci, kdy vyevolvovaný program pracuje správně s daty z trénovací množiny, avšak na testovacích datech selhává. Výsledkem evolučního procesu je pak jedinec s vysokou fitness hodnotou, který ale neřeší daný problém. Tento jev se nazývá *overfitting* [38].

Smyčky a rekurze

Smyčky a rekurze jsou základními principy většiny programovacích jazyků. Pokud chceme, aby programy konstruované pomocí GP měly dostatečnou vyjadřující sílu pro řešení komplexnějších úkolů, musí být smyčky a rekurze součástí množiny funkcí.

Problém iterací spočívá v možnosti zacyklení. Nekonečnou smyčku nejsme schopni teoreticky odhalit, viz problém zastavení Turingova stroje [9]. V problematice GP to znamená, že budeme čekat na vyhodnocení programu, který uvíznul v nekonečné smyčce. Tento problém lze řešit zavedením mechanismu, který bude hlídat předem definované parametry pro smyčky a rekurze. Nejčastěji používanými jsou:

- maximální počet iterací, které smí každý program provést
- časový interval, ve kterém musí být iterace dokončena
- vhodná volba množin funkcí a terminálů, aby u řešeného úkolu nedošlo k zacyklení nebo dlouhým smyčkám

Kapitola 4

Kryptoanalýza s využitím genetického programování

Ze znalostí získaných o symetrických šifrovacích algoritmech a možnostech jejich kryptoanalýzy z kapitoly 2 a informací o genetickém programování 3 je možné zkonstruovat kryptoanalytický systém. Tento systém bude pro prolomení zvoleného symetrického šifrovacího algoritmu používat hnací sílu živé přírody, kterou je evoluce.

Návrh takového systému ilustruje diagram 4.1, kde jsou evoluční principy aplikovány pomocí genetického programování s reálným využitím metody symbolické regrese. V diagramu je pro lepší názornost použit algoritmus DES, avšak tento systém lze aplikovat pro kterýkoliv symetrický šifrovací algoritmus. Jediná změna bude u trénovací množiny, kde se bude lišit délka clear a cypher textu podle velikosti bloku, se kterým šifrovací algoritmus pracuje.

Princip navrženého kryptoanalytického systému je následující:

1. **Vyvoření trénovací množiny** - tento kryptoanalytický systém pracuje s KPA, je tedy nutné znát dvojice plain text a k němu odpovídající cypher text o délce šifrovaného bloku (př. pro DES $64b$). Dostatečná velikost N trénovací množiny by mohla být $N = 1000$ těchto dvojic. Ukázka vytvoření trénovací množiny je zobrazena v horní části diagramu.

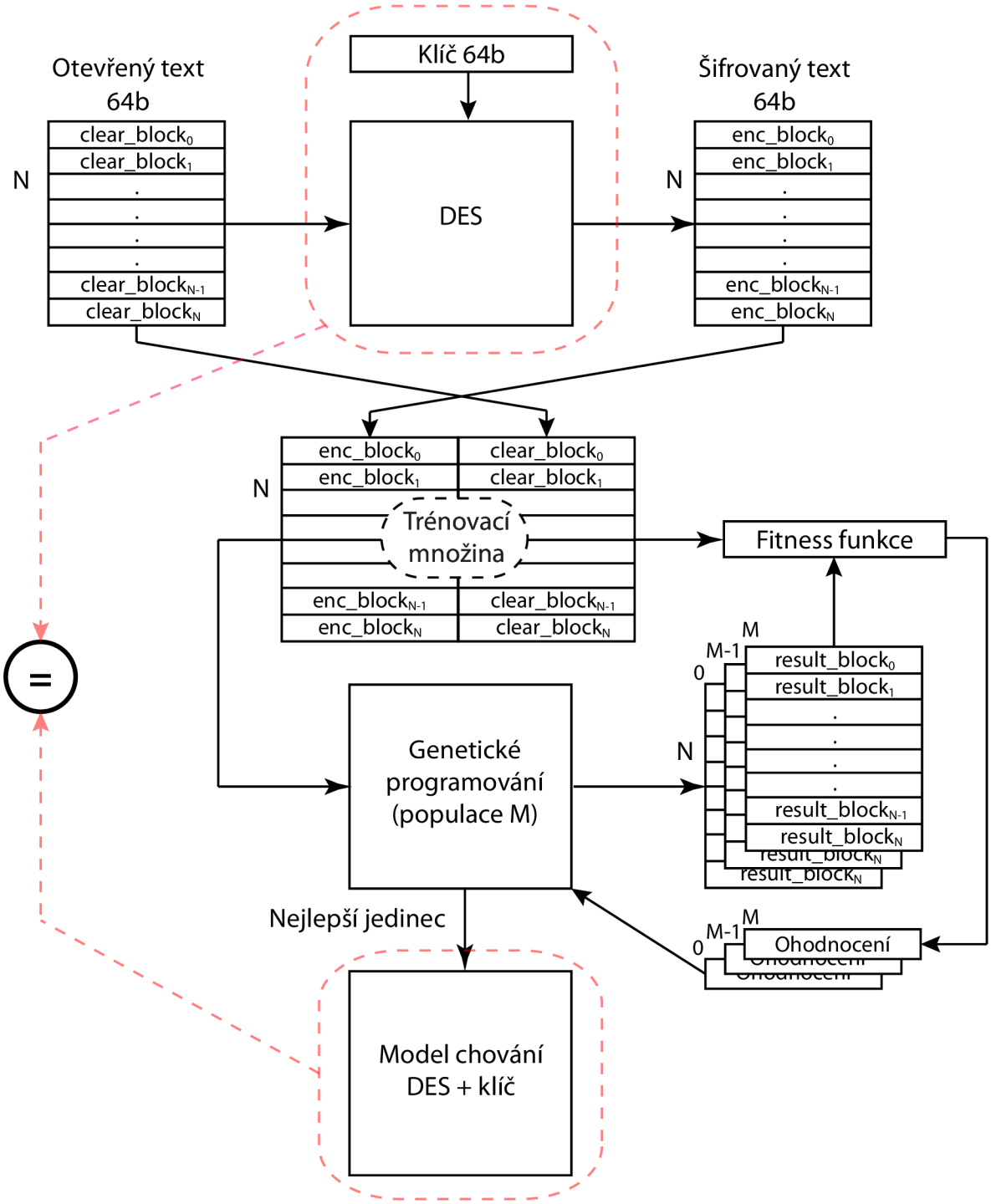
2. **Genetické programování** - evolučními principy jsou na základě dat z trénovací množiny konstruovány kvalitnější programy. Cílem interpretací programu je získat data, která jsou totožná s daty referenčními. K porovnání podobnosti dat a tím určení kvality programu/jedince slouží fitness funkce.

3. **Fitness funkce** - jejím základem je funkce $hDist$, která představuje Hammingovu vzdálenost dvou $64b$ řetězců A (referenční blok) a B (blok získaný interpretací programu), kde index i (počítáno od 1) reprezentuje pozici bitu v řetězci. Jedná se tedy o součet bitových rozdílů mezi dvěma bloky dat. Výsledná $fitness$ funkce je průměrem Hammingových vzdáleností na celé trénovací množině.

$$hDist = \frac{1}{64}\sum_{i=1}^{64} A_i \oplus B_i \tag{4.1}$$

$$fitness = \frac{1}{N}\sum_{i=1}^{N} hDist_i \tag{4.2}$$

4. **Nejlepší jedinec** - modeluje chování zvoleného šifrovacího algoritmu s klíčem, kterým byla vytvořena trénovací množina. Takový jedinec by měl být schopný při obdržení dalších zašifrovaných bloků od původního šifrovacího algoritmu se stejným klíčem vracet správný dešifrovaný blok dat.



Obrázek 4.1: Diagram modelace chování algoritmu DES pomocí genetického programování.

Kapitola 5

Návrh a implementace kryptoanalytického systému

Na základě zjištěných poznatků z teoretické části je programová realizace implementována jako grafické uživatelské rozhraní, protože některá řešení mají velký počet vstupních parametrů a v rámci experimentování je práce pomocí grafických ovládacích prvků rychlejší a přehlednější. Program je impelementován v jazyce `C++` a využívá objektového návrhu. Pro grafické uživatelské rozhraní byla použita knihovnu `QT` a jako vývojové prostředí bylo zvoleno IDE Netbeans.

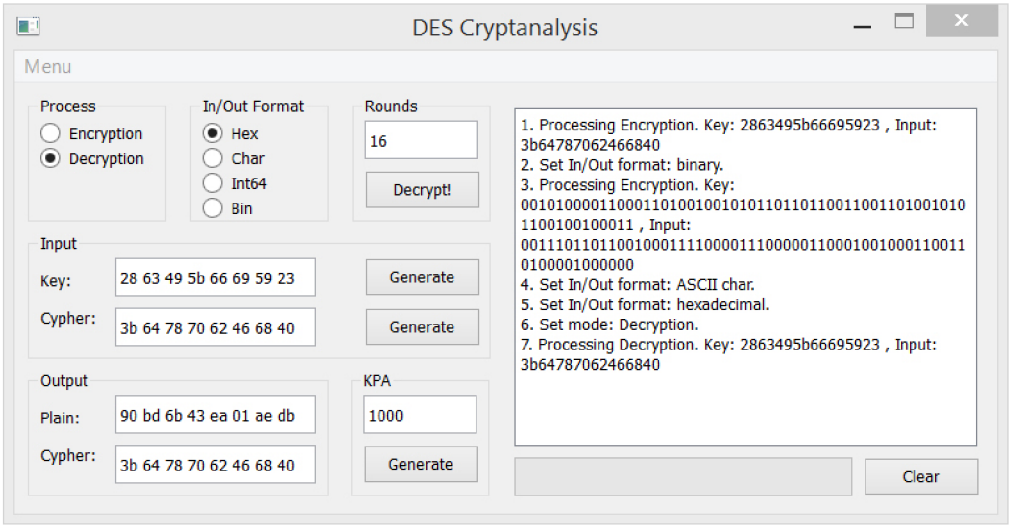
Celý program je rozdělen do třech samostatných částí, kde každá řeší příslušný problém. V této kapitole budou postupně popsány jednotlivé podprogramy. Sekce 5.1 popisuje realizaci algoritmu DES. Implementace lineární kryptoanalýzy je popsána v sekci 5.2 a kryptoanalytický systém využívající genetické programování popisuje sekce 5.3.

5.1 Algoritmus DES

Algoritmus DES byl zvolen jako základní symetrický šifrovací algoritmus této práce. Implementace byla převzata z [13] a následně byla provedena refaktorizace kódu rozdělením na samostatné funkce, mezí které patří permutace a výpočet funkce F. Tyto funkce nezávisle využívá lineární kryptoanalýza i genetické programování. Algoritmus DES byl dále upraven tak, aby bylo možné definovat počet kol, ve kterých probíhá šifrování popř. dešifrování, za účelem experimentování se šifrovací silou algoritmu.

Atributy algoritmu jako jsou konstanty, definice S-boxů a permutačních funkcí byly ověřeny podle zdroje [31], který definuje standardní implementaci algoritmu DES. Verifikace algoritmu byla provedena na základě chybového modelu definovaného panem Rivestem v článku [34]. Pan Rivest navrhnul chybový model a jednoduchý iterační algoritmus, který s tímto modelem pracuje. Spočítal všechny možné výskyty chyb v rámci všech operací, se kterými pracuje algoritmus DES. Na základě tohoto modelu vytvořil test, který při úspěchu zaručuje, že implementace algoritmu splňuje standardní specifikaci. Pseudokód tohoto iteračního algoritmu společně s očekávanými výstupy je součástí přílohy 9.

Program algoritmu DES umožňuje šifrování a dešifrování pro definovaný počet kol. Převody mezi číselnými soustavami, kdy je možné definovat binární, dekadickou, hexadecimální a ASCII reprezentaci. Dále umožňuje generování trénovacích množin známých otevřených textů, které jsou využívány u genetického programování.



Obrázek 5.1: Grafické uživatelské rozhraní algoritmu DES.

5.2 Lineární kryptoanalýza

Jako implementaci existujícího útoku jsem zvolil lineární kryptoanalýzu a to z několika důvodů. Na rozdíl od diferenciální kryptoanalýzy pracuje se známým otevřeným textem, který je totožně využíván v implementaci genetického programování. Dále se jedná o modernější typ útoku, který je se stejným principem aplikován na různé symetrické blokové šifrovací algoritmy a v neposlední řadě je efektivnější.

Základ lineární kryptoanalýzy byl popsán v kapitole 2.3.4. V této části se zaměříme na reálnou aplikaci zvoleného útoku. Sekce 5.2.2 popisuje postupy k získání nejlepších lineárních aproximací. V sekci 5.2.3 jsou popsány algoritmy pro odhalení části tajného klíče, které pracují s lineárními aproximacemi. Závěr kapitoly je vyhrazen pro sekci 5.2.4, která popisuje praktickou ukázkou útoku na 8-kolový DES a teoretický útok na standardní DES o 16ti kolech.

5.2.1 Notace a příprava

V celé této kapitole budeme využívat níže uvedené notace, kde IP, IP^{-1} a PC-1 jsou permutační funkce, se kterými pracuje algoritmus DES.

P	Data o velikosti 64 bitů po aplikaci IP; oteřený text.
C	Data o velikosti 64 bitů před aplikací IP^{-1}; šifrovaný text.
P_H, P_L	Horních 32 bitů P, dolních 32 bitů P.
C_H, C_L	Horních 32 bitů C, Dolních 32 bitů C.
X_r	Subdata o velikosti 32 bitů v kole r.
K	Data o velikosti 56 bitů po aplikaci PC-1; tajný klíč.
K_r	Subklíč o velikosti 56 bitů v kole r.
$F_r(X_r, K_r)$	Funkce F v kole r.
$S_a(x)$	V pořadí a-tý S-box.
$A[i]$	V pořadí i-tý bit symbolu A.
$A[i, j, ...k]$	$A_i \oplus A_j \oplus ... \oplus A_k$.

5.2.2 Nalezení lineárních aproximací

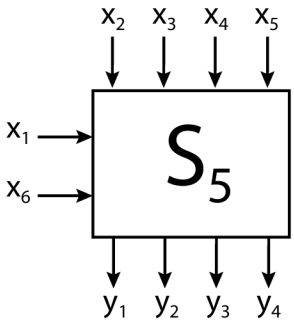
Základem lineární kryptoanalýzy je nalezení lineárních aproximací, které platí s pravděpodobností $p \neq \frac{1}{2}$. Definujme lineární aproximaci jako funkci nad $\mathbb{Z}_2$ s 1-bitovým výstupem, pro kterou platí, že výstup je roven XOR vstupu a celá rovnice platí s určenou pravděpodobností.

Příklad 5.1. $g(x_1, x_2, x_3) = x_1 \oplus x_3$

Pokud je funkce F algoritmu DES lineární, pak můžeme DES prolomit. Matsui se zaměřil ve funkci F na jednotlivé S-boxy, protože ty představují jediný nelineární prvek algoritmu DES. U jednotlivých S-boxů zkoumal korelace mezi vstupními a výstupními bity pro náhodné vstupní hodnoty. Podívejme se nyní na vlastnosti S-boxu S_5.

S-box S_5

Hodnoty 1, 2, 7, 11 se vyskytují pouze na levé straně. Hodnoty 4, 12, 13 se vyskytují 3x na levé straně. Hodnoty 8, 10, 14 se vyskytují 2x na každé straně. Hodnoty 0, 3, 5, 9, 15 se vyskytují pouze na pravé straně. Hodnota 6 se vykytuje 3x na pravé straně. XOR všech hodnot na levé straně je 1.



2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Tabulka 5.1: S-box S_5

S-box S_5 nevypadá náhodně a této vlastnosti Matsui využil. Aplikací různých bitových masek na vstup a výstup S_5 zjistil, že s pravděpodobností $\frac{52}{64} > 0.8$ platí následující rovnice

$$\overline{x_2} = y_1 \oplus y_2 \oplus y_3 \oplus y_4. \tag{5.1}$$

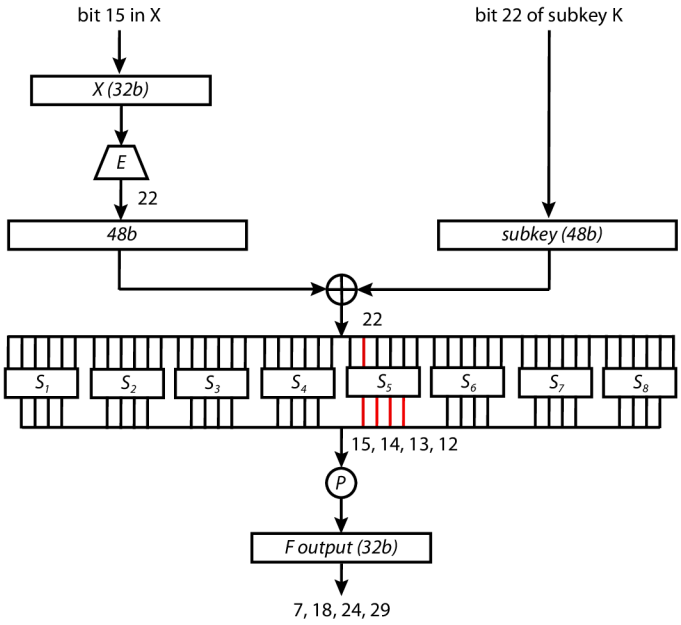
Funkce F

Funkce F je společně s S-boxy tvořena ještě permutačními funkcemi a operací XOR. Abychom lineární aproximaci S-boxu S_5 rozšířili na celou funkci F, musíme sledovat cestu jednotlivých bitů až ke vstupu a výstupu funkce F a aplikovat na ně příslušné operace, které tyto bity ovlivňují.

