evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Demonstrátor KEYMAKER

Název: Využití evolučních algoritmů pro hledání S-boxů

Vypracoval: Bc. Bedřich Hovorka

Datum: 25. 11. 2010

evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost

Tato práce vznikla za podpory projektu MŠMT CZ.1.07/2.3.00/09.0067 TeamIT – Budování konkurenceschopných výzkumných týmů pro IT.

Obsah

Obsah 1

1 Úvod 3

2 Úvod do kryptografie 4
2.1 Symetrická kryptografie . . . 6
2.2 Vnitřní struktura šifrujícího bloku . . . 8
2.3 Dosahovaní bezpečnosti . . . 10
2.4 Lineární kryptoanalýza . . . 11
2.5 Diferenciální kryptoanalýza . . . 12

3 Problematika S-boxů 13
3.1 Definice s-boxů . . . 13
3.2 Kritéria bezpečnosti s-boxů . . . 14
3.2.1 SAC . . . 14
3.2.2 Bent funkce . . . 15
3.2.3 Největší odchylka z lineárních výrazů . . . 16
3.2.4 Největší pravděpodobnost z diferenciálů . . . 17
3.2.5 Faktor větvení . . . 17
3.2.6 Řád algebraického polynomu . . . 17
3.2.7 Bijektivita . . . 18
3.2.8 Další kritéria . . . 18
3.3 Příklady dalšího užití s-boxů . . . 19
3.3.1 S-box v DES . . . 19
3.3.2 S-box v AES . . . 20

4 Evoluční algoritmy 21
4.1 Genetický algoritmus . . . 21
4.2 Genetické programování . . . 22
4.2.1 Kartézské genetické programování . . . 23
4.3 EDA . . . 24
4.3.1 Činnost EDA algoritmu . . . 24
4.3.2 UMDA . . . 25
4.3.3 BMDA . . . 25
4.4 Multikriteriální optimalizace . . . 26
4.4.1 Pareto optimum . . . 26
4.4.2 Algoritmus VEGA . . . 27
4.4.3 Algoritmus SPEA . . . 27

4.5 Další metody k hledání s-boxů . . . 29
4.6 Hledání s-boxů vs. symbolická regrese . . . 30

5 Návrh programu pro experimenty **31**
5.1 Shrnutí technických a funkčních požadavků . . . 31
5.2 Prototypový návrh aplikace . . . 32
5.3 Různé typy reprezentace s-boxu . . . 32
5.3.1 Seznam výstupů . . . 32
5.3.2 Binární reprezentace . . . 33
5.3.3 Acyklický graf hradel . . . 33
5.3.4 Posloupnost trojinstrukcí platformy i686 . . . 33
5.4 Použité jazyky a platforma . . . 35

6 Popis implementace **37**
6.1 Implementace evolučních algoritmů . . . 37
6.2 Implementace reprezentací s-boxů . . . 39
6.3 Naimplementovaná kritéria a jejich vyhodnocení . . . 40
6.4 Rozhraní komunikace z Pythonem . . . 41
6.5 Diskuze možností paralelizace výpočtů . . . 41

7 Provedené experimenty a zhodnocení výsledku **43**
7.1 Stanovení pravděpodobností mutace a křížení pro jednotlivá kritéria . . . 43
7.2 Porovnání výsledků EDA algoritmů s Genetickým algoritmem . . . 46
7.3 Parametry paralelního náhodného prohledávání . . . 48
7.4 Parametry multikriteriálního vyhledávání . . . 51
7.5 Hledání složitějších s-boxů . . . 53
7.6 Dvoufázové hledání . . . 54
7.7 Celkové zhodnocení výsledku . . . 54

8 Závěr **55**

Seznam použitých zkratek **58**

A Chromozóm v Kartézském genetickém programování **61**

B Přehled příkazů pro skriptování experimentů **62**

C Podrobné tabulky a grafy **65**

Kapitola 1

Úvod

Rostoucí výkon počítačů a nárůst jejich použití širokou masou uživatelů v celosvětové počítačové síti si žádají vyšší nutnost zaobírat se bezpečností těchto uživatelů. Neboť součástí této sítě nejsou jen samí lidé a organizace s dobrými úmysly, ale i tací, kteří se snaží porušit právo na soukromí odposlechnutím důvěrných informací jiné osoby a využít je ve svůj prospěch či tu osobu zkompromitovat jiným způsobem, například se za ní vydávat. Proto je důležité těmto záškodníkům tyto činnosti prakticky znemožnit převodem důvěrné informace do podoby, kterou budou schopny přečíst jen vybrané osoby a nebude schopna přečíst žádná nežádoucí osoba – důvěrnost. Dále musí být součástí tohoto převodu ověření identity původce přenášené informace – autentizace. Algoritmy, které zajišťují důvěrnost a autentizaci přenášených dat, se zabývá kryptografie. Těmto algoritmům se též říká šifra. Díky čím dál tím jejich větším používáním a objevováním novějších druhů kryptoanalýzy životnost šifer klesá. Poučení z historie šifer říká, že šifra byla neprolomitelná do té doby, než někdo objevil její slabinu. Proto musela být vzápětí vždy vytvořena šifra nová. V dnešní době probíhá neustálý výzkum a vývoj nových šifer, které by měli být odolnější vůči stále přibývajícím novějším druhům útoků.

Tato práce se zabývá částí šifrovacího algoritmu zvanou substituční box (zkráceně s-box) a jeho vývojem. K jejich konstrukci se využívají jak čistě matematické přístupy s využitím teorie Galois polí, tak i různé stochastické způsoby návrhu, ke kterým se řadí například evoluce, která je použita v této práci. Evoluční výpočetní techniky jsou, jak již název napovídá, inspirovány evoluční teorií z biologie. Cílem práce je použít některé vlastnosti substitučních boxů jako objektivní funkce k jejich evolučnímu vývoji. Přestože již bylo mnoho vlastností studováno, relativně málo práce bylo uskutečněno při hledání s-boxů především v multikriteriálním genetickém prohledávání.

Nejprve však bylo nutné navrhnout a naimplementovat kolekci algoritmů, která tyto vlastnosti umožní experimentálně zjistit. Tato implementace musí především umožnit rychle zadávat různé evoluční experimenty a, pokud možno, provést je co nejrychleji.

Tato práce vznikla jako samostatné dílo, které nenavazuje na žádný školní projekt.

Kapitola 2

Úvod do kryptografie

Kryptografie [8, 5] se zabývá především utajováním informace a ověřováním jejího původu tj. převodem dat do podoby, která je nepoužitelná nežádoucím osobám, které budou dále v textu pro zjednodušení souhrnně nazvány slovem „útočník“, přestože nemusí útočit přímo a může jich být více. Žádoucí osoby mají narozdíl od těch nežádoucích speciální znalost. Tou znalostí byl v minulosti i postup zašifrování, od toho se v dnešní době upouští a tou znalostí je pouze klíč. Předpokládá se, že algoritmus zná i útočník. Dále může i k neutajované informaci přidat dodatečnou informaci o jejím původu, tzv. elektronický podpis či časové razítko, a to takovým způsobem, že je prakticky nemožné ověřit tímto podpisem jinou informaci než v tu, která identicky odpovídá té při podpisu. Takový převod by měl řešit hlavní problémy v bezpečnosti:

důvěrnost útočník nesmí získat žádnou užitečnou informaci ze zašifrovaného komunikačního kanálu

autentizaci ověření identity odesílatele zprávy

integritu útočník nemůže podvrhnout data či jejich část a vydávat je příjemci za taková, jakoby pocházela od určitého odesílatele – tj. i změnit či upravit data pocházející od pravého odesílatele

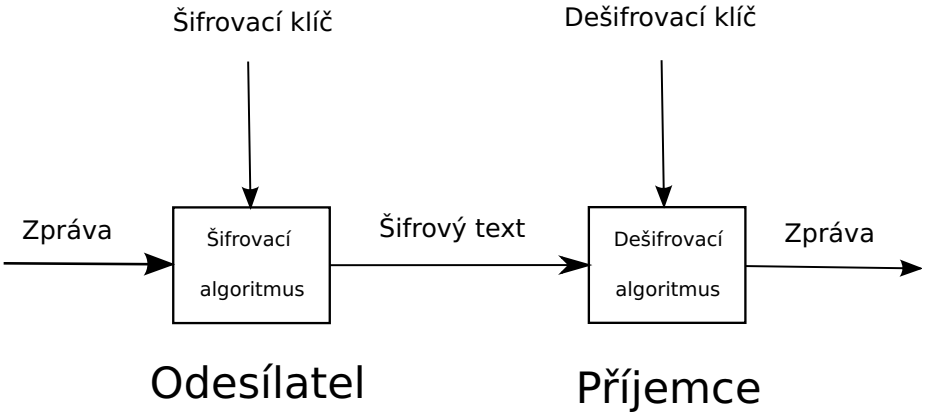
nepopiratelnost odesílatel nemůže popřít u jím vytvořené a podepsané zprávy, že ji odeslal on

Tato transformace se dá zjednodušit do matematického výrazu $C = F(K_1, P)$, kde P je prostý text (angl. „plaintext“), C je šifrový text (angl. „ciphertext“) a K je klíč, který představuje možnost nastavit, jakým způsobem šifra převede text. K opačnému procesu zvanému dešifrování $P = F^{-1}(K_2, C)$ nemusí být stejný klíč. Proces je též znázorněn na obrázku 2.1. Čím delší klíč, tím je složitější provést útok hrubou silou k jeho získání. Funkce F se dále v textu nazývá šifra či také kryptosystém.

Moderní kryptografie se dělí na dvě hlavní kategorie (dle počtu klíčů):

symetrická Za symetrický kryptosystém je označován takový, když lze dešifrovací klíč snadno odvodit z klíče šifrovacího. Většinou existuje jen jeden klíč tzv. „tajný klíč“, který se používá jak pro šifrování, tak pro dešifrování. Problémem při používání tohoto systému je distribuce klíče.

asymetrická Za asymetrický systém je považován takový, kdy dešifrovací klíč není možné prakticky odvodit z klíče šifrovacího. Většinou existuje pár klíčů, jenž vygeneruje



Obrázek 2.1: Znázornění procesu utajování informace

příjemce, jeden klíčů v páru je „veřejný“ a v procesu utajování informace jej odesílatel použije k zašifrování. Druhý klíč je „soukromý“ (kromě příjemce jej nikdo nesmí znát), kterým příjemce přijatou zprávu dešifruje. Veřejný klíč může díky tomu, že z něho nelze snadno odvodit soukromý, být kdekoliv vystaven. Distribuce klíče je vyřešena za cenu výpočetní náročnosti šifrování.

Oba přístupy se v praxi většinou kombinují tak, že odesílatel nejprve vygeneruje náhodný tajný klíč K_{rand}, ten zašifruje asymetrickou šifrou veřejným klíčem příjemce

$$C_1 = F_{asym}(K_{public_{recv}}, K_{rand})$$

Vlastní zprávu P pak zašifruje symetricky klíčem K_{rand}: $C_2 = F_{sym}(K_{rand}, P)$, protože zašifrovat symetricky je o dost rychlejší. Nakonec příjemci odešle data ve tvaru C_1, C_2 (zjednodušeně vyjádřeno). Blokem je dále myšlena jakákoliv posloupnost po sobě jdoucích bitů. Šifrujícím blokem „krabička“, která převede blok prostého textu na blok šifrového. Jednotlivé bloky budou dále znázorněny jako signály, které vyznačují propojení mezi jednotlivými částmi algoritmu a tok informace.

V předešlých odstavcích byl ilustrován postup při utajování informace. Pro elektronický podpis, se používá asymetrická šifra s opačnými klíči odesílatele. Při podpisu odesílatel použije svůj soukromý klíč a příjemce jej dešifruje pomocí veřejného klíče odesílatele a tím u zprávy ověří původ. Odesílatel musí mít dva různé páry klíčů, jeden pár pro šifrování a druhý pár pro podpis, aby mu útočník nemohl podstrčit k podpisu jinou informaci, která je určená pro odesílatele a kterou by tím pádem dešifroval. A navíc, protože je asymetrické šifrování výpočetně náročná operace, je při podpisu zprávy nejprve vypočítán otisk (tzv. „hash“), a ten je poté asymetricky zašifrován soukromým klíčem odesílatele a připojen ke zprávě. Příjemce zprávu ověří tak, že nejprve spočítá „hash“ přijaté vlastní zprávy a zároveň pomocí veřejného klíče odesílatele dešifruje „hash“ přijatý, poté oba „hashe“ porovná. Hashovací algoritmy jsou další skupinou kryptografických algoritmů. Hashovací algoritmus tedy provádí transformaci libovolně dlouhé zprávy do pevně zvolené délky tzv. „hashe“. Tento „hash“ by měl být k dané zprávě unikátní. Dvě různé zprávy, byť by se lišily v jednom bitu, nesmí mít stejný „hash“.