Výsledná ukázka funkce F, na kterou je rozšířena nalezená lineární aproximace, je zobrazen na diagramu 5.2.

Výstupy S-boxu S_5 jsou bity na pozicích 12, 13, 14 a 15, kde jako pozici bitu 0 předpokládáme nejméně významný bit odpovídajících dat. Tyto bity následně vstupují do permutační funkce P, která tyto bity přemapuje na bity na pozicích 7, 18, 24 a 29. Požadovaným vstupem S-boxu S_5 je bit na pozici 22. Stejným způsobem je nutné zpracovat expanzní permutaci E. Aby výstupem E byl požadovaný bit na pozici 22, musí do expanzní permutace E vstupovat bit na pozici 15. Požadovaný bit na pozici 22 je před vstupem do S-boxu

S_5 XORován s bitem podklíče. S bity podklíče se již žádné operace neprovádějí, proto je poslední potřebná proměnná určena bitem podklíče na pozici 22.



Obrázek 5.2: Schéma rozšíření lineární aproximace na funkci F.

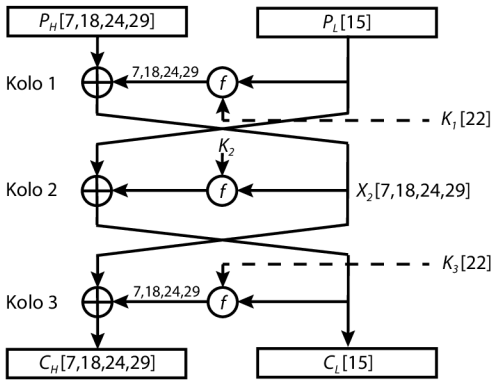
Nyní můžeme zapsat lineární rovnici, která aproximuje celou funkci F algoritmu DES v následujícím tvaru

$$output[7, 18, 24, 29] = \overline{X[15] \oplus K[22]}. \tag{5.2}$$

Lineární aproximace funkce F platí se stejnou pravděpodobností jako lineární aproximace S-Boxu S_5 a to $\frac{52}{64} > 0.8$.

Nalezení lineární aproximace pro 3-kolový DES

Dokážeme aproximovat funkci F lineární rovnicí, která platí s definovanou pravděpodobností. Nyní můžeme přistoupit k rozšíření lineární aproximace na celý algoritmus DES a představíme postup, jak získat nejlepší lineární aproximaci pro jeho 3 kola.



Obrázek 5.3: Lineární aproximace 3-kolového algoritmu DES.

Na diagramu 5.3 jsou popsány a doplněny jednotlivé vstupy a výstupy pro sestavení výsledné lineární aproximace 3-kolového algoritmu DES tak, jak je očekává funkce F v diagramu 5.2.

Nyní aplikujme rovnici 5.2 na první kolo algoritmu DES. Získáme tak rovnici, která platí s pravděpodobností $p = \frac{52}{64}$ a má tvar

$$X_2[7, 18, 24, 29] \oplus P_H[7, 18, 24, 29] = \overline{P_L[15] \oplus K_1[22]}. \tag{5.3}$$

Rovnici upravme tak, abychom měli na pravé straně pouze hledanou informaci o podklíči. Upravená rovnice platí s pravděpodobností $p = 1 - \frac{52}{64} = \frac{12}{64}$ a má tvar

$$X_2[7, 18, 24, 29] \oplus P_H[7, 18, 24, 29] \oplus P_L[15] = K_1[22]. \tag{5.4}$$

Stejným způsobem získáme rovnici pro poslední kolo algoritmu DES. Rovnice opět platí s pravděpodobností $p = \frac{12}{64}$ a má tvar

$$X_2[7, 18, 24, 29] \oplus C_H[7, 18, 24, 29] \oplus C_L[15] = K_3[22]. \tag{5.5}$$

Sečtením rovnice 5.4 a 5.5 získáme výslednou aproximaci 3-kolového algoritmu DES. Při sčítání nám z výsledné rovnice vypadne neznámá $X_2[7, 18, 24, 29]$, čímž odstraníme prostřední kolo šifrovacího algoritmu DES. Výsledkem je rovnice, u které se nevyskytují žádné neznámé a má tvar

$$P_H[7, 18, 24, 29] \oplus C_H[7, 18, 24, 29] \oplus P_L[15] \oplus C_L[15] = K_1[22] \oplus K_3[22]. \tag{5.6}$$

Tato rovnice platí s pravděpodobností $p = (\frac{12}{64})^2 + (1 - \frac{12}{64})^2 = 0.7$ pro náhodné páry otevřeného textu P a k nim odpovídajícího zašifrovaného textu C. Zároveň se jedná o nejlepší lineární aproximace 3-kolového algoritmu DES. Pan Matsui ve své práci [27] popisuje způsob, jak spočítat pravděpodobnost zvolené lineární aproximace s využitím *Lemma* 1.

Lemma 1. (Piling-up Lemma) Nechť $X_i (1 \leq i \leq n)$ jsou nezávislé náhodné proměnné jejichž hodnoty jsou 0 s pravděpodobností p_i nebo 1 s pravděpodobností $1 - p_i$. Pak pravděpodobnost, že $X_1 \oplus X_2 \oplus ... \oplus X_n = 0$ je

$$\frac{1}{2} + 2^{n-1} \prod_{i=1}^{n} \left(p_i - \frac{1}{2}\right). \tag{5.7}$$

Příklad 5.2. Pravděpodobnost, se kterou platí rovnice 5.6 může být spočítána také jako

$$\frac{1}{2} + 2\left(\frac{12}{64} - \frac{1}{2}\right)^2 = 0.695. \tag{5.8}$$

Proces získání lineárních aproximací algoritmu DES pro různý počet kol lze algoritmizovat a jednotlivé postupy jsou prezentovány ve zdroji [11]. Algoritmus DES je standardizovaný podle [31] a jednou nalezené nejlepší lineární aproximace jsou tak vždy použitelné a platné. Pan Matsui ve své práci [27] prezentuje nejlepší lineární aproximace pro různá kola algoritmu DES a tyto rovnice jsou společně s jejich pravděpodobnostmi součástí přílohy B.1.

5.2.3 Algoritmy pro získání části klíče

Pan Matsui ve své práci [27] prezentuje 2 typy algoritmů. Oba algoritmy pracují s nejlepšími lineárními aproximacemi algoritmu DES pro zvolený počet kol a jejich pravděpodobnostmi. Algoritmy se liší v úspěšnosti a také efektivitě nalezení počtu bitů klíče. Podstatou obou algoritmů je odhalení pouze určitého počtu bitů podklíče. V této sekci bude popsán princip Algoritmu 1 a jeho rozšíření, které zahrnuje Algoritmus 2.

Algoritmus 1

Je základním algoritmem lineární kryptoanalýzy algoritmu DES. Umožňuje nalezení pouze 1-bitové informace ve tvaru $K[i, j, ..., k]$.

Algoritmus pracuje s N páry otevřených a k nim odpovídajících šifrovaných textů a počítá levou stranu nejlepší lineární rovnice. Pro každý pár z N inkrementuje čítač, pokud se levá strana rovnice rovná bitové hodnotě 0. Po zpracování všech N párů, odhadne Algoritmus 1 podle stavu čítače a pravděpodobnosti p počítané lineární aproximace výsledek pravé strany rovnice. Pseudokód Algoritmu 1 je součástí přílohy 10.

Úspěšnost Algoritmu 1 závisí pouze na hodnotě N. Čím větší počet párů známého otevřeného textu máme, tím je algoritmus úspěšnější.

Pro zjištění potřebného počtu N párů otevřeného a odpovídajícího šifrovaného textu k provedení úspěšného útoku s využitím Algoritmu 1, využijeme *Lemma* 2 tak, jak ho popisuje pan Matsui.

Lemma 2. Nechť N je počet párů otevřeného a odpovídajícího šifrovaného textu a p je pravděpodobnost, se kterou platí lineární rovnice ve tvaru $P[i, j, ...k] \oplus C[l, m, ...n] = K[o, p, ...q]$. Jestliže je odchylka pravděpodobnosti od jedné poloviny $|p - 1/2|$ dostatečná, pak je úspěšnost Algoritmu 1 definovaná

$$\int_{-2\sqrt{N}|p-\frac{1}{2}|}^{\infty} \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} dx \tag{5.9}$$

Důsledek 1. Se stejným předpokladem jako *Lemma* 2, úspěšnost Algoritmu 1 závisí pouze na $\sqrt{N}|p-\frac{1}{2}|$. Obecně můžeme uvést, že k úspěšnému útoku potřebujeme znát $N \approx O(\varepsilon^{-2})$ otevřených textů, kde $\varepsilon = |p - \frac{1}{2}|$.

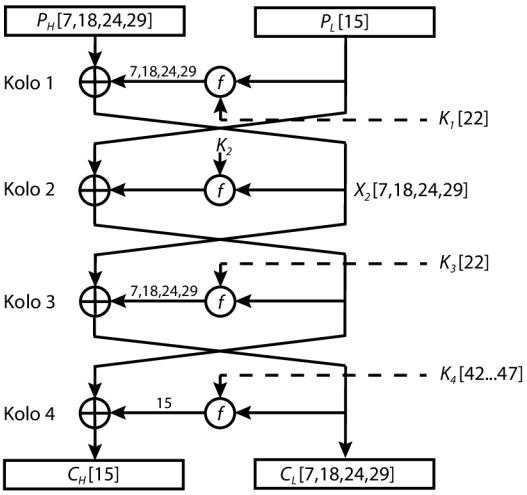
Tabulka 5.2 zobrazuje numerické výsledky rovnice 5.9 vypočtené panem Matsuiem.

N	$\frac{1}{4}\|p - \frac{1}{2}\|^{-2}$	$\frac{1}{2}\|p - \frac{1}{2}\|^{-2}$	$\|p - \frac{1}{2}\|^{-2}$
Úspěšnost	84.1%	92.1%	97.7%

Tabulka 5.2: Úspěšnost Algoritmu 1.

Algoritmus 2

Algoritmus 2 navazuje na Algoritmus 1 a zvyšuje jeho efektivitu ve smyslu nalezení výsledných bitů podklíče v posledním popř. prvním kolo DES. Pro aplikaci tohoto algoritmu je nutné aproximovat R-kolový DES pomocí lineární rovnice pro DES o $R - 1$ kolech. Podívejme se blíže na příklad 5.4, který zobrazuje, co se odehrává v posledním kole.



Obrázek 5.4: Lineární aproximace 4-kolového algoritmu DES.

Pro odvození rovnice 4-kolového DESu pro Algoritmus 2 využijeme nejlepší lineární aproximace DESu o 3 kolech, která platí s pravděpodobností $p \approx 0.7$ a má tvar

$$P_H[7,18,24,29] \oplus C_H[7,18,24,29] \oplus P_L[15] \oplus C_L[15] = K_1[22] \oplus K_3[22]. \tag{5.10}$$

Ve třetím kole se $C_H[7,18,24,29]$ beze změny přesouvá na $C_L[7,18,24,29]$ v kole čtvrtém. Jedinou neznámou tak ze třetího kola zůstává $C_L[15]$. Pokud se zaměříme na diagram 5.4, zjistíme, že $C_L[15]$ ve třetím kole získáme jako XOR $C_H[15]$ ve čtvrtém kolem a bit 15 funkce F_4 s neznámým podklíčem K_4 a C_L čtvrtého kola. Můžeme tedy psát $C_L[15] = C_H[15] \oplus F_4(C_L, K_4)$. Výsledná rovnice pro Algoritmus 2 má tedy tvar

$$\begin{aligned} &P_H[7,18,24,29] \oplus C_L[7,18,24,29] \oplus P_L[15] \oplus C_H[15] \oplus F_4(C_L, K_4)[15] \\ &= K_1[22] \oplus K_3[22]. \end{aligned} \tag{5.11}$$

Pokud budeme sledovat výstupní bit funkce $F_4[15]$ zjistíme, že je tento bit ve funkci F ovlivněn pouze 6ti neznámými bity podklíče $K_4[42,43,44,45,46,47]$). Pokud odhadneme těchto 6 bitů podklíče správně a zároveň platí lineární aproximace pro R-1 kol, musí jistě se stejnou pravděpodobností platit i lineární aproximace pro R kol. Právě této vlastnosti využívá Algoritmus 2. Úkolem je tedy počítat levou stranu rovnice pro všechna N a zkoušet všechny kombinace klíčů, v našem případě $2^6 = 64$ kombinací. Výsledkem je pak kandidátní klíč, se kterým celá rovnice odpovídá její očekávané pravděpodobnosti a protože známe všechny proměnné, můžeme spočítat i součet pravé strany rovnice. Pseudokód pro použití Algoritmu 2 v praxi je uveden v příloze 11.

Algoritmus 2 tedy dokáže odhalit několik bitů podklíče posledního kola a 1 bit jako součet pravé strany rovnice. Efektivita tohoto algoritmu je však vykoupena jeho nižší úspěšností jak dokládá tabulka naměřená panem Matsuiem 5.3. V porovnání s tabulkou 5.2 je pro zajištění 97.7% úspěšnosti Algoritmu 1 potřeba $|p - \frac{1}{2}|^{-2}$, u Algoritmu 2 je pro podobnou úspěšnost potřeba $8|p - \frac{1}{2}|^{-2}$ otevřeným známých textů.

N	$2\|p - \frac{1}{2}\|^{-2}$	$4\|p - \frac{1}{2}\|^{-2}$	$8\|p - \frac{1}{2}\|^{-2}$	$16\|p - \frac{1}{2}\|^{-2}$
Úspěšnost	48.6%	78.5%	96.7%	99.9%

Tabulka 5.3: Úspěšnost Algoritmu 2.

Symetrie algoritmu DES

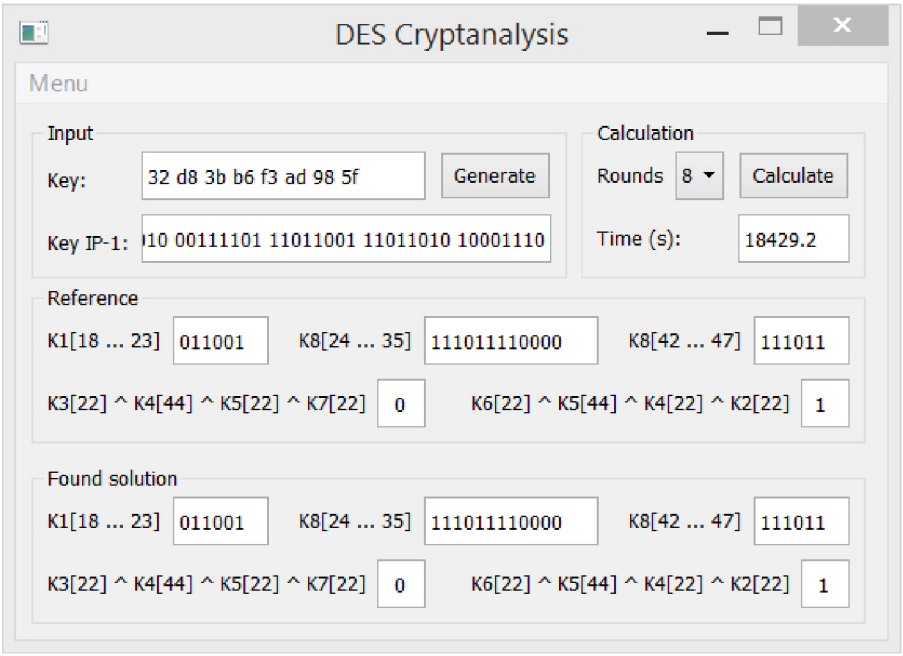
Algoritmus DES je symetrický, této vlastnosti využijeme pro odvození další rovnice z již nalezené nejlepší aproximace. Nová rovnice nám umožní odhalit další bity podklíče. Úprava již nalezené nejlepší lineární aproximace zahrnuje pouze prohození P a C a také nahrazení podklíčů ve smyslu K_i za $K_{rounds-i}$.

Dohledání klíče hrubou silou

Oba algoritmy odhalí pouze určitý počet bitů podklíče, ze kterých jsme následně schopni zrekonstruovat výsledné bity tajného klíče. Zbývající neznámé bity je vždy potřeba dopočítat pomocí útoku hrubou silou. Při lineární kryptoanalýze se proto snažíme nalézt co nejvíce bitů podklíče, aby byl samotný tajný klíč oslaben natolik, aby byl útok hrubou silou dosažitelný.

5.2.4 Ukázka útoku

Se znalostí principů Algoritmu 2 a nejlepších lineárních aproximací z přílohy B.1 můžeme provést samotný útok lineární kryptoanalýzou. Jako příklad jsem zvolil útok na 8-kolový DES, protože časová náročnost výpočtu je přiměřená a principiálně využívá stejné postupy jako útoky na DES o větším počtu kol. V závěru této části je popsán teoretický útok na standardní DES o 16 kolech. Útok na 8-kolový DES byl implementován také jako podprogram navrženého řešení a ukázka grafického uživatelského rozhraní je zobrazena na 5.5. Podprogram umožňuje simulovat útok na 8-kolový DES tak, jak je popsán v sekci 5.2.4 a pro ukázku praktické funkčnosti také umožňuje provést útok na DES o 4 kolech.



Obrázek 5.5: Grafické uživatelské rozhraní lineární kryptoanalýzy.

Útok na 8-kolový DES

Při útoku na 8-kolový DES využijeme nejlepší lineární rovnici, která aproximuje algoritmus DES o 6 kolech a použijeme ji pro aproximaci druhého až sedmého kola. Tato rovnice platí s pravděpodobností $\frac{1}{2} - 1.95 * 2^{-9}$ a má tvar

$$\begin{aligned} &P_H[7, 18, 24] \oplus C_H[7, 18, 24, 29] \oplus C_L[15] \\ &= K_2[22] \oplus K_3[44] \oplus K_4[22] \oplus K_6[22]. \end{aligned} \tag{5.12}$$

Rovnici upravíme do podoby, která aproximuje 8-kolový algoritmu DES a doplníme rovnice pro první a osmé kolo. Výsledná rovnice platí se stejnou pravděpodobností jako 5.12 a má tvar

$$\begin{aligned} &P_H[7, 18, 24] \oplus F_1(P_L, K_1)[7, 18, 24] \oplus C_H[15] \oplus C_L[7, 18, 24, 29] \\ &\oplus F_8(C_L, K_8)[15] = K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22]. \end{aligned} \tag{5.13}$$

Pomocí Algoritmu 2 budeme nyní hádat příslušné bity podklíče a také určíme informační hodnotu bitu na pravé straně. Tímto postupem získáme z rovnice 5.13 12 bitů podklíče: $K_1[18] \sim K1[23]$, $K_8[42] \sim K8[47]$ a 1-bitovou informaci: $K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22]$.

Nyní využijeme symetrie algoritmu DES popsanou v sekci 5.2.3 a aplikujeme ji na rovnici 5.13. Získáme tak další rovnici, díky které jsme schopni odhalit dalších 13 bitů podklíče. Výsledná rovnice platí se stejnou pravděpodobností jako rovnice 5.13 a má tvar

$$\begin{aligned} &C_H[7, 18, 24] \oplus F_8(C_L, K_8)[7, 18, 24] \oplus P_H[15] \oplus P_L[7, 18, 24, 29] \\ &\oplus F_1(P_L, K_1)[15] = K_6[22] \oplus K_5[44] \oplus K_4[22] \oplus K_2[22]. \end{aligned} \tag{5.14}$$

Za použití Algoritmu 2 můžeme z rovnice 5.14 získat 18 bitů podklíče: $K_1[18] \sim K1[23]$, $K_8[24] \sim K8[35]$ a 1-bitovou informaci: $K_6[22] \oplus K_5[44] \oplus K_4[22] \oplus K_2[22]$. Bity podklíče $K_1[18] \sim K1[23]$ však již známe z výsledku rovnice 5.13 a tuto informaci využijeme ke snížení počtu procházených kombinací a k získání 12 bitů podklíče $K_8[24] \sim K8[35]$.

Výsledkem výpočtů algoritmu 2 s použitím rovnic 5.13 a 5.14 získáme 26 bitů podklíče, které odpovídají 23 bitům tajného klíče, kde 3 bity podklíče jsou v reverzním procesu generování podklíčů a získání původního tajného klíče duplicitní. Výsledek odhalení části tajného klíče odpovídá bitům

$$\begin{aligned} &0, 1, 3, 5, 8, 11, 14, 15, 18, 20, 23, 24, 28, 31, 37, \\ &38, 41, 44, 46, 50, 53, 54, 2 \oplus 22 \oplus 26 \oplus 52. \end{aligned}$$

Zbývajících $56 - 23 = 33$ bitů tajného klíče můžeme nyní dohledat pomocí útoku hrubou silou, kde je nutné vyzkoušet zbývajících 2^{33} kombinací. Dosažené výsledky praktického útoku jsou popsány v tabulce 5.4.

N	2^{18}	2^{19}	2^{20}
Úspěšnost algoritmu	25.4%	86.5%	99.9%
Útok rovnice 5.13	3.7 min	5.2 min	9.6 min
Útok rovnice 5.14	114.3 min	182.5 min	307.2 min

Tabulka 5.4: Výsledky útoku na algoritmus DES o 8 kolech.

Teoretický útok na 16-kolový DES

Při tomto teoretickém útoku využijeme nejlepší lineární aproximace pro DES o 14ti kolech, která platí s pravděpodobností $\frac{1}{2} - 1.19 * 2^{-21}$ a má tvar

$$\begin{aligned} &P_L[7,18,24] \oplus C_H[7,18,24,29] \oplus C_L[15] = K_2[22] \oplus K_3[44] \oplus K_4[22] \\ &\oplus K_6[22] \oplus K_7[44] \oplus K_8[22] \oplus K_{10}[22] \oplus K_{11}[44] \oplus K_{12}[22] \oplus K_{14}[22]. \end{aligned} \tag{5.15}$$

Rovnici upravíme tak, aby aproximovala 16-kolový algoritmus DES a doplníme rovnice pro první a poslední kolo. Výsledná rovnice pak platí s pravděpodobností jako rovnice 5.15 a má tvar

$$\begin{aligned} &P_H[7,18,24] \oplus F_1(P_L,K_1)[7,18,24] \oplus C_H[15] \oplus C_L[7,18,24,29] \\ &\oplus F_{16}(C_L,K_{16})[15] = K_{14}[22] \oplus K_{13}[44] \oplus K_{12}[22] \oplus K_{10}[22] \oplus K_9[44] \\ &\oplus K_8[22] \oplus K_6[22] \oplus K_5[44] \oplus K_4[22] \oplus K_2[22]. \end{aligned} \tag{5.16}$$

Aplikací Algoritmu 2 na rovnice 5.16 získáme 12 bitů podklíče: $K_1[18] \sim K_1[23]$, $K_{16}[42] \sim K_{16}[47]$ a 1-bitovou informaci jako součet pravé strany rovnice. Opět využijeme symetrie a upravíme rovnici 5.16 do tvaru

$$\begin{aligned} &C_H[7,18,24] \oplus F_{16}(C_L,K_{16})[7,18,24] \oplus P_H[15] \oplus P_L[7,18,24,29] \\ &\oplus F_1(P_L,K_1)[15] = K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22] \oplus K_8[44] \oplus K_9[22] \\ &\oplus K_{11}[22] \oplus K_{12}[44] \oplus K_{13}[22] \oplus K_{15}[22]. \end{aligned} \tag{5.17}$$

Aplikací Algoritmu 2 na rovnice 5.17 získáme dalších 12 bitů podklíče: $K_1[42] \sim K_1[47]$, $K_{16}[18] \sim K_{16}[23]$ a 1-bitovou informaci jako součet pravé strany rovnice. Při útoku s využitím rovnic 5.16 a 5.17 tak celkově získáme 26 bitů podklíče, které po rekonstrukci odpovídají 26 bitům tajného klíče na pozicích

$$\begin{gathered} 0,1,3,4,8,9,14,15,18,19,24,25,31,32,38,39,41,42,44,45,50,51,54,55, \\ 5 \oplus 13 \oplus 17 \oplus 20 \oplus 46, 2 \oplus 7 \oplus 11 \oplus 22 \oplus 26 \oplus 37 \oplus 52. \end{gathered}$$

Zbývajících $56 - 26 = 30$ bitů tajného klíče můžeme dohledat pomocí útoku hrubou silou, kde je nutné vyzkoušet zbývajících 2^{30} kombinací.

Se znalostí *Lemma 2* a pravděpodobností rovnic 5.16 a 5.17 můžeme určit počet N párů otevřených a k nim odpovídajících šifrovaných textů k provedení útoku. Pro zajištění vysoké úspěšnosti Algoritmu 2 je $N = 16|1.19 * 2^{-21}|^{-2} \approx 2^{45}$. Samotný útok již není praktický s ohledem na výpočetní náročnost problému. Výsledky teoretického útoku prezentované panem Matsuiem jsou popsány v tabulce 5.5.

N	2^{43}	2^{44}	2^{45}
Úspěšnost algoritmu	10.6%	60.4%	98.8%

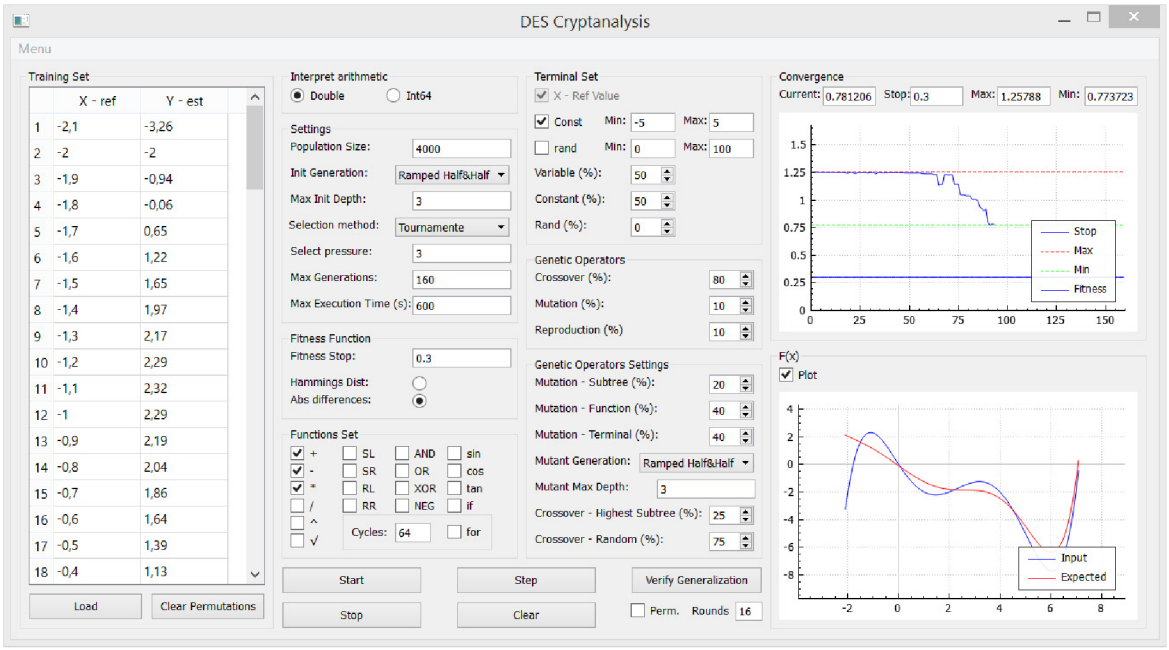
Tabulka 5.5: Předpokládané výsledky teoretického útoku na DES o 16 kolech.

Ze zjištěných poznatků o časové náročnosti výpočtu a počtu N, které je v praxi nepoužitelné vyplývá, že pro prolomení algoritmu DES o 16ti kolech, se stále jako nejlepší varianta útoku jeví standardní útok hrubou silou.

5.3 Kryptoanalýza s využitím GP

Na základě zjištěných poznatků z genetického programování a teoretického návrhu odpovídajícího kryptoanalytického systému, byl implementován potřebný simulační nástroj. Grafické uživatelské rozhraní simulátoru je zobrazeno na 5.6.

Tato kapitola je rozdělena do dvou sekcí. Sekce 5.3.1 popisuje jednotlivé implementační části simulátoru a zajímavé problémy, které se v rámci technického řešení vyskytovaly. Sekce 5.3.2 se zabývá validací funkčnosti genetického programování pomocí definovaných testů.



Obrázek 5.6: Grafické uživatelské rozhraní kryptoanalýzy s GP.

5.3.1 Implementace genetického programování

V této sekci jsou postupně popsány jednotlivé implementační části simulátoru. A to od datové reprezentace, množiny terminálů a funkcí, přes generování výchozí populace a genetické operátory až po paralelizaci výpočtu či ovládací prvky simulace.

Datová reprezentace

Programové řešení simulátoru pracuje se dvěma datovými typy.

- Datový typ `double` byl zvolen z důvodu testování a ověření funkčnosti genetického programování.
- Datový typ `u_int64` vychází ze znalosti algoritmu DES, který pracuje s bloky o velikosti 64 bitů. Otevřený i šifrovaný blok textu je tedy reprezentován 64 bitovým celým číslem. Nad tímto datovým typem lze také efektivně provádět všechny matematické funkce a potřebné bitové operace.

K datovým typům se váže automatická volba fitness funkce, která pro typ `double` využívá heuristiky hrubé fitness podle rovnice 3.2. Pro typ `uint64_t` je heuristika určena průměrnou Hammingovou vzdáleností podle rovnice 4.2.

Terminální uzly

Reprezentují listy syntaktického stromu a mohou nabývat typu proměnná nebo konstanta. Jedním z parametrů GP je definice procentuální zastoupení těchto dvou typů, tedy v jakém poměru se při generování terminálních uzlů bude volit mezi proměnnou a konstantou.

- **Proměnná** je jediným typem uzlu, který nabývá hodnot ze vstupní trénovací množiny.
- **Konstanta** je typem uzlu, který má při vytvoření pevně nastavenou hodnotu. Tato hodnota k je náhodně generovaná v rozmezí $min \leq k \leq max$, kde proměnné min a max jsou definované parametry GP. Konstantní hodnota může být v průběhu evoluce měněna pomocí mutačního operátoru.

Funkční uzly

Představují neterminální uzly syntaktického stromu, kde je každý typ definován počtem parametrů a implementací chování, kterým parametry zpracovává a vyhodnocuje.

V průběhu implementace bylo vyřešeno několik problému jako dělení nulou, optimalizace operace modulo v rámci bitových posuvů a rotací nebo mutace funkčních uzlů, kdy dochází k nahrazení za funkční uzel se stejným počtem parametrů.

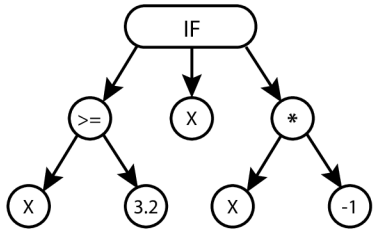
Funkční uzly mohou nabývat typů popsaných níže, kde je každý typ specifikován výrazem určujícím počet parametrů a způsob jejich vyhodnocení.

- **Matematické funkce** představují uzly typů `plus` ($arg_1 + arg_2$), `minus` ($arg_1 - arg_2$), `multiply` ($arg_1 * arg_2$), `division` (arg_1 / arg_2), `power` ($arg_1^{arg_2}$) a `root` ($\sqrt{arg}$).
- **Bitové operace** představují uzly typů shiftLeft ($arg_1 \ll arg_2$), shiftRight ($arg_1 \gg arg_2$), rotateLeft ($arg_1 \ll_R arg_2$), rotateRight ($arg_1 \gg_R arg_2$), AND ($arg_1 \,\&\, arg_2$), OR ($arg_1 \mid arg_2$), XOR ($arg_1 \oplus arg_2$), NEG ($\sim arg$).

Řídící struktury

Simulátor umožňuje kromě generování jednoduchých funkčních a terminálních uzlů generovat i řídící uzly `if` a `for`. Díky těmto dvěma typům nonterminálních uzlů získává jazyk genetického programování vysokou vyjadřovací sílu.

- **Uzel IF** umožňuje řídit tok programu v syntaktickém stromu jedince. Uzel očekává tři parametry a zpracovává je postupně zleva. První parametr je výsledkem porovnání a jeho logická hodnota rozhoduje, zda uzel IF předá svému předkovi druhý nebo třetí parametr. Příklad podmínky syntaktického podstromu je zobrazen diagramem 5.7 a jemu odpovídající pseudokód algoritmem 4.



Obrázek 5.7: Příklad syntaktického podstromu uzlu `if`.

Kořenový uzel podstromu porovnání předává uzlu `if` booleovskou hodnotu a může nabývat typů porovnání menší (`<`), menší rovno (`<=`), rovno (`==`), nerovno (`!=`), větší rovno (`>=`) a větší (`>`).