V tabulce 2.1 je uvedeno několik nejpoužívanějších kryptografických algoritmů a jejich nejpoužívanějších délek klíčů. U asymetrických algoritmů se délka klíče během let zvyšuje mnohem více než symetrických a hashovacích. Vyvinout novou symetrickou šifru je mnohem

Druh	Zkratka	počet bitů klíče
symetrický	AES	128, 192, 256
	DES	56, 168
	RC4	40 až 256
asymetrický	RSA	..., 1024, 2048, 4096, ...
	DSA	1024, 2048, 3072
hashovací		
	MD5	128
	SHA1	160
	SHA2	224, 256, 384, 512

Tabulka 2.1: Příklady nejznámějších kryptografických algoritmů

jednodušší než vyvinout asymetrickou. U asymetrické je nutné nejprve najít matematický princip, proto se jejich bezpečnost více odráží ve velikosti klíče. Symetrické šifry s klíčem menším než 80 bitů jsou považovány za slabé. U asymetrických je ta hranice již na 1024 bitech. Hashovací algoritmy MD5 a SHA1 jsou též považovány za slabé. 168-bitový klíč u DESu se nazývá TripleDES, tento klíč se rozdělí na tři 56-bitové a při šifrování se projde třemi fázemi DESu $C = F(K_3, (F^{-1}(K_2(F(K_1, P)))$, prostřední je dešifrování. Z hashovacích jsou uvedeny pouze bezklíčové. NIST[1] vypsal soutěž na nový standard SHA3, která se nyní nachází ve 2.kole, mezi postupujícími kandidáty jsou například šifry CubeHash či Luffa([2]).

2.1 Symetrická kryptografie

Symetrické šifry se dělí na blokové a proudové. Proudové šifry berou zprávu po bajtech, případně bitech či znacích. Bloková šifra bere zprávu po větších blocích (64 až 256 bitů). Vetší velikost bloku zabraňuje statistickým a slovníkovým útokům. Pro n-bitový blok je 2^n možných vstupních i výstupních kombinací a existuje $2^n!$ možných zobrazení. Příkladem blokové šifry je například DES který šifruje po 64 bitových blocích. Příkladem proudové šifry je RC4. Proudovou šifrou se může být i bloková, pokud je zapojena v určitém režimu (viz dále).

Režimy blokových šifer

Rozdělené bloky zprávy mohou být jakoukoliv blokovou symetrickou šifrou zpracovány těmito následujícími způsoby. Pokud má útočník k dispozici odpovídající dvojice bloků prostého a šifrového textu, mohou být některé režimy slabší. U některých režimů přibývá ke klíči ještě inicializační vektor, který je použit pro inicializaci šifrování prvního bloku, jako další znalost nutná k dešifrování.

Electronic Codebook (ECB) Nejjednodušší, každý i-tý blok zprávy je přímo vstupem šifrovacího bloku a výstup je přímo i-tým blokem šifrového textu. Kryptoanalýza tohoto režimu je jednodušší, protože útočník v šifrovém textu může znovu najít tentýž blok, pro který již má odpovídající prostý text ve slovníku, který si průběžně vytváří.

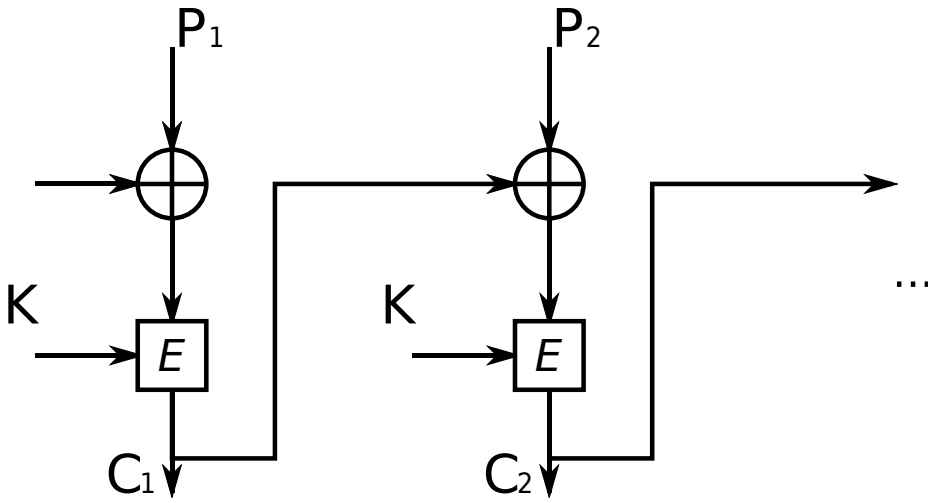
[1] Národní institut pro standardizaci a technologie USA

Tím je porušena i integrita dat, neboť útočník může podstrčit jiný blok šifrového textu.

Cipher Block Chaining (CBC) Bloky jsou zřetězeny. Na rozdíl od ECB je každý blok xorován s předchozím výstupním blokem předtím, než bude použit jako vstup šifrovacího bloku. Blok prvního je xorován s inicializačním vektorem. Znázorněn je na obrázku 2.2.

Cipher Feedback (CFB) Narozdíl od CBC režimu je blok prostého textu xorován až s výstupem bloku šifry a výsledek také přiveden na vstup následujícího šifrujícího bloku. Vstupem prvního bloku je inicializační vektor.

Output Feedback (OFB) Narozdíl do CFB režimu je na vstup následujícího bloku poslán výstup předchozího ještě před xorováním s plaintextem.



Obrázek 2.2: CBC mód [23]

Mimo toho ještě existuje tzv. „counter mód“, kdy je bloková šifra použita k generování pseudonáhodné posloupnosti, tak že jsou na vstup blokové šifry přiváděny hodnoty čítače. Výhodou je, že každý blok může být šifrován nezávisle na ostatních. Prostý text se xoruje s výstupem blokové šifry. Hodnota čítače se samozřejmě nesmí opakovat pro stejný klíč. I u režimu OFB se dá hovořit jako o proudové šifře a o režimu CFB o proudové samosynchrozující.

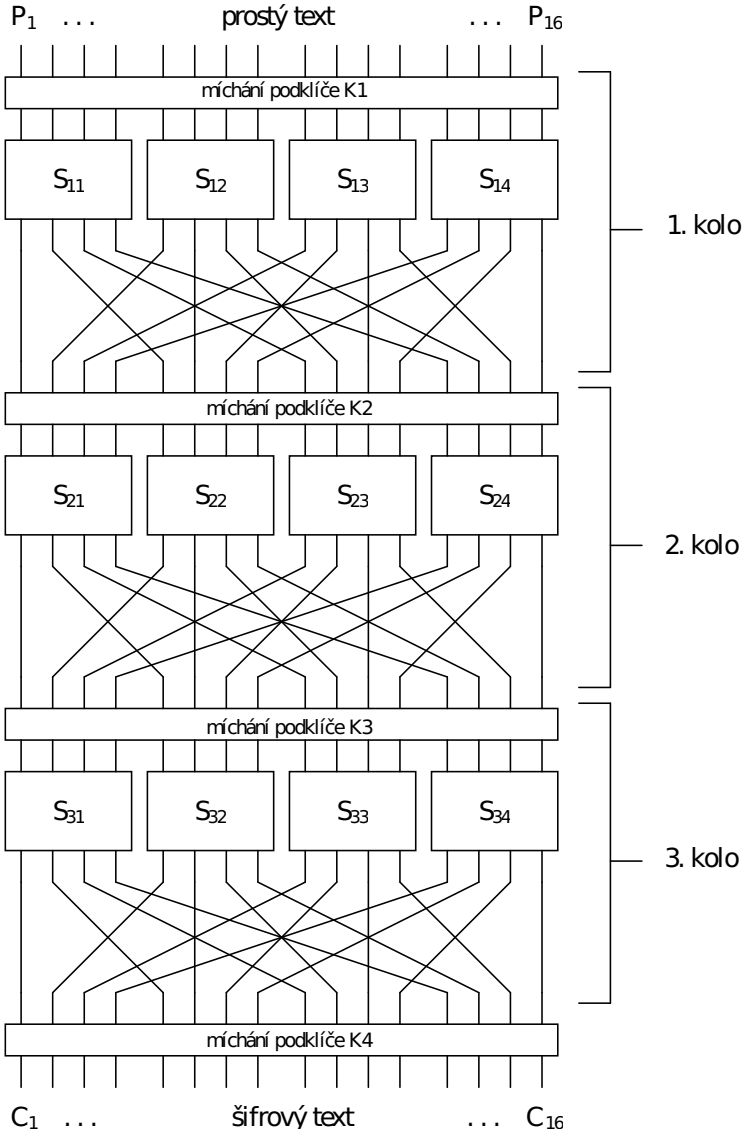
2.2 Vnitřní struktura šifrujícího bloku

Zašifrování jednoho bloku se rozděluje na různé fáze podle druhu algoritmu. Mnoho moderních šifer je založeno na Feistelově substituční/permutační síti, tj. jsou kombinací těchto transformací, probíhajících v několika kolech za sebou:

substituce Určitý blok kódu je nahrazen jiným z množiny možných kombinací

transpozice/permutace Pracuje s několika bloky, těm je změněno pořadí

míchaní se subklíčem z klíče jsou vygenerovány subklíče pro jednotlivá kola, ty jsou pak „přimíchány“, nejčastěji XORem

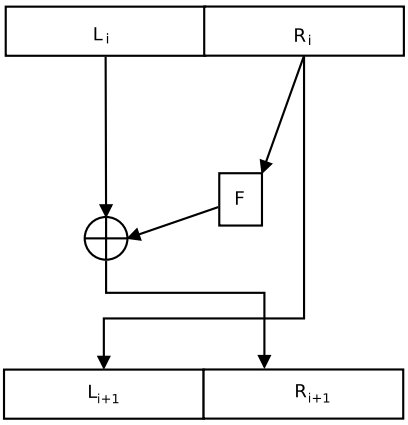


Obrázek 2.3: Jednoduchá substituční/permutační síť znázorňující šifrovací algoritmus [7]

Obrázek 2.3 znázorňuje jednoduchou vícekolovou šifru. Substituce je většinou abstrahována pomocí tzv. „substitučních boxů“, zkráceně s-boxů (prvků S_{kn} z obrázku 2.3). S-boxy jsou

znázorněny jako jako „krabičky“ S_{kn}, do kterých vstupují, a ze kterých vystupují signály. Permutace je znázorněna pomocí signálů. V každém kole a na každé pozici jsou užity s-boxy s různými zobrazeními. Na uvedeném obrázku se jedná o s-boxy typu 4×4. V této práci nás bude zajímat jejich vnitřek a výstupy. Principy s-boxů jsou nejlépe popisovány na symetrických šifrách, ač se nevylučuje jejich použití i v jiných druzích algoritmů. Jejich problematika lze ukázat nejlépe právě na symetrických šifrách, protože sami mají charakter symetrické blokové šifry, i když v menší podobě.

Tato jednoduchá síť sama o sobě však nezaručuje možnost jak šifrování tak dešifrování pomocí té samé struktury. Aby proces mohl být reverzibilní, či-li aby bylo možné šifrovat/dešifrovat ve více kolech pomocí té samé struktury, navrhl Feistel [24] strukturu na obrázku 2.4.



Obrázek 2.4: Struktura jednoho kola Feistelovy šifry [24]

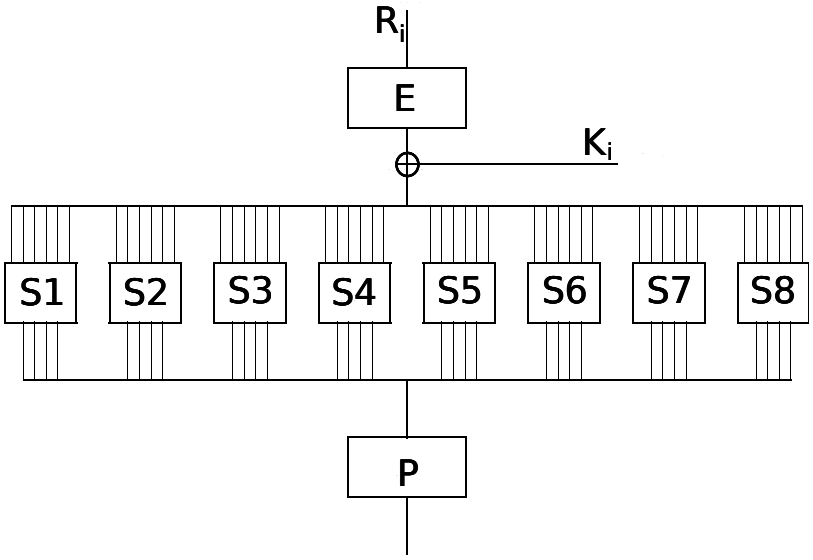
Vstupní text se rozdělí na dvě poloviny. V každém kole je pravá polovina vstupem jak šifrující funkce F, tak levou polovinou výstupu kola. Levá polovina vstupu je xornuta s výstupem šifrující funkce a výsledek je pravou polovinou výstupu. Šifrující funkce je nastavena příslušným subklíčem. Pro dešifrování se použijí subklíče v opačném pořadí. Na funkci F se přenáší odpovědnost za bezpečnost, o reverzibilitu se postará Feistel struktura. Když pro šifrování v každém kole platí (výpočet kola následujícího):

$$L_{i+1} = R_i \tag{2.1}$$
$$R_{i+1} = L_i \oplus F(K_i, R_i) \tag{2.2}$$

pro dešifrování v tom případě platí v každém kole :

$$R_i = L_{i+1} \tag{2.3}$$
$$L_i = R_{i+1} \oplus F(K_{i+1}, R_{i+1}) \tag{2.4}$$

L_i resp. R_i resp. K_i je levý vstup resp. pravý vstup resp. subklíč kola. Ve funkci F se může provádět substituce, permutace a transpozice. V ní může být obsaženo několik s-boxů. Tuto strukturu přejalo mnoho šifer, např. DES či Twofish. Liší se především ve funkci F, generování subklíčů a velikosti zpracovávaného bloku. Na obrázku 2.5 je znázorněna funkce v šifře DES. E je expanze z 32 bitů na 48 bitů. Pak následuje míchaní se 48 bitovým subklíčem. Výsledek je rozdělen mezi 8 s-boxů typu 6×4. Jejich hlavním úkolem je nelineární



Obrázek 2.5: Struktura Feistelovy funkce v DESu [8]

zobrazení jejich vstupu na výstup (vysvětleno dále). Výstupy s-boxů jsou sloučeny do 32 bitů a permutovány funkcí P.