Programová implementace zajišťuje správné chování při aplikaci operátoru křížení i mutace. Jako uzel křížení může být vybrán kterýkoliv uzel podstromu kromě uzlu porovnání. Mutace nahrazuje porovnávací uzel za kterýkoliv jiný typ uzlu porovnání, zbývající uzly podstromu mohou mutovat libovolně včetně samotného uzlu `if`.

if $X >= 3.2$ **then**
  return X;
else
  return $X * (-1)$;
end

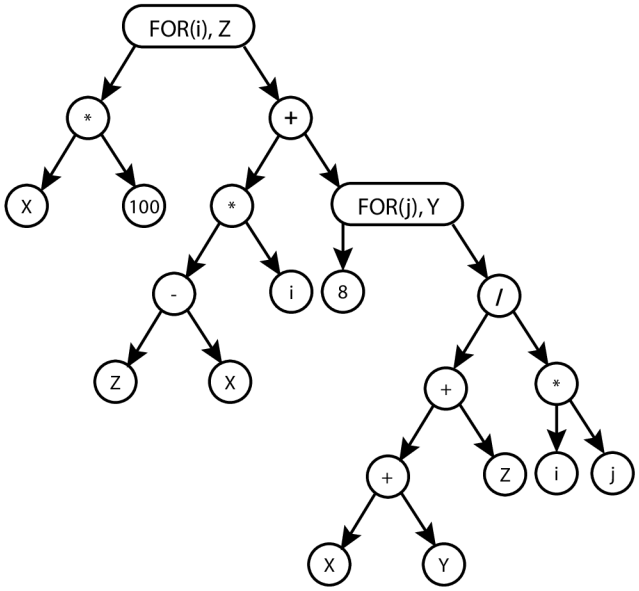
Algoritmus 4: Pseudokód uzlu `IF`.

- **Uzel FOR** umožňuje svojí konstrukcí zanést do syntaktického stromu proces opakovaného vyhodnocení. Maximální počet opakování je definován jako parametr GP.

  Uzel `for` očekává při vyhodnocení dva parametry. První parametr je modulován maximální povolenou hodnotou opakování a určuje počet cyklů. Druhý parametr představuje vyhodnocení samotného těla uzlu `for`.

  Příklad cyklu syntaktického podstromu uzlu `for` je zobrazen diagramem 5.8 a jemu odpovídající pseudokód algoritmem 5.

  Uzel `for` je svojí provázaností na různé proměnné velice komplikovaný pro operátor křížení a mutace, avšak implementační řešení všechny tyto stavy předpokládá a zajišťuje, že bude zachována správná syntaxe jazyka definovaného GP.



Obrázek 5.8: Příklad syntaktického podstromu uzlu `for`.

```
Z = X;
for i = 0 to (X * 100) mod 64 do
    Y = X;
    for j = 0 to 8 mod 64 do
        Y = (X + Y + Z)/(j * i);
    end
    Z = ((Z − X) * i) + Y;
end
return Z;
```

Algoritmus 5: Pseudokód uzlu `FOR`.

Součástí těla uzlu `for` mohou být všechny dostupné funkční uzly a stejně tak i řídící uzel `if`. Proměnné i, j, Y, Z a konstanta X z příkladu 5 tak mohou být v průběhu evoluce použity i jako vstupní parametry těchto dalších, neterminálních uzlů.

Interpret

Syntaktický strom každého jedince je tvořen z uzlů reprezentovaných objekty, které si drží reference na své následníky a svého předchůdce. Každý objekt typu uzel využívá dědičnosti a polymorfismu. Dědění probíhá od společného abstraktního uzlu a každý typ konkrétního uzlu přetěžuje metodu `evaluate()`, která příslušný typ uzlu vyhodnocuje. Rozšíření genetického programování o další typy funkčních nebo terminálních uzlů je díky tomuto návrhu velice jednoduché. Stačí vytvořit další typ uzlu a přetížit metodu `evalutate()` vlastní implementací.

Každý jedinec má jako jeden z atributů referenci na kořenový uzel, čímž má přístup k celé své genetické výbavě. Tento kořenový uzel slouží zároveň jako vstupní parametr interpretu. Algoritmus interpretu využívá rekurzivní preorder průchod syntaktickým stromem jedince a pro každý uzel volá metodu `evaluate()`. Výstupem interpretu je hodnota jedince, která odpovídá datovému typu, pro který byla simulace spuštěna. Pseudokód interpretu je popsán algoritmem 6.

```
Function interpret(class: node) :data_type
    foreach (node→parameters as descendant) do
        interpret(descendant);
    end
    return node→evaluate();
end
```

Algoritmus 6: Pseudokód interpretu GP.

Generování výchozí populace

Inicializace GP zahrnuje generování výchozí populace, pro kterou je možné v rámci parametrů zvolit metodu vytváření nových jedinců. Jako součást programového řešení byly implementovány všechny metody popsané v sekci 3.2.3.

Pro zvolenou metodu je nutné definovat parametr, který určuje maximální hloubku syntaktického stromu jedince. Posledním parametrem inicializace jedinců je určení velikosti samotné populace.

Selekční metoda

Výsledné řešení simulátoru poskytuje pro výběr jedinců pouze turnajovou selekci, protože v rámci experimentování měla tato metoda nejlepší výsledky a také se nejvíce přibližuje evolučním principům v živé přírodě.

Parametrem turnajové selekce je počet jedinců, kteří se turnaje účastní. Tento parametr tedy definuje selekční tlak. Čím více je jedinců v turnaji, tím vyšší je selekční tlak a řešení tak konverguje rychleji. S vysokým selekčním tlakem však roste pravděpodobnost uváznutí řešení v lokálním extrému.

Genetické operátory

Simulátor pracuje se všemi genetickými operátory popsanými v sekci 3.3 a pomocí volby parametrů významně rozšiřuje jejich možnosti. Pro GP je nutné definovat procentuální zastoupení genetických operátorů, tedy v jakém poměru budou aplikovány v průběhu evolučního procesu.

- **Reprodukce** využívá pouze zvolený selekční mechanismus a selekční tlak.
- **Křížení** umožňuje definovat poměr křížení mezi náhodně zvoleným uzlem křížení nebo mezi uzlem v hloubce 1, který představuje kořen podstromu s velkým množstvím genetického materiálu jedince.
- **Mutace** definuje poměr výběru mezi mutací terminálního uzlu, neterminálního uzlu nebo nahrazením náhodně zvoleného uzlu novým genetickým materiálem. Při generování nového podstromu je stejně jako u inicializace počáteční populace možné zvolit metodu generování podstromu a jeho maximální hloubku.

Paralelizace výpočtu

Genetické programování je náročné jak na výpočetní čas, tak na paměťové zdroje. V rámci zrychlení doby výpočtu je v programovém řešení využita vláknová paralelizace pomocí `Boost C++` knihovny. V implementaci genetického programování jsou výpočetně nejnáročnější dvě procedury, které je však možné efektivně paralelizovat.

- **Výpočet fitness funkce** probíhá rovnoměrným rozdělením počtu jedinců v aktuální generaci mezi definovaný počet vláken. Každé vlákno pak samostatně vyhodnocuje syntaktické stromy jedinců ve své množině. V rámci paralelizace tohoto problému nevyužívají vlákna žádnou sdílenou paměť pro zápis a pracují tak nezávisle na sobě.
- **Generování nové populace** rozděluje mezi definovaný počet vláken výpočetní úkol vytváření nových jedinců. Každé vlákno aplikuje podle definovaných parametrů genetické operátory na jedince v aktuální generaci a jejich potomky vkládá do generace nové. Nová generace je pro vlákna v tomto případě sdílenou pamětí a při zápisu potomků využívá program `mutex` zámku z knihovny `Boost C++`.

Pseudogenerátor náhodných čísel

Generování náhodných čísel je provázáno celým procesem výpočtu genetického programování. V průběhu každého experimentu je generováno velké množství náhodných čísel, kde celkové množství závisí na počtu generací v evolučním procesu, velikosti jedinců a dalších

parametrech GP. V průměru se v rámci jednoho experimentu vygeneruje od 2^{23} do 2^{27} náhodných čísel.

Jako generátory pseudonáhodných čísel byly pro programové řešení využity generátory pracující s uniformním rozložením z knihovny `Boost C++`.

Generování grafů

Simulátor v průběhu výpočtu vykresluje v reálném čase dva typy grafů, pro jejichž vizuální zobrazení byla využita knihovna `QTCumstomPlot`.

- **Graf konvergence** vykresluje pro jednotlivé generace nejlepší dosažené hodnoty *fintess*. V grafu je dále znázorněna nejhorší (max) a nejlepší (min) dosažená fitness hodnota. Vývoj fitness slouží k vizuálnímu ověření, zda proces evoluce konverguje k definované hodnotě *stop*.

- **Graf aproximace** vykreslí při spuštění simulace referenční graf odpovídající hodnotám x a y z trénovací množiny. Následně je vykreslována funkce $f(x)$, kde x je vstupní hodnota trénovací množiny a f reprezentuje funkci/program nejlepšího jedince v aktuální generaci. Pokud evoluce konverguje, pak se $f(x)$ nejlepšího jedince přibližuje k referenčnímu řešení. Graf aproximace je aktivní pouze pro simulaci s datovým typem `double`, protože vizualizace trénovací množiny algoritmu DES a funkce f nejlepšího jedince nepředstavuje vhodnou reprezentaci řešeného problému.

Odstranění permutace

Pro experimentování s konvergencí a generalizací v rámci trénovacích dat známých otevřených textů algoritmu DES umožňuje simulátor odstranit počáteční a koncovou permutaci. Výsledná trénovací množina je tedy transformována jako

$$\begin{aligned} input &= rev\_IP^{-1}(cyphertext) \\ output &= IP(plaintext). \end{aligned} \tag{5.18}$$

Ovládací prvky výpočtu

Samotná simulace evolučního procesu je řízena několika ovládacími prvky grafického uživatelského rozhraní.

- **Start** spustí automatickou simulaci evolučního procesu s definovanými parametry. Evoluční proces je zastaven splněním jedné z ukončujících podmínek výpočtu nebo pomocí ovládacího prvku Stop. Ukončující podmínky jsou součástí parametrů GP a pro každý experiment je možné definovat hodnotu maximální dosažené generace, omezení výpočetního času simulace a mezní hodnotu fitness hledaného řešení.

- **Stop** zastaví proces evoluce kdykoliv v rámci probíhajícího výpočtu a ponechá aktuální výsledky v paměti. V evoluci je možné pokračovat automaticky pomocí prvku Start nebo část výpočtu krokovat pomocí ovládacího prvku Step.

- **Clear** ruší všechny doposud vypočítané výsledky a uvádí simulátor do stavu, kdy je připraven pro opětovné spuštění.

- **Step** umožňuje krokovat evoluční proces po jednotlivých generacích a to v případě, kdy se simulátor nachází ve stavu zastavený nebo doposud nebyl spuštěn. Z kterékoliv generace je možné pokračovat v automatickém výpočtu pomocí prvku Start nebo výpočet ukončit pomocí ovládacího prvku Clear.
- **Verify generalization** vyhodnocuje úspěšnot nejlepšího jedince v generaci vzhledem k algoritmu DES s neznámým tajným klíčem. Tedy zda jedinec modeluje hledané řešení a pokud ano, tak s jakou úspěšností. Výsledek generalizace je vypisován na `stdout`. Proces ověření generalizace očekává dva parametry, počet kol algoritmu DES a zda byla trénovací množina zbavena počáteční a koncové permutace.

5.3.2 Ověření funkčnosti

Před samotnou kryptoanalýzou algoritmu DES s využitím genetického programování je nutné ověřit, zda všechny části genetického programování fungují v rámci definovaných parametrů. Pokud bude v procesu evoluce nalezen jedinec, který na zadané trénovací množině dosáhne definované fitness hodnoty, pak můžeme předpokládat, že pro zvolené parametry výpočtu funguje genetické programování správně.

Validační testy byly provedeny na sadě matematických funkcích zahrnující polynomy různých stupňů a goniometrické funkce. Uzly `IF` a `FOR` byly validovány na trénovací množině generované funkcí faktoriál. Zadání a naměřené výsledky prvního validačního testu jsou prezentovány níže. Další validační testy jsou součástí přílohy C.1.

Validační test

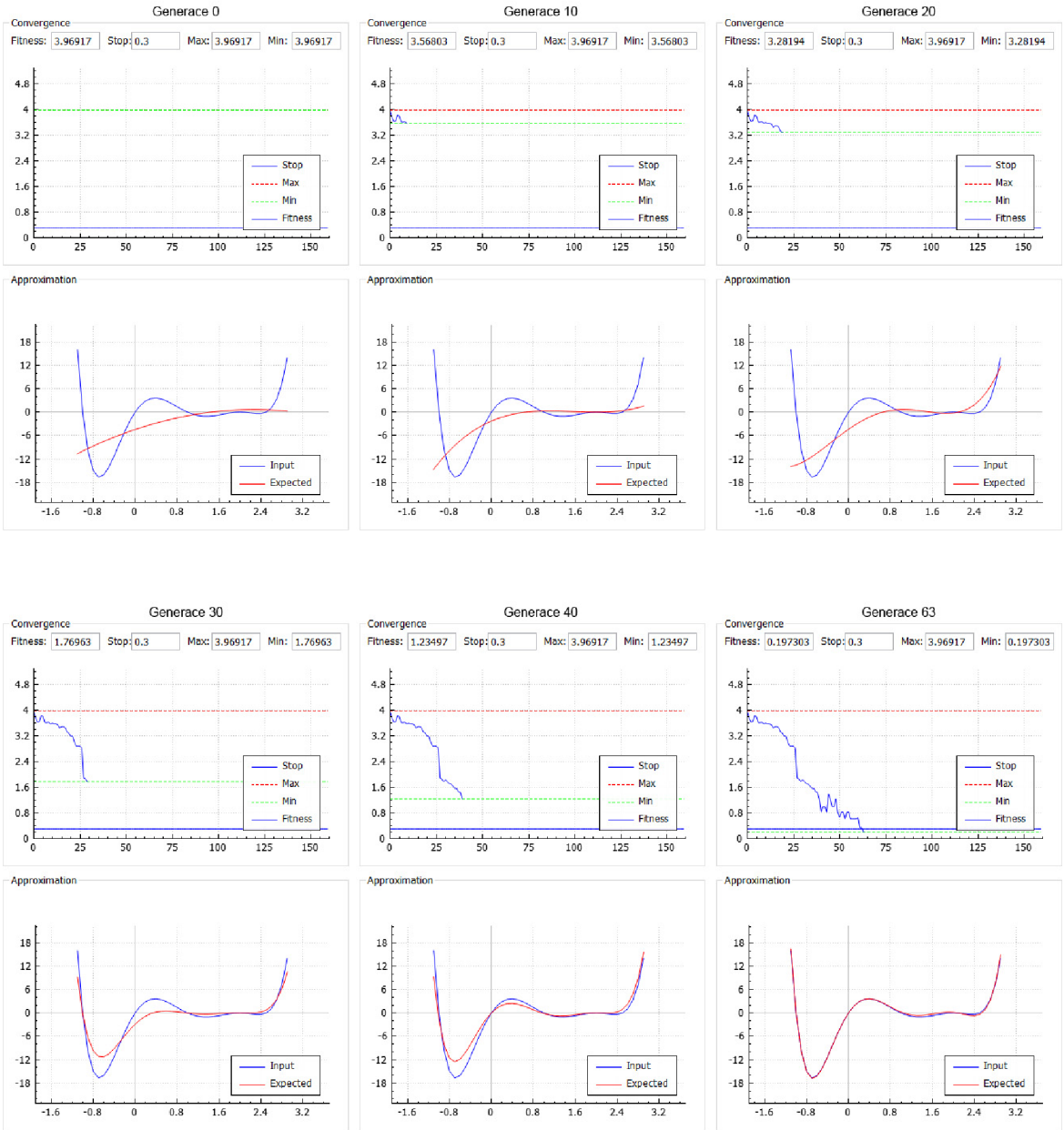
Trénovací množina je generována polynomem $2x^6 - 13x^5 + 26x^4 - 7x^3 - 28x^2 + 20x$. Pro zadanou trénovací množinu bylo v průběhu experimentování nalezeno nejlepší nastavení parametrů, které je součástí zadání validačního testu 5.6.

Cíl:	Nalezení programu, který podle zadané trénovací množiny generuje ze vstupní proměnné x očekávané výstupy y.
Množina terminálů:	T = $\{x$, náhodné konstanty $\langle -5, 5\rangle\}$
Množina funkcí:	F = $\{+, -, *\}$
Fitness hodnota:	Suma absolutních hodnot rozdílu (výstupy jedince/programu a odpovídající očekávané výstupy y)
Selekce:	Turnajová selekce (selekční tlak = 3)
Inicializace:	Metoda Ramped Half-and-Half (hloubka 2-3, 50% terminálů jsou konstanty)
Parametry:	Populace o velikosti $M = 4000$, 80% křížení, 10% reprodukce, 10% mutace, neomezená velikost stromu
Parametry křížení:	Křížení nejvyššího podstromu 50%, křížení náhodného podstromu 50%
Parametry mutace:	Mutace terminálů 45%, mutace funkcí 45%, mutace podstromu 10%, generování podstromu metodou Ramped Half-and-Half (hloubka 2-3, 50% terminálů jsou konstanty)
Ukončení:	Nalezení jedince se sumou absolutních odchylek menší než 0.3

Tabulka 5.6: Definice validačního testu pro GP.