V podkapitole 3.3 (po vysvětlení následující pojmů) jsou některé s-boxy popsány podrobněji.

2.3 Dosahovaní bezpečnosti

Při návrhu kryptosystému, by měly být položeny tyto otázky:

- Jak může kryptolog dokázat bezpečnost kryptosystému?
- Jaké statistické vlastnosti má struktura kryptosystému?
- Jak zjistí tyto vlastnosti efektivně?

Na první otázku nelze odpovědět plně uspokojivě, protože s rostoucím výkonem počítačů a úsilím kryptoanalytiků není vyloučeno objevení slabiny šifry a využití tohoto objevu k útoku. Proto lze šifru navrhovat s cílem odolnosti proti všem známým útokům, přičemž není prakticky možné celou šifru úplně zvalidovat. Lze měřit sílu částí šifer pomocí různých kritérií.

Odpovědí na druhou otázku je, že [8] šifra nesmí mít malé množství možných párů prostých textů a příslušných šifrových pro různé délky z celého textu. Jakákoliv množina výstupních bitů by neměla záviset na jakékoliv vlastní podmnožině množiny všech vstupních bitů pro pevný klíč. Jakákoliv množina výstupních bitů by neměla záviset na jakékoliv vlastní podmnožině množiny všech bitů klíče pro pevný vstupní text. Z použitelných statistických testů lze uvést např. χ^2-test či Kolmogrov-Smirnovův test. Avšak není prakticky možné projít všechny možné klíče, natož všechny možné texty. Proto se vygeneruje několik náhodných klíčů a náhodných textů. Výstup by měl mít charakteristiky náhodné posloupnosti. K ověření se využívá např. „frekvenční analýza“, jež ověřuje výskyty hodnot na

různých pozicích a jejich závislosti. Je jasné že ideálně náhodný výstup nelze prakticky vytvořit.

Bezpečným vlastnostem musí nejprve odpovídat jednotlivé prvky šifry. Různé prvky jsou odpovědné za různé vlastnosti celé šifry. Při konstrukci jednotlivých prvků může být díky malému množství vstupů a výstupů nenáročné prověřit všechny tyto vlastnosti pro všechny možné kombinace. Možnostmi jak ověřit odolnost prvku je i provedení známého útoku, který těží ze slabin způsobených špatnými charakteristikami prvku šifry. Dva nejdůležitější útoky na šifrovací algoritmy, jež se používají mj. i k ověření náhodnosti výstupu s-boxů byly vytvořeny k útoku na algoritmus DES.

2.4 Lineární kryptoanalýza

Lineární kryptoanalýza [7] zkouší těžit z vychýlené pravděpodobnosti výskytu lineárních výrazů zahrnujících jak bity prostého textu, šifrového textu a (pod)klíče (to v obecném případě, v této práci dále neuvažováno). Jedná se o tzv. „known plaintext attack“: to znamená, že se předpokládá, že útočník má k dispozici sadu prostých textů a k nim příslušných šifrových. Přesto nemá útočník možnost vybrat, který prostý text (a příslušný šifrový) je dostupný. V mnoha aplikacích a scénářích se považuje za dostatečné, že ta sada textů je náhodná. Jinými slovy shrnuto se lineární kryptoanalýza snaží z náhodné sady prostých textů a příslušných šifrových textů aproximovat šifrovací algoritmus různými lineárními výrazy a nej(ne)pravděpodobnější z nich použít k postupnému získání klíče.

Linearita je vztažena vůči mod-2 bitové operaci (např., exkluzivní součet označený pomocí „$\oplus$“). A lineární výraz je ve formě:

$$a_1 \cdot X_{i_1} \oplus a_2 \cdot X_{i_2} \oplus \cdots \oplus a_u \cdot X_{i_u} \oplus b_1 \cdot Y_{j_1} \oplus a_2 \cdot Y_{j_2} \oplus \cdots \oplus a_v \cdot Y_{j_v} = 0 \tag{2.5}$$

kde X_i reprezentuje i-tý vstupní prostý text $X_i = [X_{i_1}, X_{i_2}, \dots]$ a Y_j reprezentuje j-tý výstupní šifrový text $Y_i = [Y_{j_1}, Y_{j_2}, \dots]$.[2] Tato rovnice reprezentuje „sumu“ exkluzivních součtů vstupních a výstupních bitů, které jsou vybrány[3] bitovými vektory a (pro vstupní) a b (pro výstupní).

Konceptem lineární kryptoanalýzy je určit výrazy ve formě rovnice 2.5, které mají vysokou nebo naopak nízkou pravděpodobnost. Pokud má k takovým pravděpodobnostem tendenci, znamená to evidentně slabou schopnost šifry produkovat výstup s charakteristikami náhodně vygenerovaného. Mějme náhodně vygenerované bity $u + v$ a pokud je použijeme v rovnici 2.5, pravděpodobnost p_L by měla být přesně $\frac{1}{2}$. Odchylku od pravděpodobnosti $p_L = \frac{1}{2}$ nazýváme „bias“ značeno ϵ_i, kde i se vztahuje k výrazu. Čím je vyšší velikost této odchylky, $|\epsilon_i = p_L - \frac{1}{2}|$, tím snazší je šifru rozlousknout.

Poznamenejme, že $p_L = 1$ znamená, že lineární výraz 2.5 je přesnou reprezentací šifry a tím pádem je šifra katastrofálně slabá. Pokud $p_L = 0$, potom výraz reprezentuje afinní vztah, také znak veliké slabosti. Pro aritmetiku mod-2 je afinní funkce jednoduše komplementem lineární. Obě lineární resp. afinní aproximace, indikované pravděpodobnostmi $p_L > \frac{1}{2}$ resp. $p_L < \frac{1}{2}$, mají v lineární kryptoanalýze stejnou váhu a dále se v tomto textu budou pojmem „lineární“ myšleny oba dva vztahy.

[2] Vstupní i výstupní text jsou definovány jako vektory bitů.

[3] „Zapnuty“ pomocí bitové konjunkce označené jako „$\cdot$“

2.5 Diferenciální kryptoanalýza

Diferenciální kryptoanalýza [7] využívá vysokou pravděpodobnost společného výskytu rozdílů mezi dvěma šifrovými texty na výstupu s rozdíly mezi dvěma prostými texty na vstupu. Například mějme systém se vstupem $X = [X_1 X_2 \dots X_n]$ a výstupem $Y = [Y_1 Y_2 \dots Y_n]$. Nechť dva vstupy systému jsou X' a X'' s odpovídajícími výstupy Y' a Y''. Vstupní rozdíl je dán $\Delta X = X' \oplus X''$ kde „$\oplus$“ reprezentuje bitový exkluzivní součet n-bitových vektorů, a odtud

$$\Delta X = [\Delta X_1 \Delta X_2 \dots \Delta X_n] \tag{2.6}$$

kde $\Delta X_i = X'_i \oplus X''_i$ kde X'_i resp. X''_i reprezentuje i-tý bit X' resp. X''. Obdobně pro výstupy.

V ideální šifře[4] je pravděpodobnost, že se vyskytuje částečný rozdíl výstupů ΔY při daném částečném rozdílu vstupů ΔX, $p_D = \frac{1}{2}^n$, kde n je počet bitů v X. Diferenciální kryptoanalýza je použitelná v případech, kde se ΔY vyskytuje společně s ΔX s velmi velkou p_D (např., o hodně větší než $\frac{1}{2}^n$). Dvojice $(\Delta X, \Delta Y)$ je nazývána jako *diferenciál*.

Diferenciální kryptoanalýza je tzv. „chosen plaintext attack“, to znamená, že si útočník může vybrat vstup a prověřit výstup v pokusu odvodit klíč. V diferenciální kryptoanalýze, útočník může vybrat dvojici vstupů X'' a X'' patřící k ΔX při znalosti, že se při ΔY s největší pravděpodobností vyskytuje ΔX.

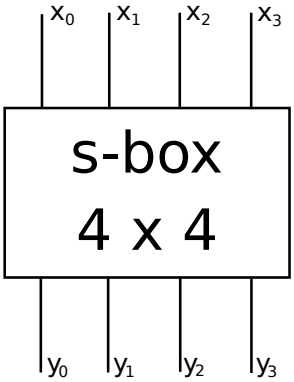
[4] tj. takové, která má na výstupu náhodné vlastnosti

Kapitola 3

Problematika S-boxů

3.1 Definice s-boxů

Substituční box (s-box) je část šifry taková, že jejím výstupem, jak již sám název název napovídá, je substituovaný vstup. Právě na s-boxy bývá přenesena většina odpovědnosti za nelineárnost a náhodnost výstupu celé šifry. Bezpečnost většiny blokových šifer (především těch na Feistelově síti) závisí na vlastnostech s-boxů použitých v jednotlivých kolech. Ty budou popsány níže. Většinou se s-box znázorňuje jako „krabička“ s jednotlivými vstupy a výstupy, kde jsou většinou znázorněny jednotlivé bity v pořadí a číslování, jak je vidět na obr 3.1.



Obrázek 3.1: Schéma s-boxu

Formálně se s-box dá definovat takto:

Definice 3.1. *[12] $n \times m$ s-box S je zobrazení $S : \{0,1\}^n \to \{0,1\}^m$. S může být reprezentován jako 2^n m-bitová čísla, označená $r_0, \ldots, r_{2^n-1}$, v tom případě $S(x) = r_x$, $0 \leq x < 2^n$ a r_i jsou řádky s-boxu. Alternativně, $S(x) = [c_{m1}(x)\, c_{m2}(x) \ldots c_0(x)]$ kde c_i je pevná Booleovská funkce $c_i : \{0,1\}^n \to \{0,1\}\, \forall i$; to jsou sloupce s-boxu.*

Velikost s-boxů se často zapisuje jako $n \times m$. Velké s-boxy [6] jsou obecně pokládány za bezpečnější než malé, ale za cenu větší složitosti realizace a validace. S-boxy s malým vstupem a velkým výstupem mají velmi dobrou lineární aproximaci. U s-boxů s dostatečně větším výstupem než vstupem je garantována nejméně jedna perfektní lineární aproximace.

3.2 Kritéria bezpečnosti s-boxů

Kritéria, jež se dají použít jako[1] heuristické či hodnotící funkce, mohou být použita k prohledávání stavového prostoru. Téměř všechny vycházejí (či mají nějaký vztah) z lineární a diferenciální kryptoanalýzy. Vycházejí z možných útoků na šifru, mohou být různě matematicky dokázány odolnosti vůči těmto útokům, jiným se spíše věří, že fungují.

3.2.1 SAC

Strict Avalanche Criterion [8] je založen na tzv. „lavinovém efektu". U s-boxů se zkoumá, kolik bitů výstupu se změní při změně jednoho bitu vstupu. Při vyzkoušení změny jednoho bitu u všech možných vstupů, by průměr změn na výstupu měl být polovina počtu výstupních bitů.

Definice 3.2 (Avalanche efekt)**.** *Funkce* $f : \{0,1\}^n \to \{0,1\}^m$ *ukazuje lavinový efekt pokud a jen tehdy:*

$$\sum_{x \in \{0,1\}^n} wt(f(x) \oplus f(x \oplus c_i^{(n)})) = m2^{n-1} \tag{3.1}$$

pro $\forall i : 1 \leq i \leq n$*, kde* $wt(x)$ *je Hammingova váha*[2] *a* $c_i^{(n)}$ *je* n*-dimenzionální vektor s* $wt(c_i^{(n)}) = 1$ *s nastavenou pozicí* i*.*

Z Avalanche efektu je definováno SAC:

Definice 3.3 (SAC, SAC 0. řádu)**.** *Funkce* $f : \{0,1\}^n \to \{0,1\}^m$ *splňuje SAC, pokud pro* $\forall i : 1 \leq i \leq n$ *platí tyto rovnice:*

$$\sum_{x \in \{0,1\}^n} f(x) \oplus f(x \oplus c_i^{(n)}) = (2^{n-1}, 2^{n-1}, \ldots, 2^{n-1}) \tag{3.2}$$

Každá funkce f *splňující SAC je nazývána „silný s-box"*

Pokud funkce splňuje SAC, každý její výstupní bit by se měl změnit s poloviční pravděpodobností, pokaždé co je změněn jeden vstupní bit. Pokud nějaký výstup závisí na pouze několika málo vstupních bitech, potom může útočník těchto vztahů využít k nalezení klíče. SAC bylo dále rozšířeno:

Definice 3.4 (SAC 1. řádu)**.** *Funkce* $f : \{0,1\}^n \to \{0,1\}^m$ *splňuje SAC 1. řádu pokud a jen tehdy:*

- f *splňuje SAC 0. řádu*
- *každá funkce získaná z* f*, tak že zachovává jeden* i*-tý bit konstantní a rovný* c *splňuje SAC pro každé* $i \in \{1, 2, \ldots, n\}$ *a* $c = 0$ *nebo* $c = 1$*.*

Obecně lze rozšíření definovat rekurzivně, a to až do řádu $k \leq n - 2$:

Definice 3.5 (SAC k-tého řádu)**.** *Funkce* $f : \{0,1\}^n \to \{0,1\}^m$ *splňuje SAC* k*-tého řádu pokud a jen tehdy:*

- f *splňuje SAC* $(k-1)$*-ho řádu*
- *každá funkce získaná z* f*, tak že zachovává* k *jejích bitů konstantních (pro všechny možné kombinace pozic a hodnot) a splňuje SAC*

Číslo funkce	00	01	10	11
1	0	0	0	1
2	0	0	1	0
3	0	1	0	0
4	1	0	0	0
5	0	1	1	1
6	1	0	1	1
7	1	1	0	1
8	1	1	1	0

Tabulka 3.1: Základní funkce splňující MOSAC [8]

SAC nejvyššího možného řádu $k = n - 2$ má zvláštní označení MOSAC[3]. V tabulce 3.1 jsou v řádcích uvedeny základní booleovské funkce splňující MOSAC. Tyto funkce mají 2-bitový vstup uvedený ve sloupcích a splňují SAC 0. řádu. Sloupce reprezentují všechny kombinace vstupů. Jak je patrné, výstupy jsou u každé funkce u třech bitů jsou stejné a jeden se liší.

Věta 3.1. *Silný s-box není lineární ani afinní.*

Důkaz. Důkaz je uveden v [8] na str. 25 □

3.2.2 Bent funkce

V [22] kombinatorice, je „bent funkce“ zvláštní typ booleovské funkce. „Bent[4] funkce“ jsou tak pojmenovány protože jsou odlišné od lineárních a afinních funkcí. Proto byly obsáhle studovány především pro jejich uplatnění v kryptografii, ačkoliv jejich spektrum aplikací je široké i v teorii kódování a návrhu kombinačních obvodů. Definice může být rozšířena v mnoha možnostech.

Definice 3.6. *[8] Booleovská funkce $g(w) : \{0,1\}^n \to \{0,1\}$, $n = 2l, l \in \mathcal{N}$ je bent funkcí, pokud každý koeficient Fourierovy transformace[5] $G(w)$ Walshovy funkce $(-1)^{g(w)}$ definované pro všechna $w \in \{0,1\}^n$*

$$G(w) = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{g(x)+x \cdot w} \tag{3.3}$$

má jednotkovou velikost

$$|G(w)| = 1 \tag{3.4}$$

Věta 3.2 (Vztah mezi bent funkcemi a funkcemi splňujícími SAC)**.** *Nechť $\mathcal{A}_n$ označuje množinu všech n-bitových booleovských funkcí, $\mathcal{B}_n$ množinu n-bitových bent funkcí a $\mathcal{S}_n$ množinu n-bitových booleovských funkcí splňujících SAC. A obzvlášť můžeme označit množinu n-bitových booleovských funkcí splňujících MOSAC jako $\mathcal{S}_n^{\max}$. Vztah mezi těmito množinami pro sudá n lze vyjádřit jako*

$$\mathcal{S}_n^{\max} \subseteq \mathcal{B}_n \subseteq \mathcal{S}_n \subset \mathcal{A}_n \tag{3.5}$$

[1] nebo podle nichž se dají sestavit
[2] počet jedničkových bitů
[3] z anglického „maximal order“
[4] česky: křivé
[5] tento zvláštní případ se nazývá častěji Walshova transformace

n	2	3	4	5	6
$\lvert\mathcal{A}_n\rvert$	16	256	65536	2^{32}	2^{64}
$\lvert\mathcal{S}_n\rvert$	8	64	4128	?	?
$\lvert\mathcal{B}_n\rvert$	8	–	896	–	$2^{32.3}$
$\lvert\mathcal{S}_n^{\max}\rvert$	8	16	32	64	128

Tabulka 3.2: Velikost množin z věty 3.2 [8]

Důkaz. Důkaz je uveden v [8] na str. 31 □

Toho lze využít k zúžení prohledávacího prostoru, při hledání funkcí splňujících SAC. Pro s-boxy s více výstupy lze z definice 3.1 předpokládat, že každý výstup je booleovská funkce a podle toho sestavit hodnotící funkci. Pro představu jsou v tabulce 3.2 uvedeny velikostí některých množin.

3.2.3 Největší odchylka z lineárních výrazů

K výpočtu nelinearity boolovské funkce f můžeme použít vzorec z [8], založený na Hammingově vzdálenosti od množiny afinních funkcí a na Walshově transformaci $\hat{F}$ funkce $\hat{f} : \{0,1\}^n \to \{1,-1\}$, obdobně jak u SAC:

$$\hat{f}(x) = (-1)^{f(x)} \tag{3.6}$$

$$\hat{F}(w) = \sum_{x\in\{0,1\}} f(x)\cdot(-1)^{x\cdot w} \tag{3.7}$$

$$\delta(f) = 2^{n-1} - \frac{1}{2}\max_{w\in\{0,1\}} |\hat{F}(w)| \tag{3.8}$$

Ale pro více výstupů bude jednodušší spočítat největší výchylku s-boxu q podle vzorce z [16] :

$$LP_{\max}(q) = \max_{a,b\neq 0} \left(2Pr_X[X\cdot a = q(X)\cdot b] - 1\right)^2 \tag{3.9}$$

Tento vzorec je založen na postupu, že se nejprve vytvoří se kompletní tabulka četností všech lineárních aproximací odpovídající tvaru rovnice 2.5. Řádky resp. sloupce odpovídají jednotlivým parametrům a resp. b, $\forall a, b$ kde $0 \leq a < n$ a $0 \leq b < m$[6], Každá buňka tabulky [7] reprezentuje pravděpodobnost výskytu (či počet shod) z lineárním výrazem odpovídajícímu parametrům a a b. A z nich se vybere největší linearita a ta se pak použije při hledání s-boxu, který má tuto maximální linearitu nejmenší. Odchylka je ve vzorci normalizovaná do intervalu $\langle 0, 1\rangle$. Jak vyplývá z textu uvedeného výše, zajímavá je její velikost, ne znaménko.

[6] n a m jsou dle def. 3.1

3.2.4 Největší pravděpodobnost z diferenciálů

Pro výpočet největší pravděpodobnosti diferenciálu s-boxu q použijeme tento vzorec [16], vycházející z diferenciální kryptoanalýzy:

$$DP_{\max}(q) = \max_{a \neq 0, b} Pr_X[q(X \oplus a) \oplus q(X) = b] \tag{3.10}$$

Tento vzorec je opět založen na tabulce [7] počtu výskytů (rozložení pravděpodobnosti) rozdílů, ve které každý řádek reprezentuje hodnoty $\Delta X = a$ a sloupce reprezentují hodnoty $\Delta Y = b$. Každá buňka tabulky reprezentuje počet případů, kdy je výstupní rozdíl roven b, pokud je vstupní rozdíl a. Poznamenejme, že například případ vyloučený z výpočtu mající parametry ($a = 0$, $b = 0$), má vždy největší pravděpodobnost v tabulce, která však nepatří ke skutečnému rozdílu. Obdobně jako u kritéria $LP_{\max}$ je největší pravděpodobnost použita z tabulky k minimalizaci. V [12] je toto kritérium nazýváno „tabulka XOR“.

3.2.5 Faktor větvení

Proces šifrování by měl kódovanou informaci pokud možno rozptýlit tak, aby prohledávací prostor byl pro útočníka co nejsložitější. Představme si prohledávací prostor ve tvaru stromu. Minimální počet následovníků každého uzlu určuje spodní odhad složitosti prohledávání, který je na něm závislý exponenciálně.

Faktor větvení (Branching Factor, Branch Number) používají v [4] jako hlavní kritérium v lineárním kroku šifry pojmenovaném „MixColumn“ po bajtech. Avšak nic nebrání tomu ji vyzkoušet i na nelineární prvek po bitech. Nechť f je s-box dle 3.1, $wt(x)$ značí Hammingovu váhu. Faktor větvení je míra mocnosti rozptýlení:

Definice 3.7. *Faktor větvení funkce f je*

$$\min_{\Delta x \neq 0, x} (wt(\Delta x) + wt(\Delta f(x))) \tag{3.11}$$

$$\Delta f(x) = f(x) \oplus f(x \oplus \Delta x) \tag{3.12}$$

Na vstupu a výstupu jsou ve skutečnosti rozdíly, počítá se minimální součet počtu změn ve vstupu a výstupu dohromady. Toto kritérium se použije k maximalizaci.

3.2.6 Řád algebraického polynomu

Funkci, jež popisuje s-box, je možné aproximovat nejen lineárními výrazy v $GF(2)$ aritmetice, ale i interpolovat polynomy v této aritmetice pro každý výstupní bit. Polynom pro jeden z bitů výstupu y_i lze definovat pomocí vzorce:

$$y_i(x) = \sum_{a \in \mathcal{A}} \left(\prod_{j=0}^{n} a_j \cdot z_j \right) \tag{3.13}$$

Kde z_0 představuje 1, $z_j = x_{j-1}$ jsou jednotlivé bity vstupu[7], $\mathcal{A}$ je množina parametrů určujících (zapínajících jednotlivé bity) polynomu. Řádem polynomu pro jeden výstupní bit je maximální Hammingova váha z množiny $\mathcal{A}$ (bit pro jedničku se nezahrnuje). Pro celý s-box je pak jako kritérium použitelný minimální řád z polynomů pro výstupy:

$$\min_{0 \leq i < m} \max_{a \in \mathcal{A}_i} \left(wt(a_{[1 \cdots n]}) \right) \tag{3.14}$$

[7] v předchozím textu se x počítá od 0

S-box 4×4 daný posloupností

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

vyjadřují tyto polynomy třetího stupně:

$$y_0 = 1 + a*b + c + b*c + a*b*c + d + a*d + a*b*d + c*d + a*c*d \quad (3.15)$$
$$y_1 = 1 + c + a*b*c + a*d + a*c*d \quad (3.16)$$
$$y_2 = 1 + a + c + a*b*c + a*b*d + c*d \quad (3.17)$$
$$y_3 = a + b + c + a*b*c + b*c*d \quad (3.18)$$

kde $a = x_0$ je první vstupní bit atd., operace + a $*$ jsou definovány v mod-2 aritmetice. A vyjadřují jak moc jsou bity výstupu závislé na vstupních bitech. Pokud má polynom nejvyšší možný řád, závisí výstupní bit na všech vstupech. Pro celý s-box je proto zvolen minimální řád ze všech polynomů vyjadřující výstupy. Jinou možností by mohl být i počet použitých proměnných, sčítání však vypovídá o linearitě.

3.2.7 Bijektivita

Některé kryptosystémy[8] vyžadují, aby funkce, kterou reprezentuje s-box byla bijektivní – tj. každý vstup byl zobrazen na unikátní výstup. Čehož lze sice jednoduše dosáhnout zakódováním problému jako permutace (viz dále), avšak pokud je chceme použít jiný typ reprezentace, musíme například bijektivitu stanovit jako další kritérium. V [8] je dále zmíněno omezení skládání bijektivního s-boxu z booleovských funkcí splňujících MOSAC. Mějme bijektivní funkci $f : \{0,1\}^n \to \{0,1\}^n$, každá lineární kombinace booleovských funkcí z nichž je složena musí mít Hammingovu váhu rovnu polovině možných výstupních kombinací:

$$wt(\sum_{i=1}^{n} a_i f_i) = 2^{n-1} \quad (3.19)$$

Jinými slovy řečeno: všechny lineární odchylky, dle výrazu 2.5 neobsahující vstupní bity, musí být nulové.

3.2.8 Další kritéria

Mezi další kritéria, která nebyla dále použita, patří například invertibilita funkce f. Nebo například kritérium BIC (nezávislost bitů) z [12] či „Cross Correlation of Avalanchne Variables“ z [8]. Obě vycházejí z avalanche efektu.

V tabulce 3.4 jsou jednotlivá použitá kritéria srovnána. Jsou zde uvedeny mezní hodnoty jednotlivých kritérií. Některé z nich jsou závislé od počtu vstupů a výstupů. U dvou z nich je hledána minimální hodnota. Hvězdičkou jsou označeny teoretické hodnoty, které nelze prakticky dosáhnout.