Naměřené výsledky

Naměřené výsledky zobrazují pro zvolené generace odpovídající graf průběhu konvergence a graf aproximace. V průběhu výpočtu validačního testu 5.6 byly zaznamenány výsledky ze šesti generací a to z generace 0, 10, 20, 30, 40, a 63, ve které fitness hodnota nejlepšího jedince dosáhla definované mezní hodnoty fitness stop. Graf aproximace zobrazuje modrou barvou referenční hodnoty trénovací množiny a červená barva představuje výstup funkce/programu nejlepšího jedince v příslušné generaci.



Obrázek 5.9: Naměřené výsledky validačního testu.

Na základě naměřených výsledků z provedených validačních testů můžeme předpokládat, že všechny součásti genetického programování pracují správně.

Kapitola 6

Testování a zhodnocení dosažených výsledků

V rámci této kapitoly se budeme zabývat kryptoanalýzou algoritmu DES, která využívá genetické programování. Metoda symbolické regrese bude pracovat s trénovací množinou párů otevřeného textu P a šifrovaného textu C, generovanou algoritmem DES. Páry trénovací množiny budou reprezentovány datovým typem `uint_64`. Experimenty budou zaměřeny na různě velké trénovací množiny generované rozdílným počtem kol algoritmu DES.

Cílem této kapitoly je zkonstruovat procesem evoluce takového jedince, jehož program modeluje pro příslušnou trénovací množinu odpovídající chování algoritmu DES.

6.1 Předpoklady pro kryptoanalýzu s GP

V této části budou popsány jednotlivé předpoklady pro budoucí experimenty a také nejlepší nalezené parametry genetického programování.

6.1.1 Odhad hodnoty fitness

Kvalita jedince je přímo úměrná jeho hodnotě fitness. Heuristika fitness funkce je určena průměrnou Hammingovou vzdáleností, tedy průměrným bitovým rozdílem mezi výstupem jedince a odpovídající referenční hodnotou trénovací množiny. Jakých hodnot může fitness jedince v rámci experimentu nabývat?

Předpoklad 1. Každý bit náhodně vygenerovaného bloku dat může nabývat hodnot 1 nebo 0 s pravděpodobností $p = 0.5$. Uvažujme 2 různé náhodně vygenerované bloky dat X a Y o stejném počtu bitů N, pak pro jejich bitový rozdíl platí

$$count(X \oplus Y) \approx N * (1 - p), \tag{6.1}$$

kde *count* je funkce, která spočítá výskyt bitů s hodnotou 1.

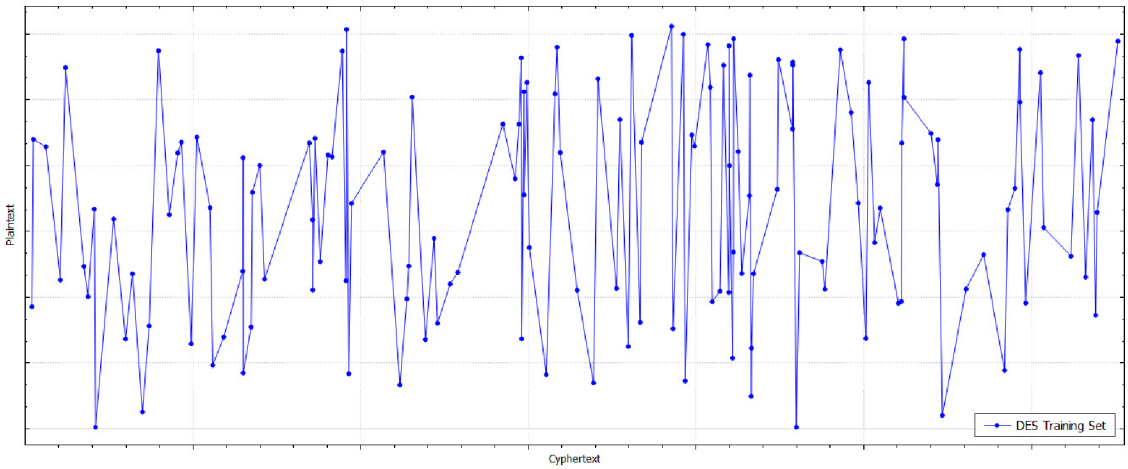
Předpoklad 2. U šifrovacího algoritmu pracujícím s bloky o velikost N by neměl existovat žádný vztah mezi otevřeným blokem P a jemu odpovídajícím šifrovaným blokem C. Můžeme tedy předpokládat, že bloky dat jsou vůči sobě náhodné. Z předpokladu 1 tedy musí platit rovnice 6.1, tedy že bitový rozdíl mezi P a C odpovídá vztahu $N * (1 - p)$, kde $p = 0.5$.

Důsledek 1. Algoritmus DES pracuje s bloky o velikosti $N = 64$ bitů. Při aplikaci rovnice 6.1 na velikost bloku N by tedy mělo platit, že průměrný bitový rozdíl mezi bloky otevřeného textu P a šifrovaného textu C odpovídá hodnotě ≈ 32 bitů.

Vizualizace trénovací množiny algoritmu DES

Představme si algoritmus DES v módu dešifrování jako funkci, která vstupní šifrovaný blok C transformuje na otevřený blok P. Bloky textu budeme reprezentovat datovým typem `uint_64`, kde je každý blok zastoupen odpovídající dekadickou hodnotu. Nyní můžeme průběh funkce DES v módu dešifrování zanést do grafu 6.1, který zobrazuje hodnoty pro 200 párů P a odpovídajícího C trénovací množiny generované algoritmem DES o 8 kolech s náhodně zvoleným tajným klíčem.

Graf nepředstavuje pro zvolenou heuristiku fitness funkce ideální reprezentaci. Avšak při vizuálním porovnání např. s průběhy funkcí polynomů můžeme předpokládat, že nalezení kvalitního jedince nebude triviální úkol. Vizuálně můžeme také ověřit, že mezi páry bloků P a C z trénovací množiny není očividně žádná souvislost. Průběh funkce DES tak odpovídá průběhu náhodně vygenerovaných dat.



Obrázek 6.1: Vizualizace průběhu funkce DES.

Data vykreslená v grafu 6.1 dostává v textové podobě genetické programování, které pracuje s metodou symbolické regrese. Nad těmito daty se snaží pomocí evolučních principů konstruovat takové jedince, kteří svojí programovou reprezentací transformují C na P, čímž prakticky modelují proces dešifrování algoritmu DES.

Praktické potvrzení předpokladů

Předpoklad 2 byl prakticky potvrzen pomocí jednoduchého iteračního algoritmu, který je popsán pseudokódem 7. Vstupem algoritmu je náhodný tajný klíč pro šifrování pomocí algoritmu DES a počet iterací výpočtu. Se zvyšujícím se počtem iterací stoupá statistická přesnost, v našem případě bylo použito 2^{20} opakování. Očekávaný výsledek pro DES pracující s bloky o velikosti 64 bitů je z předpokladu 2 hodnota ≈ 32 bitů, kterou algoritmus potvrdil.

```
Function convergence_assumption(attempts, key) :double
    int diff = 0;
    for (i = 0; i < attempts; i++) do
        plain = generate_random_plain();
        cypher = des_encrypt(plain, key);
        /* spočítá bitový rozdíl                    */
        diff += count(plain ⊕ cypher);
    end
    return diff/attempts;
end
```

Algoritmus 7: Pseudokód předpokladu bitového rozdílu P a C.

Důsledek 2. Výstupem každého jedince v rámci evolučního procesu je blok dat o velikosti 64 bitů. Pokud využijeme důsledku 1, kdy je GP nejméně úspěšné, tedy nenalezne souvislosti na trénovací množině a výstupy jedinců jsou náhodné, může být maximální rozdíl mezi výstupem jedince a referenčním blokem ≈ 32 bitů. Tato hodnota zároveň představuje nejhorší teoretické ohodnocení fitness jedince. U experimentů tedy budeme předpokládat, že pokud evoluce konverguje, fitness hodnota se zlepšuje od nejhorší možné hodnoty 32.

6.1.2 Nalezení nejlepších parametrů

Před samotnými experimenty potřebujeme určit parametry genetického programování. Nalezení optimálních parametrů GP vyžaduje dobré pochopení řešeného problému. Zaměříme se tedy na algoritmus DES, který je implementován pomocí cyklů, rotací a bitové operace XOR. Tyto typy operací zahrneme přednostně do množiny funkcí. Následně budeme experimentovat s různými trénovacími množinami, upravovat množinu funkcí a terminálů a obecné nastavení genetického programování. Hlavním ukazatelem při hledání optimálních parametrů byl vývoj grafu konvergence evoluce. Výsledkem experimentování bylo nalezení nejlepších parametrů GP popsané tabulkou 6.1.

Parametr	**Atribut**
Množina terminálů	T = {x, náhodné konstanty $\{0, 1, ..., 64\}$}
Množina funkcí	F = {+, −, `SL`, `SR`, `RL`, `RR`, `AND` `OR`, `XOR`, `FOR` max 64 cyklů}
Velikost populace	4000 jedinců, neomezená velikost stromu
Inicializace populace	Metoda Ramped Half-and-Half (hloubka 2-3, 50% terminálů jsou konstanty)
Selekční metoda	Turnajová selekce (selekční tlak = 5)
Fitness heuristika	Průměrná Hammingova vzdálenost
Genetické operátory	80% křížení, 10% reprodukce, 10% mutace
Parametry křížení	Křížení nejvyššího podstromu 50%, křížení náhodného podstromu 50%
Parametry mutace	Mutace terminálů 45%, mutace funkcí 45%, mutace podstromu 10%, generování podstromu metodou Ramped Half-and-Half (hloubka 2-3, 50% terminálů jsou konstanty)

Tabulka 6.1: Optimální parametry kryptoanalýzy DES s využitím GP.

6.2 Konvergence řešení

Tato sekce se zabývá měřením úspěšnoti konvergence fitness hodnot pro různě velké trénovací množiny, které byly generovány algoritmem DES o různém počtu kol. Součástí této sekce je také ověření, zda počáteční a koncová permutace ovlivňuje výsledky výpočtu.

6.2.1 Počáteční a koncová permutace

Každý blok C trénovací množiny je výsledkem procesu šifrování algoritmu DES se vstupním blokem P. Vstupní blok P je podle diagramu 2.9 v první kroku transformován počáteční permutací IP, následuje definovaný počet kol šifrování, výstup je opět transformován permutací IP^{-1}. Operaci odebrání počáteční a koncové permutace budeme rozumět operaci takovou, která transformuje vstupní trénovací množinu podle rovnice 5.18. V této části se pokusíme ověřit, zda má počáteční a koncová permutace nějaký vliv na dosažené výsledky fitness hodnot.

Předpoklad 3. Počáteční a koncová permutace jsou výpočetní operace, které zatěžují evoluční proces genetického programování nutností odhalit další závislosti algoritmu DES. Pokud tyto nadbytečné výpočetní operace z trénovací množiny odstraníme, měli by nad nimi konstruovaní jedinci dosahovat oproti původním trénovacím množinám lepších výsledků.

Metodika měření

Měření bude probíhat na trénovacích množinách o velikosti $N = 100$. Trénovací množina o velikosti N bude samostatně generována algoritmem DES o počtu kol $R \in \{2, 4, 8, 16\}$. Pro každou takovou trénovací množinu bude provedeno 50 měření a v průběhu evoluce bude v generaci $g = 50$ zaznamenána hodnota fitness F. Do tabulky 6.2 bude zanesen aritmetický průměr hodnot F na základě počtu provedených experimentů. Celé testování provedeme ještě jednou, tentokrát nad získanými trénovacími množinami aplikujeme operaci odebrání počáteční a koncové permutace.

Počet kol DES	**Permutace**	
	Součástí	**Odebrána**
2	28.47	28.40
4	28.51	28.54
8	28.54	28.52
16	28.43	28.38

Tabulka 6.2: Závislost permutací na výsledcích fitness hodnot.

Výsledky fitness hodnot naměřených na trénovacích množinách, které byly zbaveny počáteční a koncové permutace nepotvrzují oproti výsledkům původních trénovacích množin předpoklad 3.

Přestože transformace trénovací množiny odstraněním počáteční a koncové permutace nepřinesla lepší výsledky, rozhodnul jsem se zahrnout tuto operaci do množiny nejlepších parametrů genetického programování. Všechny další experimenty tedy předpokládají použití nalezených optimálních parametrů z 6.1 a odebrání IP a IP^{-1} z trénovacích množin.

6.2.2 Evoluce nejlepšího jedince

Cílem experimentů této části je vývoj nejlepšího jedince pomocí evolučních principů.

Předpoklad 4. Šifrovací síla algoritmu DES roste s počtem kol, které jsou při šifrování aplikovány. Pro GP by mělo být výpočetně jednodušší odhalit souvislosti algoritmu DES na trénovací množině generované s nižším počtem kol. Kvalita dosažené fitness hodnoty by tak měla být přímo úměrná počtu kol šifrování.

Předpoklad 5. Pokud GP dokáže odhalit souvislosti algoritmu DES, pak by velikost trénovací množiny neměla ovlivnit dosažené výsledky fitness hodnoty. Dosažená fitness hodnota programového řešení by tedy měla být podobná pro různě velké trénovací množiny, které byly generovány algoritmem DES o stejném počtu kol.

Metodika měření

Měření bude probíhat na množinách o velikosti $N \in \{10, 100, 1000\}$. Každá trénovací množina bude samostatně generována algoritmem DES o počtu kol $R \in \{2, 4, 8, 16\}$. Pro každou takovou trénovací množinu provedeme 50 měření. Při každém měření necháme evoluci konvergovat do stavu, kdy fitness dosáhne ustálené hodnoty.

Fitness hodnotu budeme považovat za ustálenou, pokud platí vztah

$$\left| \left(\frac{1}{10} \sum_{i=1}^{10} fitness_{(g-i)} \right) - fitness_g \right| < 0.02, \tag{6.2}$$

kde $fitness$ představuje hodnotu nejlepšího řešení v generaci g.

Z naměřených výsledků spočítáme aritmetický průměr podle počtu provedených měření a výsledky zaneseme do tabulky 6.3. Hodnoty v tabulce reprezentují průměrné nejlepší dosažené $fitness$ hodnoty. Tedy pro $N = 10$ a $R = 2$ odpovídá hodnota 19.5 kvalitám jedince takovým, že pro trénovací data dokáže převést libovolně zvolené C na P takové, které se oproti referenčnímu řešení liší z možných 64 bitů pouze průměrně v 19.5 bitech. Jedinec je tedy ze $\approx 70\%$ úspěšný pro dešifrování zvolené trénovací množiny. Průběhy konvergence nejlepších fitness hodnot jsou pro množiny N a počet kol R zobrazeny jako součást přílohy C.2.

Počet kol DES	**Počet párů trénovací množiny**		
	10	**100**	**1000**
2	19.34	27.85	30.81
4	19.41	28.01	30.72
8	19.50	27.79	30.77
16	19.33	27.83	30.73

Tabulka 6.3: Fitness hodnoty nejlepších nalezených jedinců.

Žádný z předpokladů 4 a 5 nebyl výsledky experimentů potvrzen. Úspěšnost konvergence fitness hodnoty nezávisí na počtu kol šifrování algoritmu DES, kterým byla trénovací množina generována. Za to lze pozorovat, že se s rostoucím počtem párů N trénovací množiny, zhoršuje nejlepší nalezená hodnota fitness. Výsledky měření nasvědčují tomu, že genetické programování pracuje úspěšně jen v rámci vstupní trénovací množiny a není schopné odhalit závislosti algoritmu DES.

6.3 Generalizace modelu

V této sekci se zaměříme na výsledné ověření problému generalizace, neboli s jakou úspěšností modelují nejlepší konstruovaní jedinci hledané řešení.

Předpoklad 6. Každý jedinec se v průběhu učení přizpůsobuje trénovací množině párů P_{ref} a C_{ref} a na základě kvality, jakou dokáže vyhodnotit C_{ref} obdrží odpovídající $fitness$ hodnotu podle vztahu

$$count(individual(C_{ref}) \oplus P_{ref}) = fitness, \tag{6.3}$$

kde *individual* představuje ohodnocovaného jedince a *count* funkci, která spočítá počet bitů s hodnotou 1.

Předpoklad 7. Nyní předpokládejme stejného jedince *individual* z předpokladu 6, avšak s rozdílem, že P je náhodně vygenerovaný otevřený blok, C je odpovídající zašifrovaný blok a C bylo šifrováno stejným tajným klíčem a počtem kol, jako trénovací množina, která obsahuje páry P_{ref} a C_{ref}. Pro výpočet $fitness$ hodnoty využijeme stejný vztah

$$count(individual(C) \oplus P) = fitness_g. \tag{6.4}$$

Pokud platí $32 \gg fitness_g \geq fitness$, pak jedinec hledané řešení generalizuje. Pokud $fitness_g = fitness$, pak jedinec představuje optimální řešení v rámci dosažené $fitness$.