[8] pokud $n = m$ tak v tom případně by měly logicky snad všechny

Kritérium	úprava	minimální hodnota	maximální hodnota	druh optimalizace
bent skóre	počet w jejichž $\lvert G(w)\rvert = 1$	0	$m \cdot 2^n$	max
řád SAC		-1	$n-2$	max
diferencial probability		$* \frac{1}{2^n}$	1	min
linear bias probability		* 0	1	min
řád algebraického polynomu		0	n	max
branch faktor		0	$m+1$	max
bijektivní skóre	obdobné bent skóre	0	$m \cdot 2^n$	max

Tabulka 3.4: Srovnání jednotlivých kritérií bezpečnosti

Vnější bity	Vnitřní 4 bity vstupu							
	0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**
00	0010	1100	0100	0001	0111	1010	1011	0110
01	1110	1011	0010	1100	0100	0111	1101	0001
10	0100	0010	0001	1011	1010	1101	0111	1000
11	1011	1000	1100	0111	0001	1110	0010	1101
	1000	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
00	1000	0101	0011	1111	1101	0000	1110	1001
01	0101	0000	1111	1010	0011	1001	1000	0110
10	1111	1001	1100	0101	0110	0011	0000	1110
11	0110	1111	0000	1001	1010	0100	0101	0011

Tabulka 3.5: DES s-box S_5

3.3 Příklady dalšího užití s-boxů

S-boxy jsou popsány pro mnoho šifer. Následují příklady nejznámějších z nich:

3.3.1 S-box v DES

U DESu [26] se používají s-boxy 6×4. Dobrým příkladem je s-box S_5 uvedený v tabulce 3.5: Pro 6-bitový vstup je 4-bitový výstup určen vybráním řádku použitím dvou vnějších bitů (první a poslední), sloupec je určen vnitřními čtyřmi bity. Např. vstup „011011“ má vnější bity „01“ a vnitřní „1101“. Odpovídající výstup je „1001“. Osm s-boxů algoritmu DES bylo před mnoha lety předmětem intenzivního studia s podezřením na zadní vrátka, zranitelnost známá pouze jeho tvůrcům, která by mohla být umístěna v šifře. Kritéria návrhu s-boxů byla nakonec zveřejněna (Don Coppersmith, 1994), po znovuobjevení diferenciální kryptoanalýzy, přičemž byly pečlivě vylepšeny v odolnosti vůči tomuto útoku. Pozdější výzkum ukázal, že právě tyto malé změny mohly značně zeslabit DES.

3.3.2 S-box v AES

Šifra Rijndeal[4, 25] používá s-boxy ve fázi zvané „ByteSub“. Zde se jedná o s-boxy 8×8, tj. bajt na bajt. Byl navržen s ohledem na odolnost vůči lineární a diferenciální kryptoanalýze, aby měl nejvyšší stupeň polynomu byl invertibilní a jednoduše popsatelný. Proto vybrali následující afinní transformaci, modulární násobení následované sčítáním:

$$b(x) = (x^7 + x^6 + x^2 + x) + a(x)(x^7 + x^6 + x^5 + x^4 + 1) \mod x^8 + 1 \tag{3.20}$$

Přičemž $(b(a) \neq a)$ a $(b(a) \neq \bar{a})$.

S-boxy v Twofish

Zde [16] používají 8×8 ke konstrukci 8×32. Tyto malé s-boxy určeny pomocí náhodného prohledávání, při kterém hledali permutace s $LP_{\max} \leq \frac{1}{16}$ a $DP_{\max} \leq \frac{10}{256}$.

Kapitola 4

Evoluční algoritmy

Evoluční algoritmy jsou informované metody prohledávání stavového prostoru, které [17] jsou založeny na metafoře evoluce v přírodě. Řešení nějaké úlohy je převedeno na proces evoluce populace náhodně vygenerovaných řešení. Každé řešení je zakódováno do řetězce symbolů (parametrů) a ohodnoceno tzv. fitness funkcí, která vyjadřuje kvalitu řešení – čím je hodnota větší tím je dané řešení perspektivnější a častěji vstupuje do reprodukčního procesu během něhož jsou generována nová řešení. Populace řešení se běžně nazývá populací individuí nebo chromozómů. Reprodukční proces je založen na dvou hnacích silách:

- variační operátory křížení a mutace, které přinášejí rozmanitost populace/diverzitu
- selekce, která upřednostňuje kvalitní jedince

Kombinace variace a selekce přispívá obecně ke zlepšení fitness funkce jedinců v nově se tvořící populaci. V procesu křížení jedinců, tak jako v živé přírodě jsou nová individua/potomci získávána křížením většinou dvojic rodičovských individuí. Všechny komponenty evolučního procesu jsou stochastické častěji např. vstupují do reprodukčního procesu páry s lepší fitness funkcí, ale i slabší jedinci mají šanci vstoupit do reprodukčního procesu.

Nejprve jsou na nejrozšířenějším typu evolučního algoritmu ukázány základní pojmy. U dalších algoritmů jsou uvedeny již jen odlišnosti.

4.1 Genetický algoritmus

Chromozóm (jedinec/individuum) kódující řešení je reprezentován binárním vektorem (řetězcem) konstantní délky n:

$\vec{X} = (X_0, X_1, \ldots, X_{n-1})$, kde X_i je i - tou proměnnou řetězce

$\vec{x} = (x_0, x_1, \ldots, x_{n-1})$ je řetězec konkrétních instancí proměnných

$X_i = x_i, x_i \in \{0, 1\}$

$D = (X^1, X^2, \ldots, X^N), X^j \in D$ je množina N řetězců, která specifikuje populaci D

$D \subseteq \{0, 1\}^n$.

Nechť f je účelová/cenová funkce definovaná nad množinou binárních vektorů délky n

$$f : \{0, 1\}^n \to \mathcal{R} \tag{4.1}$$

která ohodnotí každý binární vektor $\vec{x}$ reálným číslem. Cílem je nalézt globální extrém funkce f. V případě minimalizační úlohy jde o nalezení vektoru

$$\vec{x}_{opt} = \arg\min_{\vec{x}\in\{0,1\}^n} f(\vec{x}) \tag{4.2}$$

Funkce f se zpravidla transformuje na funkci výhodnosti F (fitness funkci) tak, aby původní optimalizační úloha byla převedena na maximalizační úlohu a bylo dosaženo vhodného měřítka fitness funkce. Užití této transformace usnadňuje také změnu selekčního tlaku, který výrazně ovlivňuje konvergenci evolučního procesu. Činnost standardního GA algoritmu popisuje algoritmus 4.1.

Algoritmus 4.1 (Pseudokód genetického algoritmu)**.**

Nastav $t = 0$, náhodně generuj počáteční populaci $D(0)$ s mohutností N,
while** není splněna podmínka pro ukončení algoritmu **do
Proveď ohodnocení jedinců populace $D(t)$ fitness funkcí $F(X)$,
Generuj populaci potomků $O(t)$ s mohutností $M \leq N$ použitím operátorů křížení a mutace,
Vytvoř novou populaci $D(t+1)$ nahrazením části populace $D(t)$ jedinci z $O(t)$,
Nastav $t \leftarrow t + 1$,
end while

V procesu reprodukce jsou většinou kvazináhodně vybírány dvojice rodičovských jedinců pro následné křížení a mutaci. Část jedinců z populace potomků se potom použije pro nahrazení jedinců původní populace $D(t)$ a takto získáme novou populaci $D(t+1)$. Ukončení evolučního procesu je buď dáno maximálním počtem generací nebo detekcí stagnace vývojového procesu. Genetický algoritmus může mít několik typů křížení a mutace. Některé operátory bývají navrženy podle typu řešené úlohy. Mutace je většinou provedena inverzí bitu/ů na náhodných pozicích. Křížení může být například jednobodové či dvoubodové. Po vybrání dvou rodičů se nejprve se zvolí bod(y), ve kterém se chromozóm obou rodičů rozdělí na dvě části. Vznikají dva potomci. První z nich je složen z první části prvního rodiče a z druhé části druhého rodiče. A chromozóm druhého potomka je poskládán ze zbývajících částí. Selekce může probíhat:

- turnajem mezi N náhodně vybranými jedinci,
- proporcionálně (podíl fitness jedince na součtu fitness všech jedinců) neboli tzv. ruleta
- lineárním uspořádáním (pravděpodobnost výběru je lineární funkcí pozice po seřazení populace podle fitness)
- exponenciálním uspořádáním (jako lineární, leč pravděpodobnost je závislá exponenciálně)

4.2 Genetické programování

Reprezentace Genetické programování [10] pracuje s tzv. spustitelnými strukturami. Nejčastěji se jedná o programy, které jsou reprezentovány stromy a grafy. Tyto struktury mají (ale nemusí vždy mít) proměnnou délku. Narozdíl od klasického GA se zde pracuje přímo se spustitelnou strukturou, která představuje jak genotyp tak fenotyp kandidátního řešení.

Genetické operátory pracují nad spustitelnými strukturami. Kromě běžných operátorů (jako jsou křížení a mutace) existuje řada pokročilých operátorů umožňujících generovat programy s podprogramy, moduly atd.

Evaluace V rámci zjištění fitness hodnoty je proveden kód kandidátního programu (obecně spustitelné struktury) pro definovanou množinu vstupů a jsou vyhodnoceny získané výsledky.

Zatímco v obecném genetickém programování dochází k prohledávání v prostoru řešení, který se v průběhu evoluce mění, v tomto textu se budeme zabývat pouze statickým stavovým prostorem. V genetickém programování lze navrhnout libovolnou spustitelnou strukturu, která nejlépe vystihuje řešený problém. A k této struktuře navrhnout vlastní operátory.

4.2.1 Kartézské genetické programování

Kartézské genetické programování (CGP) [20] lze chápat jako variantu genetického programování, která v reprezentaci řešeného problému používá acyklický graf s pevným počtem uzlů. Každý z uzlů může realizovat jednu z několika pevně daných funkcí a může být spojen na výstup některého z předešlých uzlů. Konektivitu ovlivňuje tzv. *l-back* parametr. Chromozom je určitá posloupnost celočíselných hodnot mající fixní délku. V tom lze spatřovat dělení na genotyp a fenotyp jako u klasických GA. Algoritmus používaný v CGP odpovídá evoluční strategii $(1 + L)$, kde hodnota L se pohybuje řádově v jednotkách. Používá se pouze operátor mutace.

Kódování řešeného problému

Chromozom je složen z $m \cdot n$ trojic (m - počet sloupců, n - počet řad) celočíselných hodnot, kde první dvě hodnoty udávají číslo výstupu, na který je připojen první a druhý vstup trojici příslušejícího elementu, třetí hodnota udává jakou logickou funkci element realizuje. Na konci chromozomu je o-tice o velikosti shodné s počtem výstupů hledaného obvodu. Každý prvek o-tice určuje index elementu (jeho výstupu), na který bude primární výstup obvodu připojen. Dalšími parametry jsou i - počet vstupů, l - *l-back*, F - množina funkcí.

Uvažujme následující tvar chromozomu, odpovídající jednoduché booleovské funkci[1], splňující SAC:

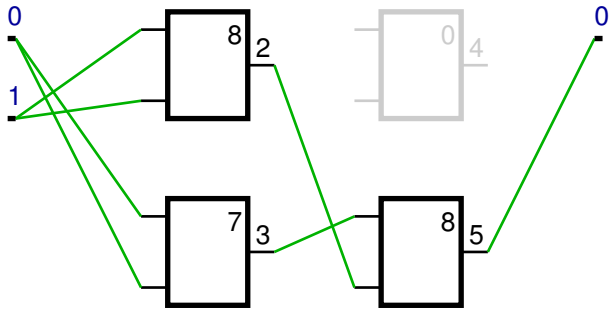
```
{2,1,2,2,2,1,3}([2]1,1,8)([3]0,0,7)([4]1,3,0)([5]3,2,8)(5)
```

Tvar chromozómu je vysvětlen v příloze A. Na obrázku 4.1 je tento chromozóm znázorněn v grafové podobě.

Programová implementace

Během [11] generování nových jedinců nebo při mutaci musí být splněno několik podmínek, aby chromozóm reprezentoval správný program. Nechť ${c^n}_{kj}$ reprezentuje gen k-tého vstupu

[1] Eliminací hradel z ní lze získat AND, prostě nejjednodušší bent funkce uvedená pro názornost



Obrázek 4.1: Příklad jednoduché Booleovské funkce splňující SAC nalezené kartézským genetickým programováním

ve sloupci j (nejlevější sloupec má $j = 0$), potom musí splňovat tato omezení:

$$c_{kj} < e_{\max} \quad pokud \quad j < l \tag{4.3}$$

$$e_{\min} \leq c_{kj} < e_{\max} \quad pokud \quad j \geq l \tag{4.4}$$

$$e_{\min} = i + (j - l)n \tag{4.5}$$

$$e_{\max} = i + jn \tag{4.6}$$

Nechť ${c^o}_k$ reprezentuje gen k-tého výstupu celého obvodu, pak pro něj platí tato omezení:

$$h_{\min} \leq {c^o}_k < h_{\max} \tag{4.7}$$

$$h_{\min} = i + (m - l)n \tag{4.8}$$

$$h_{\max} = i + (m - 1)n \tag{4.9}$$

Nechť ${c^f}_k$ reprezentuje gen funkce k-tého uzlu, pak:

$${c^f}_k \in F \tag{4.10}$$

Z těchto podmínek je patrná i skutečnost, že realizace křížení, při, kterém by bylo nutné dodržet tyto podmínky, by byla komplikovanější a i proto se raději nepoužívá.

4.3 EDA

[17] Během operace křížení klasického genetického algoritmu dochází k rozbíjení stavebních bloků. Tato skutečnost je častou příčinou špatné konvergence optimalizačního procesu. Proto byly navrženy algoritmy, které se tento problém snaží vyřešit pomocí globálního pohledu na celou populaci. V celé populaci jsou spočteny statistiky výskytů a závislostí alel na jednotlivých pozicích. Podle těchto statistik jsou následně generováni jedinci noví. Tyto algoritmy jsou souhrnně nazývány EDA – Estimation of Distribution Algorithm.

4.3.1 Činnost EDA algoritmu

Na rozdíl do GA se v každém kroku vybere $M < N$ „slibných“ jedinců, ze kterých se provede odhad pravděpodobnostního rozložení vybraných jedinců $p(X|Ds(t))$. Poté se vygeneruje

množina potomků $O(t)$ s mohutností $M < N$ (vzorkováním získaného rozložení). Nová populace $D(t+1)$ je vytvořena nahrazením části populace $D(t)$ jedinci z $O(t)$.

Fundamentálním problémem je odhad pravděpodobnostního rozložení $p(X|Ds(t))$. Z teorie je znám vztah pro sdružené rozložení pravděpodobnosti náhodného vektoru pomocí lokálních pravděpodobnostních rozložení s využitím podmíněných pravděpodobností:

$$p(X) = \prod_{i=0}^{n-1} p(X_i|X_0, X_1, \ldots, X_{i-1}) \tag{4.11}$$

A právě v tomto výpočtu (složitosti pravděpodobnostního modelu) se EDA algoritmy dělí na více druhů, které jsou uvedeny níže.

4.3.2 UMDA

UMDA (Univariate Marginal Distribution Algorithm) používá nejjednodušší statistický model. Předpokládá, že lokusy jsou navzájem nezávislé, výskyt allel na každé pozici je řízen distribuční funkcí. Nechť se populace P o velikosti N skládá z chromozomů délky n. Pro každý lokus $i \in \{0..n-1\}$ a každou allelu $x_i \in \{0, \ldots, r_{i-1}\}$ definujeme $p_i(x_i)$ jako četnost výskytu řetězců, které mají x_i na i-té pozici v množině P.

$$p_i(x_i) = \frac{n_i(x_i)}{N} \tag{4.12}$$

Pravděpodobnost, se kterou bude vygenerován algoritmem UMDA nový jedinec $X = a_1 a_2 \ldots a_{n-1}$ pak musí být

$$p(X) = \prod_{i=0}^{n-1} p_i(a_i) \tag{4.13}$$

Tedy nejprve jsou z populace slibných řešení na každé pozici vypočítány četnosti různých allel, poté je pro každé nové individuum bit na i-té pozici nastaven na hodnotu a s pravděpodobností $p_i(a)$. Graf závislosti odpovídající UMDA algoritmu neobsahuje žádné hrany ($E = \emptyset$).

4.3.3 BMDA

Kromě nezávislých lokusů algoritmus BMDA (Bivariate Marginal Distribution Algorithm) připouští i podvojné závislosti mezi lokusy, které jsou zjišťovány statistickými testy. Pro generování hodnot na závislých pozicích je využíván aparát podmíněné pravděpodobnosti. Opět uvažujeme populaci P o velikosti N s chromozomy délky n. Pro každé dvě pozice $i \neq j \in \{0..n-1\}$ a každé dvě možné allely $x_i \in \{0, \ldots, r_{i-1}\}$, $x_j \in \{0, \ldots, r_{j-1}\}$ na těchto pozicích definujeme $p_{i,j}(x_i, x_j)$ jako četnost výskytu řetězců s hodnotami x_i a x_j na pozicích i a j:

$$p_{i,j} = \frac{n_{i,j}(i,j)}{N} \tag{4.14}$$

Podmíněná pravděpodobnost

$$p_{i,j}(x_i|x_j) = \frac{p_{i,j}(x_i, x_j)}{p_j(x_j)} \tag{4.15}$$

je pak odhadována z četnosti výskytu allely x_i na pozici i v případech, kdy je x_j na j-té pozici. Graf závislosti [17, 14] pro BMDA algoritmus má stromovou strukturu, každý

gen je závislý maximálně na jednom jiném genu. Některé skupiny genů mohou být zcela nezávislé, pak má graf podobu množiny stromů. Pravděpodobnost generování jedince $X = a_1 a_2 \dots a_{n-1}$ pak je

$$p(X) = \prod_{j \in R} p(a_j) \prod_{(i,j) \in E} p(a_j | a_i) \tag{4.16}$$

kde R resp. E je množina kořenových uzlů (tedy nezávislé proměnné) resp. hran grafu G.

K testování nezávislosti kvalitativních znaků se ve statistice používá rozdělení χ^2

$$\chi^2 = \sum \frac{real\_value - expected\_value}{expected\_value} \tag{4.17}$$

V našem případě se jedná o testování nezávislosti obsahů dvou lokusů:

$$\chi^2_{i,j} = \sum_{x_i=0}^{r_i-1} \sum_{x_j=0}^{r_j-1} \frac{(N p_{i,j}(x_i, x_j) - N p_i(x_i) p_j(x_j))^2}{N p_i(x_i) p_j(x_j)} \tag{4.18}$$

kde r_i je počet různých allel na lokusu i, r_j počet různých allel na lokusu j (u binárních chromozomů $r = s = 2$). Tento vzorec vychází z předpokladu, že pro výskyt hodnot na nezávislých pozicích musí podle teorie pravděpodobnosti platit $p_{i,j}(x_i, x_j) = p_i(x_i) p_j(x_j)$. Pro všechny dvojice lokusů jsou při průchodu populací sestaveny kontingenční tabulky. S pomocí těchto tabulek je pak pomocí χ^2-testu vytvořena množina závislých dvojic, která je uspořádána podle hodnoty χ^2. Dva binární lokusy jsou závislé na 95%, pokud hodnota χ^2 je větší než 3,84. Z množiny závislých dvojic je utvořen strom závislostí, který je pak použit ke generování nových jedinců.

Další varianty EDA algoritmů (např. BOA) z důvodů uvedených níže nebyly použity.

4.4 Multikriteriální optimalizace

Úloha hledání s-boxů může být založena na více kritériích, které mohou jít proti sobě. V multikriteriální optimalizaci nehledáme prosté optimum jako v jednokriteriální, ale snažíme se najít vhodný kompromis při optimalizaci jednotlivých kritérií. Formálně tedy hledáme vektor parametrů $\vec{x}^* = [x_1^*, x_2^*, \dots, x_n^*]^T$ který:

- splňuje m omezení: $g_i(\vec{x}) \leq 0, i = 1, 2, \dots, m$
- splňuje p omezení: $h_i(\vec{x}) = 0, i = 1, 2, \dots, p$
- bude optimalizovat funkci: $f(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \dots, f_p(\vec{x}))$

Z toho důvodu bylo třeba tento problém zahrnout do evolučních algoritmů. K tomuto účelu bylo navrženo několik postupů.

4.4.1 Pareto optimum

- Pro jakékoliv dva vektory $\vec{a}$, $\vec{b}$ říkáme, že vektor $\vec{a}$ silně dominuje nad vektorem $\vec{b}$, když platí:
  1. $f_i(\vec{a}) \geq f_i(\vec{b})$ pro $\forall i = 1, \dots, p$
  2. a zároveň platí: $f_i(\vec{a}) > f_i(\vec{b})$ alespoň pro jedno i

- Vektor $\vec{a}$ pak slabě dominuje nad $\vec{b}$, když platí pouze první podmínka
- Na základě Pareto dominance lze definovat pojem „Pareto Optimum“: Vektor $\vec{a}$ je Pareto optimální pokud neexistuje žádný vektor $\vec{b}$ takový, který dominuje vektoru $\vec{a}$. Jinými slovy: vektor $\vec{a}$ je Pareto optimální, pokud v rozhodovacím prostoru neexistuje žádný vektor $\vec{b}$, který by snížil nějaké kritérium bez toho, aniž by způsobil souběžné zvýšení alespoň jednoho dalšího kritéria.
- Tento koncept nám ale nedává jen jediné řešení, ale většinou celou množinu řešení, tzv. Pareto optimální množinu řešení (Pareto optimalset).
- Vektory $\vec{a}$, které jsou i v Pareto optimální množině (POS) jsou nazývány nedominantními. Křivka/hyperplocha vedená těmito nedominantními vektory z POS se nazývá Paretova fronta.
- Řešením MOP je pak libovolný bod z Paretovy fronty

4.4.2 Algoritmus VEGA

VEGA je nejjednodušší možný multikriteriální GA. Pro několik kritérií (značíme M) musí být řízeno dělení GA populace v každé generaci do M stejných podskupin náhodně. Každé podskupině je přidělena fitness funkce založená na kriteriální funkci, odlišné pro každou podskupinu. Populace je v každé generaci rozdělena na M stejných dílů. Každému jedinci v první podskupině je přidělena fitness funkce založená jen na první kriteriální funkci, zatímco každému jedinci ve druhé podskupině je přidělena fitness funkce podle druhé kriteriální funkce, a tak dále. Je lepší promíchat populací předtím, než je rozdělena do podskupin. Pro každé řešení je přidělená fitness funkce, výběrový operátor, vyhrazený mezi řešeními každé podskupiny. Výběrový operátor je aplikován do té doby, než je kompletní podskupina naplněna. Od té chvíle mají všichni jedinci v podskupině přidělené fitness funkce založené na jednotlivé kriteriální funkci. Ta omezuje operaci výběru jen uvnitř podskupiny a zdůrazňuje dobrá řešení odpovídající té které jednotlivé kriteriální funkci. Ve VEGA je v podpopulacích využíván proporcionální výběrový operátor.

4.4.3 Algoritmus SPEA

Tento algoritmus [17, 19] udržuje externí elitní populaci P_e. Tato populace obsahuje pevné množství nedominovaných řešení, které byli nalezeny od začátku evoluce. V každé generaci jsou nově nalezená nedominovaná řešení srovnávána s existující externí populací a výsledné nedominující řešení jsou uchována. SPEA dělá více, než jen uchovává elitu. Také užívá tuto elitu k účasti na genetických operacích spolu s nynější populací.

Tento algoritmus začíná s náhodně vytvořenou populací P_0 velikosti N a prázdná externí populace P_{0e} s maximální kapacitou N_e. V každé generaci t, jsou nejlepší nedominovaná řešení (patřící první nedominované frontě) z populace P_t kopírována do externí populace P_{te}. Dominovaná řešení v upravené externí populaci jsou nalezena a odstraněna z této populace. V externí populaci zůstávají nejlepší nedominovaná řešení kombinované populace obsahující staré a nové elity. Za účelem omezení přerůstání populace, je velikost externí populace vázána na limit N_e. Pokud velikost populace je rovná velikosti externí populace, jsou všechna řešení obsažena i v externí populaci. Nicméně, pokud velikost populace překračuje N_e, ne všechny elity mohou být umístěny v externí populaci. Elitní jedinci, kteří jsou v méně

přeplněných regionech v nedominované frontě, jsou uloženi. Jakmile jsou nové elity uchované pro další generaci, algoritmus se obrátí na nynější populaci a za využití genetických operátorů, nalezne populaci novou. První krok je přiřadit fitness ke každému řešení v populaci (i externí). Ve skutečnosti se v algoritmu SPEA nejprve přiděluje fitness (síla) S_i ke každému členu i z externí populace. Síla S_i je úměrná počtu nynějších členů populace (n_i), kteří externímu řešení i dominují:

$$S_i = \frac{n_i}{N+1} \tag{4.19}$$

Tím je zajištěno, že větší sílu má elita, která ovládá více řešení v aktuální populaci. Dělení hodnotou $(N+1)$ zabezpečí, že maximální hodnota síly libovolného členu externí populace bude vždy menší nežli 1. Navíc, nedominované řešení dominující nad menšími řešeními má lepší fitness. Fitness nynějšího člena populace j je určena jako suma sil všech externích členů, které silně[2] dominují řešení j:

$$F_j = 1 + \sum_{i \in P_{te} \wedge i \leq j} S_i \tag{4.20}$$

Přidání jedničky dělá hodnotu fitness libovolného nynějšího člena populace P_t větší než hodnotu fitness jakéhokoliv externího člena populace P_{te}. Externí členové populace získávají menší fitness hodnoty než členové nové populace. Členové populace, kteří jsou dominováni mnoha externími členy, dostanou velkou fitness hodnotu. Proto se při selekci nad oběma populacemi se preferují menší hodnoty (tzv. minimalizace). Obě populace jsou pak použity jako celek při křížení a mutaci.

Shlukování

Algoritmus seskupování redukuje velikost externí populace P_{te} velikosti z N'_e na N_e (kde $N'_e > N_e$). Zpočátku je každé řešení v P_{te} považované za jedno řešení v jednom odděleném shluku. Takže je v externí populaci zpočátku N'_e shluků. Následně jsou vypočítány vzdálenosti mezi jednotlivými páry shluků. Obvykle, vzdálenost d_{12} mezi dvěma shluky C_1 a C_2 je definována jako průměr euklidovské vzdálenosti všech párů řešení :

$$d_{12} = \frac{\sum_{i \in C_1, j \in C_2} d(i,j)}{|C_1||C_2|} \tag{4.21}$$

Jakmile jsou vypočítány všechny shluky, jsou dva shluky s minimální vzdáleností sdružené dohromady a společně tvoří jeden větší shluk. Tento proces slučování pokračuje dokud počet shluků v externí populaci není snížen na velikost N_e. Potom se v každém shluku ponechá jen řešení s minimální průměrnou vzdáleností od dalších řešení v shluku, a všechna další řešení jsou vymazána.

V tabulce 4.2 jsou oba algoritmy shrnuty. Zatímco VEGA více straní jednomu kritériu, SPEA se snaží najít širší spektrum řešení.