Předpoklad 8. Pokud z rovnice 6.4 platí $fitness_g \approx 32$, pak můžeme předpokládat, že zkonstruovaný jedinec hledané řešení negeneralizuje.

Pro ověření generalizace byl navržen iterační algoritmus, který je součástí simulátoru genetického programování. Činnost algoritmu je popsána pseudokódem 8. Vstupem je tajný klíč, počet kol algoritmu DES, kterým byla vytvořena trénovací množina a počet opakování algoritmu. Se zvyšujícím se počtem opakování roste statistická přesnost výsledku. Pro experimentování byla určena jako dostačující hodnota 2^{20} opakování.

```
Function generalization(attempts, individual, key, rounds) :double
    solution = 0;
    for (i = 0; i < attempts; i++) do
        plain = generate_plain();
        cypher = des_encrypt(plain, key, rounds);
        /* odebrání permutací                                    */
        plain = IP(plain);
        cypher = rev_IP^-1(cypher);
        /* vyhodnocení jedince                                   */
        out = individual→evaluate(cypher);
        /* spočítá bitový rozdíl                                 */
        solution += count(out ⊕ plain);
    end
    return solution/attempts;
end
```

Algoritmus 8: Pseudokód algoritmu ověření generalizace.

Metodika měření

Měření bude probíhat na trénovacích množinách o velikosti $N \in \{10, 100, 1000\}$, kde výsledky pro každé N reprezentují postupně tabulky 6.4, 6.5 a 6.6. Každá trénovací množina o velikosti N bude samostatně generována algoritmem DES o počtu kol $R \in \{2, 4, 8, 16\}$. Pro každou takovou trénovací množinu bude provedeno 50 měření a v průběhu evoluce bude v generacích $g \in \{0, 25, 50, max\}$ spočítána hodnota generalizace G a zaznamenaná dosažená hodnota fitness F. Generace max představuje opět generaci, ve které bylo dosaženo ustálené hodnoty fitness. Do tabulky bude zanesen aritmetický průměr hodnot F a G z počtu provedených experimentů.

Počet kol DES	**Generace**							
	0		**25**		**50**		**max**	
	F	**G**	**F**	**G**	**F**	**G**	**F**	**G**
2	28.03	32.004	23.86	32.001	21.42	32.002	19.34	31.998
4	28.20	32.000	23.74	32.002	21.47	31.996	19.41	32.001
8	28.13	32.001	24.02	31.999	21.46	31.999	19.50	32.003
16	27.96	31.999	24.05	31.996	21.56	31.998	19.33	32.000

Tabulka 6.4: Výsledky generalizace pro $N = 10$.

Počet kol DES	**Generace**							
	0		**25**		**50**		**max**	
	F	**G**	**F**	**G**	**F**	**G**	**F**	**G**
2	30.67	31.999	29.50	32.000	28.46	32.000	27.85	32.002
4	30.78	32.002	29.48	32.001	28.50	32.000	28.01	31.997
8	30.62	31.998	29.42	31.996	28.58	31.999	27.79	32.001
16	30.71	32.003	29.37	32.000	28.42	32.002	27.83	31.999

Tabulka 6.5: Výsledky generalizace pro $N = 100$.

Počet kol DES	**Generace**							
	0		**25**		**50**		**max**	
	F	**G**	**F**	**G**	**F**	**G**	**F**	**G**
2	31.63	31.998	31.20	31.999	31.03	32.000	30.81	31.999
4	31.62	32.001	31.17	32.002	30.98	31.999	30.72	31.997
8	31.58	31.999	31.15	31.998	30.95	32.001	30.77	32.000
16	31.64	32.000	31.14	32.000	30.97	32.001	30.73	31.999

Tabulka 6.6: Výsledky generalizace pro $N = 1000$.

Z výsledků měření vyplývá, že nezávisle na velikosti trénovací množiny, počtu kol, kterým byly množiny generovány a generaci, ve které fitness hodnota dosáhla jakékoliv úspěšné úrovně, výsledek generalizace odpovídá ve všech případech rozdílu ≈ 32 bitů.

Shrnutí naměřených výsledků

Výsledky hodnot generalizace potvrzují předpoklad 8, tedy že nejlepší jedinci negeneralizují hledané řešení. Metoda symbolické regrese se ukázala jako úspěšná v rámci konvergence hledaného řešení na poskytnuté trénovací množině, avšak konstruovaní nejlepší jedinci nemodelují v jakémkoliv aspektu algoritmus DES v módu dešifrování. Úspěšnost chování nejlepšího nalezeného jedince se na základě předpokladu 1 podobá programovému chování, které generuje vůči očekávaným výstupům náhodné bloky dat.

Naměřené výsledky neúspěšné generalizace mohou být způsobeny několika faktory.

- **Parametry genetického programování** nemusejí představovat parametry optimální. Při experimentech tak nemusela být použita nejvhodnější množina funkcí, množina terminálů, nastavení GP nebo fitness funkce. Tento problém vychází ze samotných principů GP, mezi které patří správnost a určitost. Tyto principy říkají, že na základě zvolených parametrů nejsme schopni rozhodnout, zda je řešení dostatečně dobré nebo zda bylo nalezeno řešení nejlepší.

- **Problém škálovatelnosti** je jedním z obecných problému GP a uvádí, že v současné době jsou procesem evoluce konstruovány jednoduché programy obsahující řádově stovky uzlů. Přestože provedené praktické experimenty ve vysokých generacích konstruovaly jedince, jejichž syntaktické stromy obsahovaly až 6000 uzlů, dosahovaly maximální hloubky 70 a průměrné hloubky 40, nemusela tato programová reprezentace stačit na pokrytí potřebného programového vybavení hledaného řešení.

- **Složitost algoritmu DES** je oproti úkolům, které v současné době řeší GP, enormní. Výpočet algoritmu DES je parametrizován klíčem o velikosti 2^{56} bitů a společně tak implikují rozsáhlý programový prostor, ve kterém je za současných technických možností nalezení odpovídajícího modelu vysoce nepravděpodobné.

Kapitola 7

Závěr

Cílem této diplomové práce bylo nastudovat principy symetrických šifrovacích algoritmů včetně jejich současných kryptoanalytických metod, dále principy genetického programování a symbolické regrese. Na základě teoretických znalostí navrhnout možné využití genetického programování v oblasti kryptoanalýzy symetrických šifrovacích algoritmů. Spolu s tím z praktické části provést vybraný existující kryptoanalytický útok na zvoleném šifrovacím algoritmu. Hlavním cílem práce bylo implementovat navržený kryptoanalytický systém, který využívá genetické programování a metodu symbolické regrese a při praktickém útoku na zvoleném šifrovacím algoritmu pak ověřit jeho úspěšnost.

Během prací na programovém řešení jsem implementoval řadu zajímavých algoritmů z oblasti překladačů, umělé inteligence, kryptoanalýzy, statistiky a pravděpodobnosti. Součástí implementace byl také vhodný objektový návrh celého řešení, návrh grafického uživatelského rozhraní a různé paralelizace výpočtů.

Hlavní přínos mé práce shledávám v provedeném kryptoanalytickém útoku využívajícího metodu symbolické regrese a genetického programování. Útok byl proveden na symetrický šifrovací algoritmus DES a v rámci experimentů byly potvrzeny či vyvráceny definované předpoklady. Jako další přínos mohu zmínit návrh a implementaci simulačního nástroje pracujícího s genetickým programováním, který umožňuje provádět metodu symbolické regrese na libovolných vstupních datech bez zaměření na algoritmus DES.

Případný další vývoj mé práce může být zaměřen na další experimentování s kryptoanalýzou algoritmu DES využívající genetické programování. Experimenty se mohou zabývat různou strukturou syntaktických stromů ve smyslu generování uzlů do hloubky nebo pevnou definicí struktury jedinců při inicializaci populace. Funkční a terminální množiny mohou být rozšířeny o další typy uzlů. Syntaktické stromy jedinců mohou začít pracovat s automaticky definovanými funkcemi. Experimentování by mohlo probíhat v rámci dalších typů fitness funkcí jako např. N-gram nebo subsequence kernel. Ze závěru naměřených výsledků týkajících se složitosti algoritmu DES a problému škálovatelnosti však nepředpokládám, že by v rámci současných technických možností mohlo byt zkonstruováno řešení, které by bylo v praxi použitelné pro úspěšné dešifrování. Pokud budeme uvažovat platnost Moorova zákona, se zvyšujícím se výkonem bude GP schopna prohledávat větší programové prostory. Avšak na druhé straně se zvýšený výpočetní výkon projeví i na zvýšené bezpečnosti nových šifrovacích algoritmů, které tak budou implikovat rozsáhlejší programové prostory nutné k prohledání. Metoda symbolické regrese v rámci genetického programování tak pravděpodobně nebude nikdy vhodná pro řešení tohoto typu problému.

Literatura

[1] Affenzeller, M.; Winkler, S. M.; 0002, S. W.; aj.: *Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications.* CRC Press, 2009, ISBN 978-1-58488-629-7.

[2] Baker, J. E.: Adaptive Selection Methods for Genetic Algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985, ISBN 0-8058-0426-9, s. 101–111.

[3] Banzhaf, W.; Francone, F. D.; Keller, R. E.; aj.: *Genetic Programming: An Introduction: on the Automatic Evolution of Computer Programs and Its Applications.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, ISBN 1-55860-510-X.

[4] Biham, E.; Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. In *Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '90, London, UK, UK: Springer-Verlag, 1991, ISBN 3-540-54508-5, s. 2–21.

[5] Curtin, M.: *Brute Force: Cracking the Data Encryption Standard.* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005, ISBN 0387201092.

[6] Darwin, C.: *On the Origin of Species by Means of Natural Selection, Or, The Preservation of Favoured Races in the Struggle for Life.* J. Murray, 1859.

[7] Diffie, W.; Hellman, M. E.: Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *Computer*, ročník 10, č. 6, Červen 1977: s. 74–84, ISSN 0018-9162.

[8] Dostál, M.: *Evoluční výpočetní techniky.* Katedra informatiky, přírodovědecká fakulta Univerzita Palackého, Olomouc, 2007.
URL http://phoenix.inf.upol.cz/esf/ucebni/evt.pdf

[9] Češka, M.; Vojnar, T.; Smrčka, A.: *Teoretická informatika.* Studijní opora, Vysoké učení technické v Brně, Fakulta informačních technologií, 2013.

[10] Feistel, H.: Cryptography and computer privacy. *Scientific american*, ročník 228, 1973: s. 15–23.

[11] Forquest, R.; Loidreau, P.; Tavernier, C.: Finding good linear approximations of block ciphers and its application to cryptanalysis of reduced round DES. In *Proceedings of the International Workshop on Coding and Cryptography (WCC'09)*, 2009, s. 501–515.

[12] Goldberg, D. E.: *Genetic algorithms in search, optimization, and machine learning.* 2, Addison-Wesley, Reading, MA, 1989.

[13] Gonzalez, D. H.: *An approach to DES algorithm version 1.0.* Barcelona, ES, 2012. URL https://github.com/dhuertas/DES

[14] Hanáček, P.; Staudek, J.: Bezpečnost informačních systemů. *Metodická příručka. Úřad pro státní informační systém*, 2000.

[15] Hellman, M.; Merkle, R.; Schroeppel, R.; aj.: *Results of an Initial Attempt to Cryptanalyze the NBS Data Encryption Standard.* Information Systems Lab., Dept. of Electrical Eng., Stanford Univ.

[16] Holland, J. H.: *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University of Michigan Press, 1975, ISBN 0-262-58111-6.

[17] Hynek, J.: Genetic Algorithms in a Nutshell. *In Economics and Management*, ročník 5, 2002: s. 48–54, ISSN 1212-3609.

[18] Hynek, J.: *Genetické algoritmy a genetické programování.* Praha: Grada Publishing a.s., 2008, ISBN 978-80-247-2695-3.

[19] Katz, J.; Lindell, Y.: *Introduction to modern cryptography: principles and protocols.* CRC Press, 2007.

[20] Koza, J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA, USA: MIT Press, 1992, ISBN 0-262-11170-5.

[21] Koza, J. R.: *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, MA, USA: MIT Press, 1994, ISBN 0-262-11189-6.

[22] Koza, J. R.; Andre, D.; Bennett III, F. H.; aj.: *Genetic Programming 3: Darwinian Invention and Problem Solving.* Morgan Kaufman, 1999, ISBN 1-55860-543-6.

[23] Kvasnička, V.; Pospíchal, J.; Tiňo, P.: *Evolučné algoritmy.* Vydavatelství STU Bratislava, 2000, ISBN 80-227-1377-5.

[24] Lai, X.; Massey, J.: A Proposal for a New Block Encryption Standard. In *Advances in Cryptology - EUROCRYPT '90*, *Lecture Notes in Computer Science*, ročník 473, Springer Berlin Heidelberg, 1991, ISBN 978-3-540-53587-4, s. 389–404.

[25] Luke, S.; Panait, L.: A Comparison of Bloat Control Methods for Genetic Programming. *Evol. Comput.*, ročník 14, č. 3, Září 2006: s. 309–344, ISSN 1063-6560.

[26] Mařík, V.; Štěpánková, O.; Lažanský, J.: *Umělá inteligence (3).* Praha: ACADEMIA, 2001, ISBN 80-200-0472-6.

[27] Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In *Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology*, EUROCRYPT '93, Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1994, ISBN 3-540-57600-2, s. 386–397.

[28] Menezes, A. J.; Van Oorschot, P. C.; Vanstone, S. A.: *Handbook of applied cryptography.* CRC press, 1997, ISBN 0-8493-8523-7.

[29] Mitchell, M.: *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998, ISBN 0262631857.

[30] Morris, R.; Sloane, N. J. A.; Wyner, A. D.: Assessment of the NBS proposed Federal Data Encryption Standard. *Cryptologia*, ročník 1, č. 3, Červenec 1977: s. 281–291, ISSN 1558-1586.

[31] National Bureau of Standards: *Data Encryption Standard*, FIPS pub. 46, U.S. Department of Commerce, Leden 1977.

[32] Paar, C.; Pelzl, J.: *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer-Verlag New York Inc, 2010, ISBN 978-3-642-04100-6.

[33] Poli, R.; Langdon, W. B.; McPhee, N. F.: *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008, ISBN 1409200736, 9781409200734.

[34] Rivest, R. L.: Testing Implementations of DES, unpublished draft.

[35] Robling Denning, D. E.: *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., 1982, ISBN 0-201-10150-5.

[36] Safe, M.; Carballido, J.; Ponzoni, I.; aj.: On stopping criteria for genetic algorithms. *Advances in Artificial Intelligence–SBIA 2004*, 2004: s. 405–413.

[37] Schneier, B.: *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C*. New York, NY, USA: John Wiley & Sons, Inc., 1995, ISBN 0-471-11709-9.

[38] Schwarz, J.; Sekanina, L.: *Aplikované evoluční algoritmy*. Studijní opora, Vysoké učení technické v Brně, Fakulta informačních technologií, první vydání, 2006.

[39] Stallings, W.: *Cryptography and Network Security: Principles and Practice*. Upper Saddle River, NJ, USA: Prentice Hall Press, páté vydání, 2010, ISBN 0136097049, 9780136097044.

[40] Stinson, D. R.: *Cryptography: Theory and practice. 3rd ed.* Boca Raton, FL: Chapman & Hall/CRC Press, třetí vydání, 2006, ISBN 1-58488-508-4.

[41] Weise, T.: *Global Optimization Algorithms - Theory and Application*. Self-Published, druhé vydání, 2009.
URL http://www.it-weise.de/

Příloha A

Algoritmus DES

```
procedure des_test()
begin
  result = "9474B8E8C73BCA7D";    // vstup a zároveň klíč
  i = 0;

  repeat {
    // sudý krok šifruje, lichý krok dešifruje
    if (i % 2 == 0) {
      result = encrypt(result, result);
    } else {
      result = decrypt(result, result);
    }
    i++;
  } until (i < 16);

  if (result == "1B1A2DDB4C642438") return SUCCESS;
  else return FAILURE;
end
```

Algoritmus 9: Pseudokód validačního testu pro algoritmus DES.

i	Result	i	Result
0	9474B8E8C73BCA7D	9	739B68CD2E26782A
1	8DA744E0C94E5E17	10	2A59F0C464506EDB
2	0CDB25E3BA3C6D79	11	A5C39D4251F0A81E
3	4784C4BA5006081F	12	7239AC9A6107DDB1
4	1CF1FC126F2EF842	13	070CAC8590241233
5	E4BE250042098D13	14	78F87B6E3DFECF61
6	7BFC5DC6ADB5797C	15	95EC2578C2C433F0
7	1AB3B4D82082FB28	16	1B1A2DDB4C642438
8	C1576A14DE707097		

Tabulka A.1: Výsledky validačního testu.

Příloha B

Lineární kryptoanalýza

Počet kol	Lineární aproximace	Pravděpodobnost
3	$P_H[\alpha] \oplus P_L[15] \oplus C_H[\alpha] \oplus C_L[15] = K_1[22] \oplus K_3[22]$	$1/2 + 1.56 * 2^{-3}$
4	$P_H[\alpha] \oplus P_L[15] \oplus C_H[15] \oplus C_L[\alpha, \beta] = K_1[22] \oplus K_3[22] \oplus K_4[\gamma]$	$1/2 - 1.95 * 2^{-5}$
5	$P_H[15] \oplus P_L[\alpha, \beta] \oplus C_H[15] \oplus C_L[\alpha, \beta] = K_1[\gamma] \oplus K_2[22] \oplus K_4[22] \oplus K_5[\gamma]$	$1/2 + 1.22 * 2^{-6}$
6	$P_H[\delta] \oplus C_H[\alpha] \oplus C_L[15] = L_2 \oplus K_6[22]$	$1/2 - 1.95 * 2^{-9}$
7	$P_H[\delta] \oplus P_L[12, 16] \oplus C_H[\alpha] \oplus C_L[15] = K_1[19, 23] \oplus L_3 \oplus K_7[22]$	$1/2 + 1.95 * 2^{-10}$
8	$P_H[\delta] \oplus P_L[12, 16] \oplus C_H[15] \oplus C_L[\alpha, \beta] = K_1[19, 23] \oplus L_3 \oplus K_7[22] \oplus K_8[\gamma]$	$1/2 - 1.22 * 2^{-11}$
9	$P_H[15] \oplus P_L[\beta, \delta] \oplus C_H[15] \oplus C_L[\alpha, \beta] = K_1[\gamma] \oplus K_2[22] \oplus L_4 \oplus K_8[22] \oplus K_9[\gamma]$	$1/2 - 1.91 * 2^{-14}$
10	$P_L[\alpha] \oplus C_H[\alpha] \oplus C_L[15] = L_2 \oplus L_6 \oplus K_{10}[22]$	$1/2 - 1.53 * 2^{-15}$
11	$P_H[\alpha] \oplus P_L[15] \oplus C_H[\alpha] \oplus C_L[15] = K_1[22] \oplus L_3 \oplus L_7 \oplus K_{11}[22]$	$1/2 + 1.91 * 2^{-16}$
12	$P_H[\alpha] \oplus P_L[15] \oplus C_H[15] \oplus C_L[\alpha, \beta] = K_1[22] \oplus L_3 \oplus L_7 \oplus K_{11}[22] \oplus K_{12}[\gamma]$	$1/2 - 1.19 * 2^{-17}$
13	$P_H[15] \oplus P_L[\alpha, \beta] \oplus C_H[15] \oplus C_L[\alpha, \beta] = K_1[\gamma] \oplus K_2[22] \oplus L_4 \oplus L_8 \oplus K_{12}[22] \oplus K_{13}[\gamma]$	$1/2 + 1.49 * 2^{-19}$
14	$P_L[\delta] \oplus C_H[\alpha] \oplus C_L[15] = L_2 \oplus L_6 \oplus L_{10} \oplus K_{14}[22]$	$1/2 - 1.19 * 2^{-21}$
15	$P_H[\delta] \oplus P_L[12, 16] \oplus C_H[\alpha] \oplus C_L[15] = K_1[19, 23] \oplus L_3 \oplus L_7 \oplus L_{11} \oplus K_{15}[22]$	$1/2 + 1.19 * 2^{-22}$
16	$P_H[\delta] \oplus P_L[12, 16] \oplus C_H[15] \oplus C_L[\alpha, \beta] = K_1[19, 23] \oplus L_3 \oplus L_7 \oplus L_{11} \oplus K_{15}[22] \oplus K_{16}[\gamma]$	$1/2 - 1.49 * 2^{-24}$

Tabulka B.1: Nejlepší lineární aproximace algoritmu DES a jejich pravděpodobnosti.

Notace:

α : $7, 18, 24, 29$

β : $27, 28, 30, 31$

γ : $42, 43, 45, 46$

δ : $7, 18, 24$

L_i : $K_i[22] \oplus K_{i+1}[44] \oplus K_{i+2}[22]$

```
procedure matsui_algorithm1(aprox, KPA)
begin
  T = count(KPA);              // počet párů
  p = probability(aprox);      // pravděpodobnost lineární aproximace
  i = 0;
  counter = 0;                 // čítač úspěšnosti

  repeat {
    // podle vyhodnocení lineární aproximace inkrementuj čítač
    if (aprox(KPA[i]) == 0) counter++;
    i++;
  } until (i < T);             // projdeme postupně všechny páry

  if (counter > T/2) {   // podle hodnoty čítače odhadni výsledek
     if (p > 0.5) return 0 else return 1;
  } else {
     if (p > 0.5) return 1 else return 0;
  }
end
```

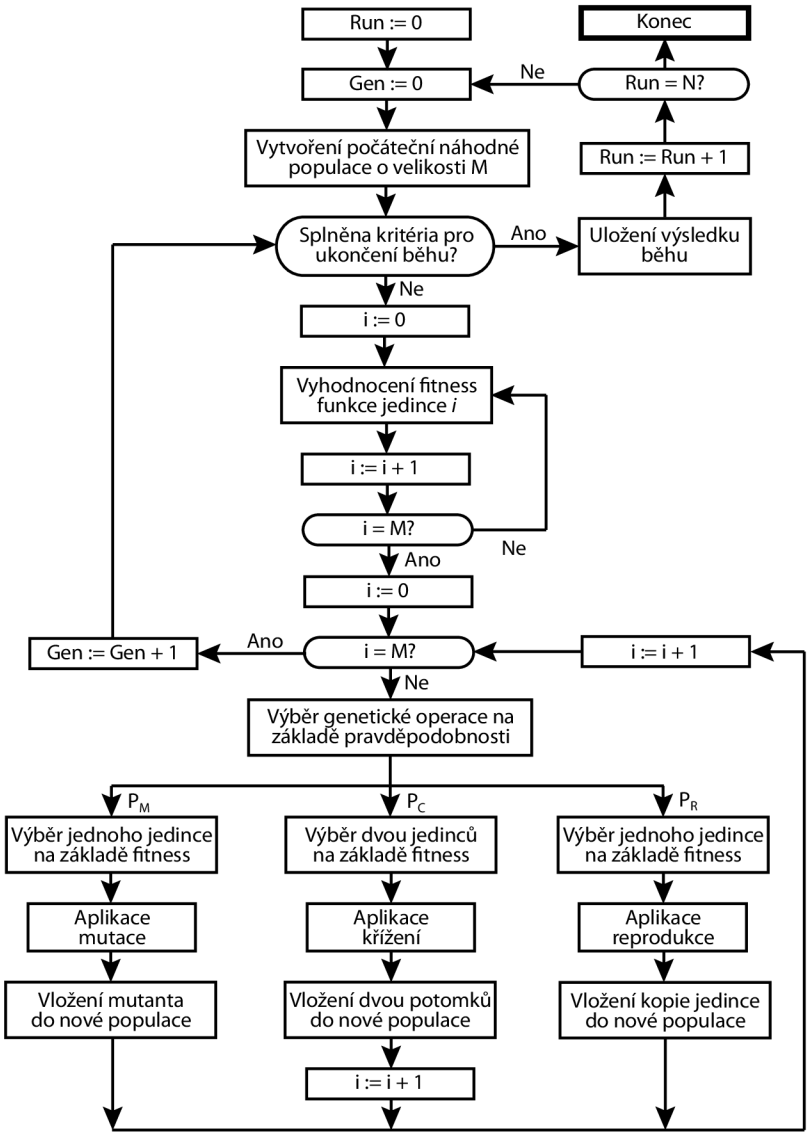
Algoritmus 10: Pseudokód Matsuiho Algoritmu 1

```
procedure matsui_algorithm2(aprox, KPA, combinations)
begin
  T = count(KPA);              // počet párů
  p = probability(aprox);      // pravděpodobnost lineární aproximace
  counter[combinations] = 0;  // čítače pro kandidátní klíče
  i = 0, k = 0;

  repeat {
    repeat {
      // podle vyhodnocení lineární aproximace inkrementuj čítač
      if (aprox(KPA[i], k) == 0) counter[k]++;
      k++
    } until (k < combinations);  // pro všechny kandidáty na klíč
    i++;
  } until (i < T);           // projdeme postupně všechny páry

  max = find_max(counter);  // najde čítač s max odchylkou od T/2;
  if (max > T/2) {           // podle hodnoty čítače odhadni výsledek
     if (p > 0.5) return (max.key, 0) else return (max.key, 1);
  } else {
     if (p > 0.5) return (max.key, 1) else return (max.key, 0);
  }
end
```

Algoritmus 11: Pseudokód Matsuiho Algoritmu 2

Příloha C

Genetické programování



Obrázek C.1: Diagram genetického programování.

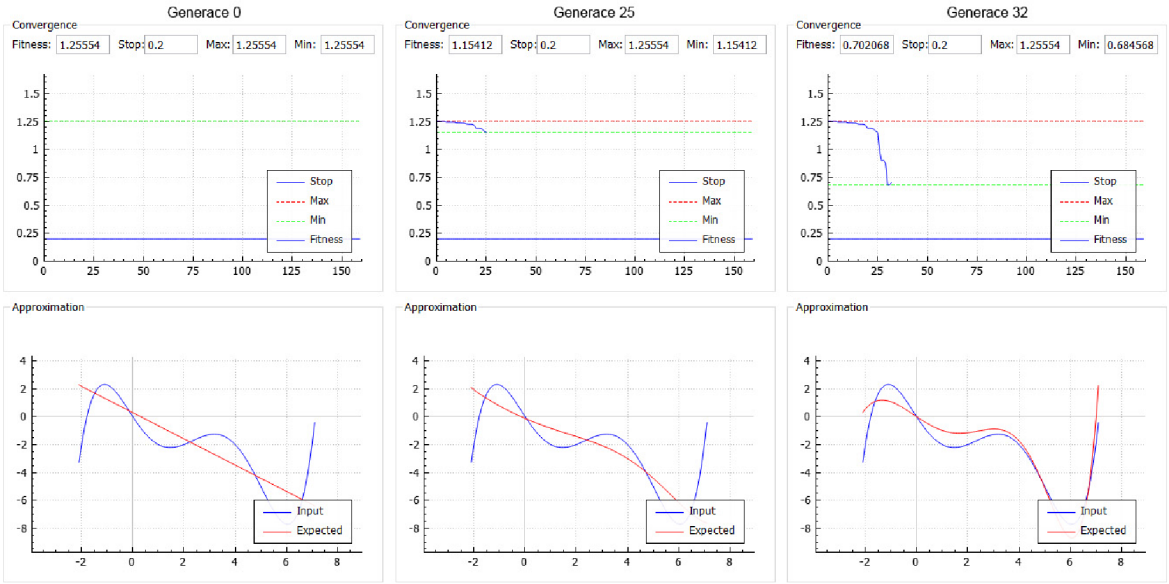
C.1 Validační testy

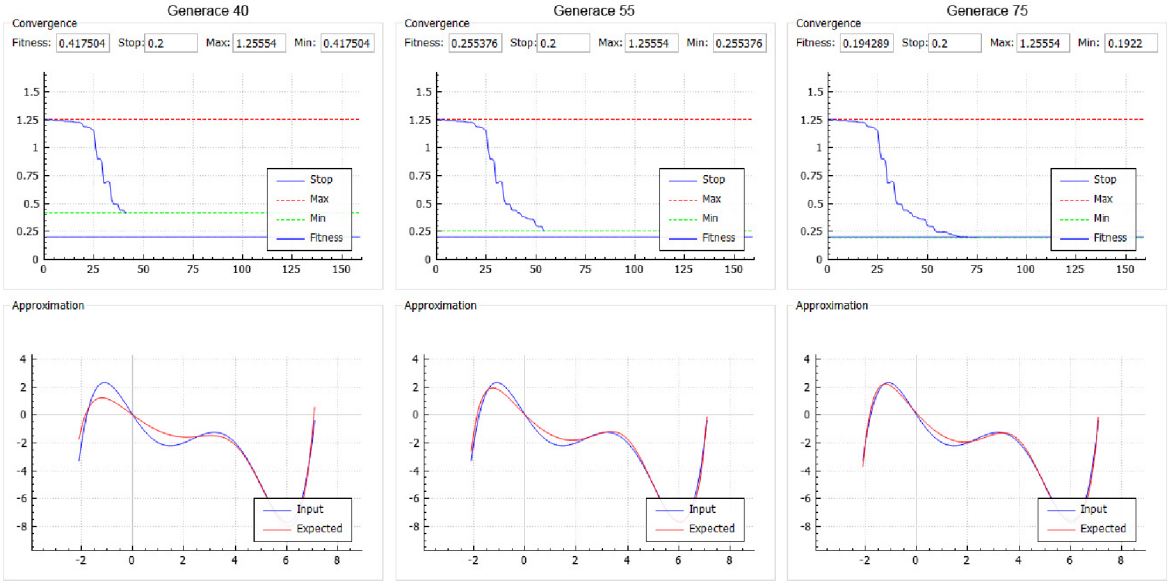
C.1.1 Příklad 1

Trénovací množina generována polynomem $\frac{1}{56}(x^5 - 12x^4 + 35x^3 + 20x^2 - 156x)$.

Cíl:	Nalezení programu, který podle zadané trénovací množiny generuje ze vstupní proměnné x očekávané výstupy y.
Množina terminálů:	T = $\{x$, náhodné konstanty $\langle -5, 5\rangle\}$
Množina funkcí:	F = $\{+, -, *\}$
Fitness hodnota:	Suma absolutních hodnot rozdílu (výstupy jedince/programu a odpovídající očekávané výstupy y)
Selekce:	Turnajová selekce (selekční tlak = 5)
Inicializace:	Metoda Ramped Half-and-Half (hloubka 2-3, 50% terminálů jsou konstanty)
Parametry:	Populace o velikosti $M = 4000$, 80% křížení podstromu, 10% reprodukce, 10% mutace podstromu, není omezena velikost stromu
Parametry křížení:	Křížení nejvyššího podstromu 25%, křížení náhodného podstromu 75%
Parametry mutace:	Mutace terminálů 40%, mutace funkcí 40%, mutace podstromu 20%, generování podstromu metodou Ramped Half-and-Half (hloubka 2-3, 50% terminálů jsou konstanty)
Ukončení:	Nalezení jedince, jehož suma absolutních odchylek je menší než 0.2

Tabulka C.1: Definice problému polynom pro GP.





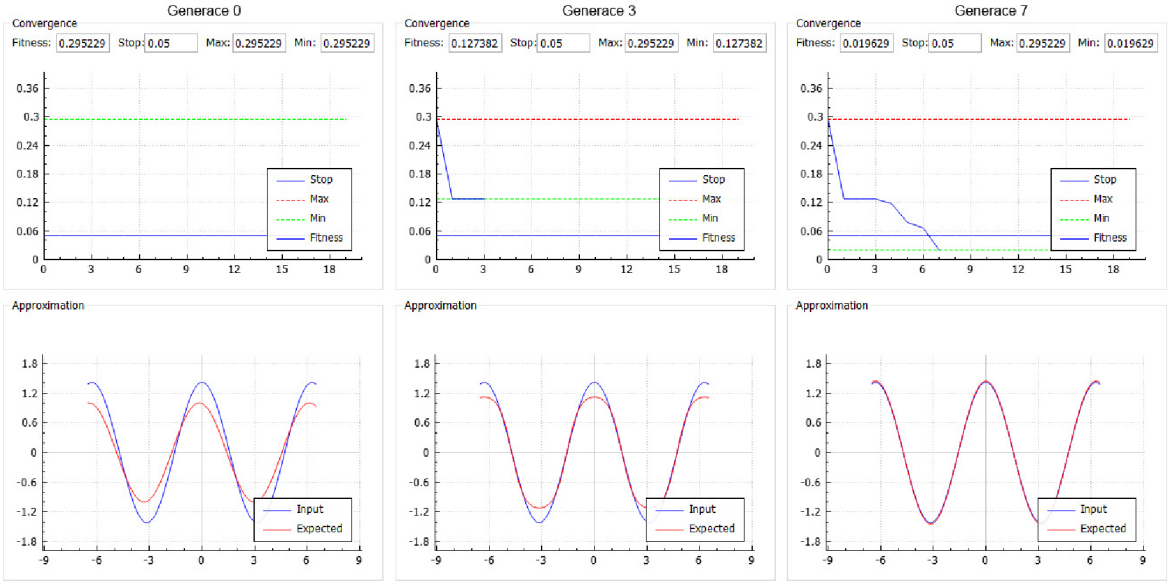
Obrázek C.1: Grafy konvergence a aproximace pro zvolené generace.

C.1.2 Příklad 2

Trénovací množina generována goniometrickými funkcemi $sin(\frac{\pi}{4}+x)+cos(\frac{\pi}{4}+x)$. Nejlepší řešení nalezeno v generaci 7 s reprezentací $cos(x) * 1.41359$.