[2] v původním textu [17] je slabě, v implementaci však byla použita dominance silná

	Výhody	Nevýhody
VEGA	• jednoduchá idea, snadná k realizaci • potřeba jen nepatrných změn jednokriteriálního GA • stejná výpočetní složitost jako GA • tendence nalézt řešení blízko individuálního nejlepšího řešení každého kritéria	• hodnotí každé řešení pouze jedinou kriteriální funkcí • nenalézá různorodá řešení v populaci, směřuje všechna řešení k jednomu šampiónovi
SPEA	• ukládá řešení nejbližší Pareto-optimální frontě za celý průběh evoluce • shlukování umožňuje lépe rozprostřít omezenou velikost optimálních řešení	• parametr N_e – musí být zajištěna rovnováha s N vhodným poměrem

Tabulka 4.2: Srovnání algoritmů pro multikriteriální optimalizaci

4.5 Další metody k hledání s-boxů

K hledání s-boxů by bylo možné použít i další metody prohledávání stavového prostoru. Pro informované metody se kritérium použije jako heuristická funkce dané metody. Cílový stav je definován požadovanou hodnotou kritéria, časem či počtem kroků (generací). O použití heuristik v prohledávání s-boxů je zmínka např. v [1].

Gradientní algoritmus – horolezecký

Vybírá [9] k expanzi odvozený uzel, který je nejslibnější. Rodič i sourozenci jsou zapomenuti. Pokud jsou všechny odvozené uzly horší, algoritmus končí. Takto často končí v lokálním extrému.

Uspořádané prohledávání

Narozdíl od gradientního algoritmu ukládá neexpandované uzly do prioritní fronty (či seřazeného seznamu) podle ohodnocení. Nevýhodou je velká paměťová náročnost. Velikost fronty lze omezit, ale tím se toto prohledávání přiblíží ke gradientnímu a bude náchylné na skončení/uvíznutí v lokálním extrému.

Náhodné prohledávání

Je použito v [16]. Toto prohledávání je jako jediné z uvedených slepé. Odvodí nové řešení náhodnou změnou (mutací) aktuálního řešení. Může být ukládáno průběžně nejlepší řešení. Slouží i k porovnání při ladění evolučního algoritmu.

Paralelní náhodné prohledávání

Narozdíl od náhodného prohledávání, které je v principu zcela slepé, toto vyhledávaní v jednom kroku vygeneruje z aktuálního jedince více mutantů (pracuje s populací řešení – paralelnost) a z nich vybere nejlepšího, který se liší od aktuálního, a ten se použije pro mutace v dalším kroku. Defakto se používá v Kartézském genetickém programováni (viz str. 23).

Srovnání těchto náhodného a paralelního náhodného prohledávání je součástí následujícího textu v podkapitole 7.3.

4.6 Hledání s-boxů vs. symbolická regrese

Pokud lze hledání s-boxů, tedy alespoň jeho část, převést na nějaký teoretický problém řešitelný pomocí evoluce, bude to nejspíš symbolická regrese. Symbolickou regresí se míní hledání matematického výrazu, jenž popisuje data z trénovací množiny. Výraz může být reprezentován jako jedinec v evolučním procesu, pro kterého jsou v průběhu výpočtu hodnotící funkce zkoušeny hodnoty trénovací množiny. Hodnotící funkce je dána například sumou kvadrátů odchylek na požadovaných výstupech nebo počet shod. Proces evoluce je v tomto pohledu analogický s učením neuronové sítě. Výraz lze reprezentovat jako strom (původní genetické programování), derivaci gramatiky (gramatická evoluce) či jakoukoliv instanci spustitelné struktury či chromozóm, jenž nejlépe vystihuje řešený problém, pokud například hledáme optimální hardwarovou či softwarovou realizaci.

Při hledání s-boxů můžeme uvažovat tyto varianty hledání:

1. Nedá se hovořit o žádné trénovací/testovací množině. Je hledáno podle kritérií bezpečnosti, tudíž nejde o symbolickou regresi.

2. Je hledán výraz jenž přesně vyhovuje trénovací množině, získané z předchozí fáze. Například pokud napřed nalezneme vhodnou (bezpečnou) permutaci a poté k ní necháme najít optimální realizaci. Trénovací množina je složena ze všech možných vstupních kombinací.

Symbolickou regresi lze využít pouze jako pomocný mechanizmus k hledání optimální realizace. Symbolická regrese však musí nejprve najít přesnou realizaci.

Kapitola 5

Návrh programu pro experimenty

5.1 Shrnutí technických a funkčních požadavků

Cílem práce je implementace programu, který hledá s-boxy pomocí evolučních algoritmů, a následné provedení experimentů s tímto programem. Implementovaný program má v tomto případě tyto hlavní technické nároky, seřazené sestupně podle důležitosti:

Snadné definování experimentů Při popisu experimentů je nutné se vyhnout rutinním krokům, např. komplikovaným deklaracím a době kterou zdržuje kompilace kódu, nejlépe pokud by je šlo zadávat a vyhodnocovat interaktivně.

Rychlost Běh evoluce a výpočet kritérií se musí provést co nejrychleji.

Přenositelnost Program by měl jít zkompilovat a spustit na nejširší možné skupině platforem používaných pro vědecké výpočty na běžném osobním počítači.

Požadavky na funkčnost:

Různé reprezentace s-boxů Kvůli nalezení s-boxu v nejvhodnější reprezentaci (viz podkapitola 5.3).

Různé druhy prohledávacích algoritmů Pro účely srovnání aplikovatelnosti EA (viz kapitola 4) je dobré je porovnat s náhodným prohledáváním.

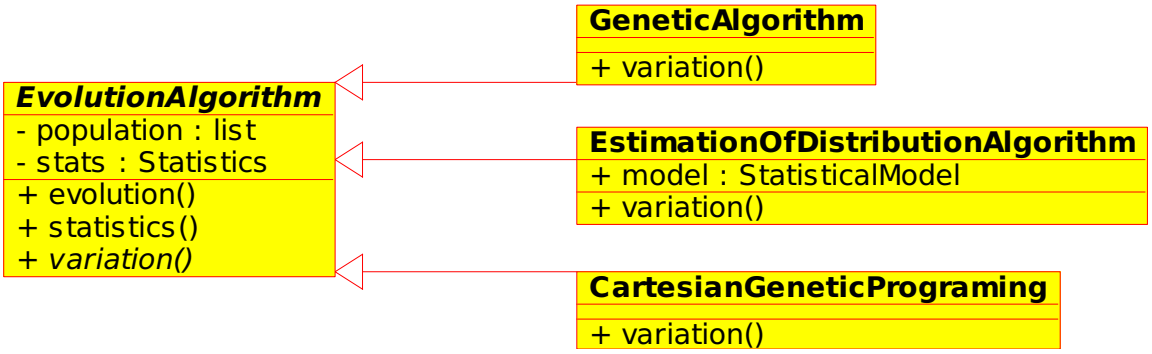
Prohledávací kritéria Požadovaná prohledávací kritéria (viz. podkapitola 3.2) jsou parametrem prohledávacích algoritmů.

Spuštění experimentů Vlastní spouštění experimentů by mělo mít jednotné aplikační rozhraní. Možnost nejprve nadefinovat více experimentů a poté je spustit najednou (např. paralelně).

Sběr statistik Pro vyhodnocení jednotlivých běhů je nutné, aby program během evoluce zaznamenával statistiky. Evoluční experimenty se často vyhodnocují pomocí tzv. „krabicových grafů“, pro které je nutné spočítat minimální, maximální hodnotu a kvartily, odchylku fitness při pevném počtu generací. Nebo [11] pomocí porovnání jaké úsilí (např. počet generací, velikost populace, doba běhu) s jak dobrými výsledky (nejvyšší fitness) bylo vynaloženo.

5.2 Prototypový návrh aplikace

Po stanovení hlavních požadavků byl nejprve pro ujasnění, kolik toho je třeba naimplementovat, sestaven prototyp aplikace, ve formě obdobné UML diagramům, obsahuje třídy s prázdnými metodami. Nadtřída evolučních algoritmů uchovává populaci kandidátních řešení. Obsahuje obecnou logiku pro průběh evoluce a sběr statistik. Obrázek 5.1 znázorňuje hlavní evoluční algoritmy.



Obrázek 5.1: Prototypový diagram tříd evolučních algoritmů

Každý prohledávací algoritmus má v každé generaci svojí specifickou změnu populace, definovanou pomocí metody `variation()`. Při konstrukci objektů (před spuštěním `evolution()`) je předán typ reprezentace s-boxu a parametry evoluce. Množiny evaluátorů kritérií jsou předány též při konstrukci. U EDA algoritmu je důležitý atribut `model()`, který reprezentuje statistický model (poddruh EDA) bude použit. Jednoduchost diagramu se ukázala důležitá ve fázi implementace, kdy bylo na základě něho vytvořeno rozhraní mezi Pythonem a C++ a byl dále zesložiťován.

5.3 Různé typy reprezentace s-boxu

Některé evoluční algoritmy mohou vyžadovat určitou reprezentaci substitučního boxu. Některé z reprezentací mohou umožňovat i optimální realizaci, pokud jsou modelem cílového systému. Každá z nich má svou specifickou inicializaci, mutaci a křížení.

5.3.1 Seznam výstupů

Nejvíce se s-boxy reprezentují jako seznam výstupů (pole) o velikosti všech vstupních kombinací, kde vstup s-boxu je indexem a výstup je prvek na dané pozici. Pokud má být s-box bijektivní tak tento seznam představuje permutaci. Například

2 0 3 1

je s-box 2×2, který je bijektivní. U této reprezentace je možné zvolit jestli se má vyžadovat bijektivita. Podle toho jsou určeny genetické operátory. Pro bijektivní se použije mutace pomocí prohození genů. Křížení probíhá jako u permutačních problémů na základě pořadí. Pokud by se bijektivita nevyžadovala křížilo by se jednobodově, stejně jako u následující reprezentace, ale s tím, že zde jiná množina alel.

5.3.2 Binární reprezentace

Binární reprezentace je složená z booleovských funkcí pro každý vstup. Bit booleovské funkce na i-té pozici představuje výstup, když je na vstupu i. S-box z předcházející reprezentace, lze pomocí binární zapsat takto (sloupce stále přestavují tutéž hodnotu):

$$\begin{matrix} 0\ 0\ 1\ 1 \\ 1\ 0\ 1\ 0 \end{matrix}$$

A každý řádek představuje booleovskou funkci. Mimochodem tento s-box lze reprezentovat i jako polynomy jednotlivých funkcí, kde proměnné jsou vstupních bitů:

$$s(x)_0 = b, s(x)_1 = 1 \oplus a$$

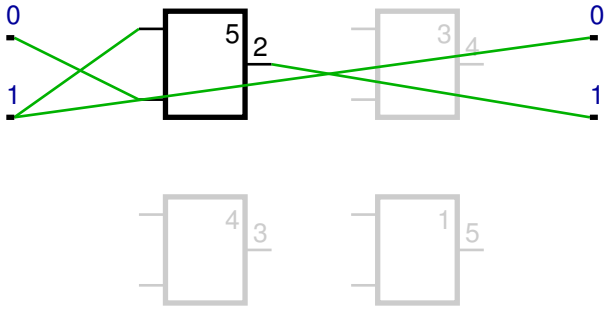
Pro genetické operátory je seznam booleovských funkcí brán jako jednolitý chromozóm, nad kterým je provedeno jednobodové křížení a mutace změnou bitu na náhodné pozici.

5.3.3 Acyklický graf hradel

Tato reprezentace pochází z kartézského genetického programování (značení viz příloha A) a je vhodnou pro hledání optimální realizace v hardwaru. S-box z předchozí reprezentace lze popsat například chromozómem ve tvaru

```
{2,2,2,2,2,1,1}([2]1,0,5)([3]1,0,4)([4]1,3,3)([5]3,3,1)(1,2)
```

který je též uveden na obrázku 5.2. První (index 0) výstupní bit je napojen na druhý vstupní. A druhý výstupní bit je negací prvního vstupu, což odpovídá polynomům $s(x)_0 = b, s(x)_1 = 1 \oplus a$. V této reprezentaci je přípustná pouze mutace splňující podmínky již



Obrázek 5.2: Příklad acyklického grafu hradel

zmíněné v podkapitole 4.2.1.

5.3.4 Posloupnost trojinstrukcí platformy i686

Tato reprezentace byla inspirována implementací SubCrumb algoritmu Luffa [2]. S tou obměnou, že je na vstupech a výstupech určená pro bity. Počet vstupních i výstupních bitů jsou násobky čtyř. Jediným parametrem struktury je počet čtveřic bitů p na vstupech a výstupech (počet vstupů a výstupů se shoduje). Reprezentace je modelem procesorů řady

i686, z důvodu hledání optimální realizace na těchto procesorech – optimální v softwaru. Model je složen z pěti registrů, v každém je uloženo p bitů. Výpočet každého s-boxu je složen z šesti cyklů. V každém cyklu jsou tři instrukce o nichž se předpokládá, že je procesor provede paralelně, proto se nesmí opakovat stejný registr na výstupu instrukce (paralelismus CREW). Instrukcemi, které se mohou vyskytnout v každém cyklu jsou XOR, AND, OR, NOT. Navíc v prvním cyklu je možné provést MOV a v posledním NOP.

Průběh výpočtu začíná rozdělením vstupu na čtyři části a_0, a_1, a_2, a_3. Každá část a_i má p bitů a je nahrána do registru r_i, $r_4 = 0$. V simulaci jednoho cyklu se nejprve uloží hodnoty registrů do dočasných proměnných a provedením jednotlivých instrukcí nastaveny hodnoty nové. Výstup s-boxu je složen z p-bitových částí y_i takto: $y_0 = r_0$, $y_1 = r_1$, $y_2 = r_3$ a $y_3 = r_4$.