Cíl:	Nalezení programu, který podle zadané trénovací množiny generuje ze vstupní proměnné x očekávané výstupy y.
Množina terminálů:	T = $\{x$, náhodné konstanty $\langle -5, 5\rangle\}$
Množina funkcí:	F = $\{+, *, sin, cos\}$
Fitness hodnota:	Suma absolutních hodnot rozdílu (výstupy jedince/programu a odpovídající očekávané výstupy y)
Selekce:	Turnajová selekce (selekční tlak = 8)
Inicializace:	Metoda Ramped Half-and-Half (hloubka 2-3, 50% terminálů jsou konstanty)
Parametry:	Populace o velikosti $M = 500$, 80% křížení podstromu, 10% reprodukce, 10% mutace podstromu, není omezena velikost stromu
Parametry křížení:	Křížení nejvyššího podstromu 50%, křížení náhodného podstromu 50%
Parametry mutace:	Mutace terminálů 45%, mutace funkcí 45%, mutace podstromu 10%, generování podstromu metodou Ramped Half-and-Half (hloubka 2-3, 50% terminálů jsou konstanty)
Ukončení:	Nalezení jedince, jehož suma absolutních odchylek je menší než 0.05

Tabulka C.2: Definice problému goniometrické funkce pro GP.



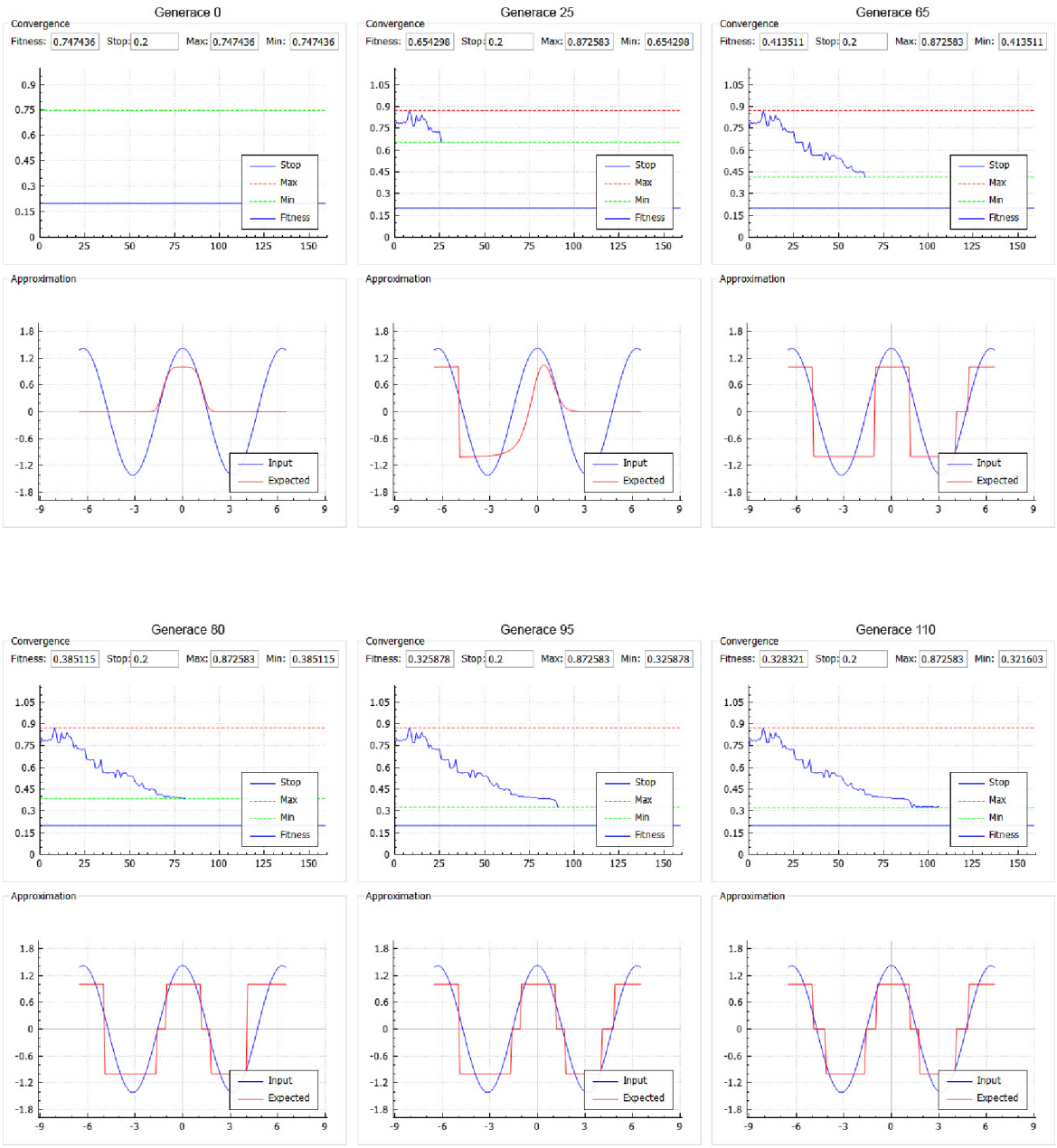
Obrázek C.2: Grafy konvergence a aproximace pro zvolené generace.

C.1.3 Příklad 3

Trénovací množina generována goniometrickými funkcemi $sin(\frac{\pi}{4}+x)+cos(\frac{\pi}{4}+x)$. Množina funkcí omezena pouze na $F = \{+, -, \texttt{pow}\}$.

Cíl:	Nalezení programu, který podle zadané trénovací množiny generuje ze vstupní proměnné x očekávané výstupy y.
Množina terminálů:	T = $\{x$, náhodné konstanty $\langle -5, 5\rangle\}$
Množina funkcí:	F = $\{+, -, pow\}$
Fitness hodnota:	Suma absolutních hodnot rozdílu (výstupy jedince/programu a odpovídající očekávané výstupy y)
Selekce:	Turnajová selekce (selekční tlak = 3)
Inicializace:	Metoda Ramped Half-and-Half (hloubka 2-3, 50% terminálů jsou konstanty)
Parametry:	Populace o velikosti $M = 4000$, 80% křížení podstromu, 10% reprodukce, 10% mutace podstromu, není omezena velikost stromu
Parametry křížení:	Křížení nejvyššího podstromu 50%, křížení náhodného podstromu 50%
Parametry mutace:	Mutace terminálů 45%, mutace funkcí 45%, mutace podstromu 10%, generování podstromu metodou Ramped Half-and-Half (hloubka 2-3, 50% terminálů jsou konstanty)
Ukončení:	Nalezení jedince, jehož suma absolutních odchylek je menší než 0.2

Tabulka C.3: Definice problému goniometrické funkce 2 pro GP.



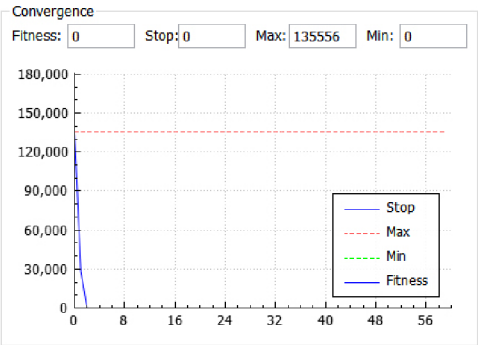
Obrázek C.3: Grafy konvergence a aproximace pro zvolené generace.

C.1.4 Příklad 4

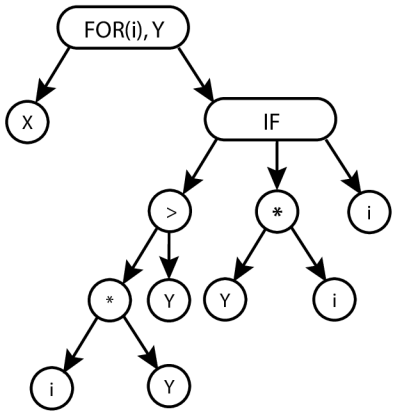
Trénovací množina generována funkcí `fact(`x`)`. Příklad testuje aplikaci uzlů `if` a `for`.

Cíl:	Nalezení programu, který podle zadané trénovací množiny generuje ze vstupní proměnné x očekávané výstupy y.
Množina terminálů:	T = {x, náhodné konstanty $\{0, 1, 2\}$}
Množina funkcí:	F = {$*$, `IF`, `FOR` max 16 opakování}
Fitness hodnota:	Suma absolutních hodnot rozdílu (výstupy jedince/programu a odpovídající očekávané výstupy y)
Selekce:	Turnajová selekce (selekční tlak = 3)
Inicializace:	Metoda Ramped Half-and-Half (hloubka 2-3, 60% terminálů jsou konstanty)
Parametry:	Velikosti populace M = 8000, 50% křížení, 50% reprodukce, 0% mutace, neomezená velikost stromu
Parametry křížení:	Křížení nejvyššího podstromu 50%, křížení náhodného podstromu 50%
Ukončení:	Suma absolutních odchylek menší než 0.0001

Tabulka C.4: Definice problému faktoriál pro GP.

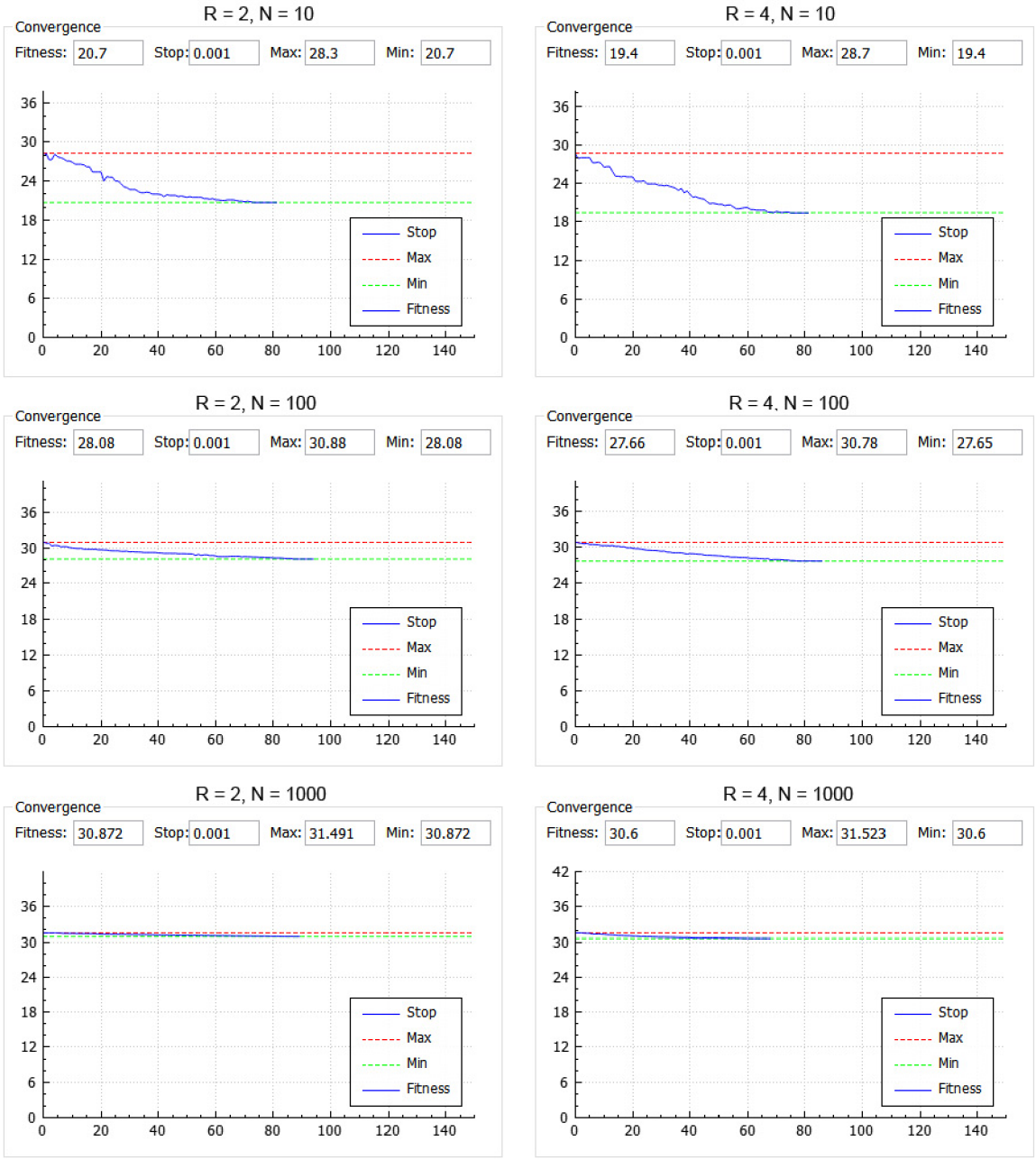


Obrázek C.4: Graf konvergence nalezeného řešení ve 2. generaci.

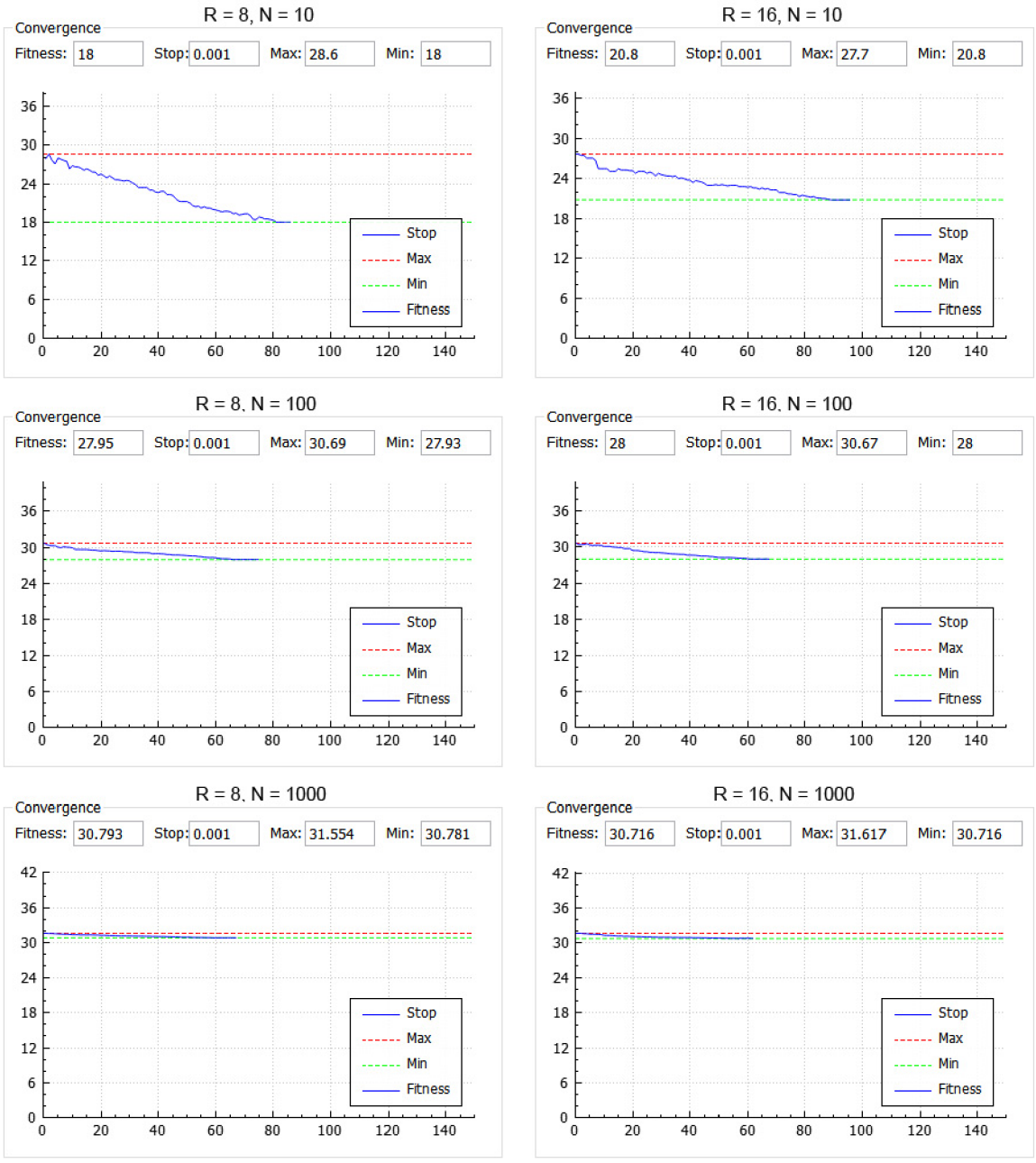


Obrázek C.5: Nalezený syntaktický strom evolvovaného jedince.

C.2 Průběh konvergence nejlepšího řešení



Obrázek C.6: Průběh konvergence fitness hodnoty pro $R \in \{2, 4\}$ a $N \in \{10, 100, 100\}$.



Obrázek C.7: Průběh konvergence fitness hodnoty pro $R \in \{8, 16\}$ a $N \in \{10, 100, 100\}$.

Příloha D

Obsah CD

- Aplikace - source/app.zip
- Příklady trénovacích množin pro GP - source/examples
- Informace pro zprovoznění aplikace - conf/Readme.txt
- Technická zpráva - dp.pdf
- Technická zpráva ve zdrojových souborech (*.tex) - dp.zip