Pro seznam výstupů

2 9 13 13 2 9 13 13 2 9 13 13 13 6 6 6

je posloupnost trojinstrukcí

```
{AND r2, r3 ; MOV r3, r1 ; OR r4, r0}
{XOR r1, r3 ; XOR r4, r4 ; OR r0, r1}
{AND r1, r3 ; NOT r4 ; AND r3, r3}
{OR r2, r3 ; XOR r0, r2 ; AND r4, r1}
{OR r2, r3 ; OR r0, r4 ; OR r3, r0}
{XOR r0, r1 ; NOT r1 ; OR r4, r0}
```

A pro seznam výstupů

15 8 15 8 15 3 15 3 15 8 1 3 15 3 1 8

je posloupnost trojinstrukcí

```
{AND r2, r2 ; AND r3, r1 ; OR r1, r2}
{AND r3, r4 ; AND r2, r0 ; XOR r4, r3}
{XOR r4, r2 ; NOT r0 ; NOT r2}
{AND r1, r4 ; NOT r3 ; AND r0, r2}
{XOR r1, r2 ; NOT r4 ; XOR r0, r1}
{NOP ; OR r3, r3 ; AND r2, r4}
```

Instrukce jsou ve tvaru `INST výstup, vstup`. U NOT je vstup zároveň výstupem k jeho definici se použije první operand. A NOP nemá žádné operandy. Takže se nepotřebné operandy ignorují.

Křížení probíhá na úrovni cyklů. S cykly od rodičů se provede jednobodové křížení. Při mutaci se prochází všemi cykly. V každém je s pravděpodobností $pMut$ vybrána instrukce ke změně. Pokud je instrukce vybrána vybere se rovnoměrným rozložením pravděpodobnosti jedna ze tří možných změn:

1. změna typu instrukce – s rovnoměrným rozložením pravděpodobnosti se vybere jedna z povolených v aktuálním cyklu
2. změna druhého operandu – s rovnoměrným rozložení je vybrán jeden z pěti možných registrů
3. změna prvního operandu – obdobně, ale zde se nesmí vybrat registr, který je již v aktuálním cyklu použit jako první operand v jiné instrukcí

Algoritmus	Binární	Seznam výstupů	CGP	i686
základní GA	vhodné	vhodné	nemožné	nemožné
GP	možné	možné	možné	vhodné
EDA	vhodné	nevhodné	nemožné	nemožné
Random	možné	možné	možné	možné
ParallelRandom	možné	možné	vhodné	možné
VEGA	vhodné	vhodné	vhodné	vhodné
SPEA	vhodné	vhodné	vhodné	vhodné

Tabulka 5.1: Použitelnost reprezentací s-boxu v jednotlivých prohledávacích algoritmech

Tabulka 5.1 ukazuje, u kterého prohledávacího algoritmu je možné určitou reprezentaci použít z hlediska striktní definice těchto algoritmů. Pro acyklický graf hradel není definováno křížení, takže je nutné ho zakázat. Pokud EDA algoritmus pracuje s číselnými alelami, může hledat i číselný seznam, avšak ten statistický model bude zbytečně komplikovaný. Algoritmy VEGA a SPEA vlastně zvládnou to co GP, v této tabulce je zohledněna potřeba multikriteriální optimalizace.

5.4 Použité jazyky a platforma

Ke splnění základních technických požadavků byla zvolena kombinace Python a C++.

Python

Výhodou Pythonu je množství knihoven, které jsou v něm napsány, a snadnost jejich spolupráce. Mezi ně patří například pickle pro jednoduché perzistentní uložení téměř jakéhokoliv objektu, což se dá využít pro průběžné ukládání částečných výsledků experimentů. Knihovna numpy především pro práci s velkými vektory a maticemi se spoustou matematických funkcí. Tyto funkce jsou napsány v jazyce C, a s tímto jazykem umí snadno pracovat. K tomu existuje navíc ještě knihovna ctypes, která umožňuje pracovat s datovými typy jazyka C, ve funkcích které jsou součástí dynamicky linkované knihovny do Pythonu. Pro zápis experimentů je výhodou interaktivita zadávání příkazů, tzv. „ipython“ interpret, který ve spojení s numpy a knihovnou na kreslení grafů je alternativou Matlabu. Nevýhodou je pomalé provádění kódu, což se musí řešit implementací časově kritických funkcí v C či C++.

C++

C++ dokáže vygenerovat rychlý kód pro mnoho procesorů a platforem. Napsaný kód je při dodržení určitých pravidel přenositelný. Je vhodný pro implementaci výpočtů, které mají běžet rychle. Také pro něj existuje mnoho knihoven s vyřešenými algoritmy. Nevýhodou jsou komplikované deklarace a definice, které brzdí popisy experimentů.

Galib

Galib je knihovna původně pro genetické algoritmy, svým obecným návrhem však zvládne i obecnější třídu např. i genetické programování. Je určená pro C++. Ostatní evoluční algoritmy je nutné doimplementovat. Knihovna Galib nejlépe vyhovovala předchozím požadavkům. Nejdůležitější byla obecnost kvůli srovnávání. Tato knihovna [21] umožňuje definovat vlastní prohledávací algoritmus, vlastní operátory či vlastní reprezentaci genomu atd. Při programování se pracuje z dvěma hlavními třídami reprezentujícími genom a druhá genetický algoritmus. Každá instance genomu reprezentuje jedno řešení problému. Genetický algoritmus používá objektivní funkci (definovanou uživatelem) k určení kritéria přežití genomu (fitness). Běžné genetické operátory (selekce, mutace, křížení) jsou vestavěné. Během evoluce sbírá statistiky. Evaluátory a operátory lze měnit za běhu evoluce. Na druhou stranu její knihovní funkce fungují správně jen při sekvenčním provádění, pokud je třeba paralelizovat nutné upravit její kód. Mnoho metod není virtuálních a při použití vlastních komponent je nutné navíc i přetypovávat[1].

OpenMP

OpenMP je specifikace pro paralelizaci algoritmů pomocí direktiv, které překladače přeloží podle cílové platformy a procesoru. OpenMP je navržená pro různé jazyky, např. v C++ se direktivy uvádí pomocí `#pragma omp`. Překladače GCC implementují některé z jejích direktiv pro jazyky C/C++ pomocí POSIX threadů. Tyto direktivy jsou navržené tak, aby na jednoprocesorových strojích bylo možné přeložit zdrojový kód bez přepínače, zapínajícího OpenMP, tím způsobem, že tyto direktivy budou jednoduše ignorovány. Což je jedním ze znaků přenositelnosti, kterou tato specifikace má. Největší uplatnění tyto direktivy mají především při paralelizaci stávajícího sekvenčního kódu. OpenMP se automaticky snaží využít všech dostupných procesorů či jader pro paralelní provedení kódu označených direktivami určitým způsobem (paralelní cyklus, task), že by mohli být provedeny paralelně. Programátor mnohdy musí pomocí těchto direktiv označit sdílené a lokální proměnné. Další direktivy jako bariéra a kritická sekce, umožňují provádět synchronizaci mezi vlákny.

Diskuse dalších variant návrhu programu

Jak bylo naznačeno výše, celý projekt by bylo možné realizovat pouze v Pythonu za použití matematického systému sagemath, který již obsahuje funkce pro práci s s-boxy a výpočty jejich kritérií. Implementace evolučních algoritmů by však nebyla efektivní z důvodů vysoké úrovně jazyka. Náročné výpočty by bylo možné provádět i v GPU. Nejprve je však dobré mít implementaci pro CPU pro srovnání, zdali se vůbec implementace v GPU vyplatí.

[1] To nabádají i v příkladech

Kapitola 6

Popis implementace

Ve fázi návrhu byla vybrána kombinace jazyků C++ a Python. V C++ byly následně s pomocí knihovny GAlib naimplementovány časově náročnější operace, tj. průběh evoluce, reprezentace s-boxů, výpočty kritérií bezpečnosti z výstupů s-boxů. K evolučním experimentům byly zvoleny právě i algoritmy, které nejsou součástí knihovny Galib. Dále bylo naimplementováno rozhraní mezi Pythonem a C++, aby mohly být experimenty spouštěny Pythonu a tam i zpracovány výsledné statistiky.

Třídy v C++ se dělí mezi ty, jenž se týkají reprezentací s-boxů, a ty, které provádí prohledávání. Jakoukoliv reprezentaci s-boxu je možné použít v kterémkoliv prohledávacím algoritmu kromě algoritmu EDA, jenž podle definice pracuje s číselnými alelami, v této implementaci pouze s binárními. V následujícím textu jsou vybrány zajímavé součásti implementace.

6.1 Implementace evolučních algoritmů

Nejprve byly naimplementovány chybějící prohledávací algoritmy. Všechny prohledávací algoritmy jsou zděděny od bázové třídy `SboxSearchAlgorithmBase`, která dědí od `GASimpleGA` z Galibu. Tímto způsobem bylo možno použít obecnější rozhraní k prohledávacím algoritmům. Většina logiky následující algoritmů byla implementována v předefinované metodě `step()`, která provede jednu generaci evolučního procesu, z nadtřídy `GASimpleGA` v Galibu. Následují text popisuje implementované metody a pomocné objekty, které jsou během provedení jedné generace použity specializovaně pro konkrétní algoritmus.

Eda algoritmy

Třída EDA algoritmu rozšiřuje základní třídu o vlastnost model, ten může být dvojího typu daném třídou objektu. Bázová třída `EdaModel` implementuje UMDA statistický model, její podtřída `BmdaModel` totiž též používá prosté pravděpodobnosti výskytů alel k jeho výpočtům. Ve třídě `EstimationOfDistributionAlgorithm` je definována obecná logika EDA algoritmu a odpovědnost za vytvoření statistického modelu a vygenerování nových jedinců se přenese právě na instanci třídy `EdaModel`.

Ve třídě `EdaModel` byly naimplementovány dvě hlavní polymorfní metody pojmenované `learnStructure`, která jednoduše projde všemi jedinci a zaznamená počty výskytů jedničkových bitů v poli o délce binárního chromozómu, a `sampleModel`, která toto pole používá ke generování jedinců s pravděpodobností $P(genome[i] = 1) = count[i]/popSize$.

Třída `BmdaModel` používá pro vytvoření statistického modelu třídu `ContingentTable` a k jeho uložení les[1] vytvořený pomocí instancí tříd `BmdaNode`. V metodě `learnStructure` nejprve spočítá kontingenční tabulky závislostí mezi lokusy. Kontingenční tabulky, u nichž vyjde hypotéza „závisí“ χ^2-testu, jsou seřazeny podle hodnoty tohoto χ^2-testu. Pokud není jsou všechny lokusy nezávislé, algoritmus skončí pouze s vypočteným UMDA modelem. V opačném případě jsou ze seřazených kontingenčních tabulek vygenerovány stromy závislostí[2]. V metodě se podle typu aktuálního uloženého modelu vybere, buď generování jedinců v nadtřídě, nebo se při generování každého jedince projde lesem a hodnoty na jednotlivých lokusech jsou spočítány na základě prošlých podmíněných pravděpodobností v uzlech na větvích stromů.

Nadtřída multikriteriálních algoritmů[3] se stará o množinu požadovaných kritérií. Dále obsahuje rutiny, které používají oba multikriteriální algoritmy. Mezi ně patří například rozhodnutí o dominanci a výpočet nedominované elity metodou průběžně aktualizovaného přístupu.

Algoritmus VEGA

V implementaci algoritmu VEGA (viz podkapitola 4.4.2) ve třídě `VegaAlgorithm` jsou v metodě `mainPopulationToSegments` rozděleny jedinci do podpopulací, dle počtu kritérií. V každé podpopulaci je jedincům nastavena její přidělená kriteriální funkce, případně jsou zneplatněno hodnocení těm, kteří byly v minulé generaci vyhodnoceni jiným kritériem. V metodě `segmentsToMainPopulation` jsou jedinci v rámci podpopulace vyhodnoceni. Skóre je nastaveno každému jako podíl hodnoty vypočteného kritéria jedince a součtu všech. Poté jsou podpopulace sloučeny do jedné.

Algoritmus SPEA

Algoritmus SPEA (viz podkapitola 4.4.3) ve třídě `SpeaAlgorithm` je díky výpočtu nedominované sady řešení a shlukování výpočetně nejsložitější. Výpočet sil v metodě nazvané `computePowers` nejprve všem jedincům z normální populace nastaví sílu na jedna. Poté prochází všemi jedinci v elitě E a v každém kroku pro každého jedince E_i provede:

1. zaznamená všechny jedince D^i z normální populace, jež jsou dominováni jedincem E_i
2. jejich počet $|D^i|$ využije ke stanovení síly jedince E_i
3. všem dominovaným jedincům $D^i{}_j$ je připočtena síla jedince E_i

Pro implementaci shlukování (uvedeného na str. 28) slouží ve třídě `SpeaAlgorithm` tyto datové typy:

```
typedef std::vector<Sbox*> Cluster;
class ClusterPair : public std::pair<Cluster*, Cluster*>;
typedef std::vector<ClusterPair* > ClusterPairVector;
typedef std::multimap<double, ClusterPair*> ClusterDistances;
typedef std::map<Cluster*,  ClusterPairVector> PairsForCluster;
```

[1] množina stromů
[2] Algoritmus vytvoření tohoto lesa je uveden např. v [17]
[3] ve zdrojových kódech `MulticriterialGeneticAlgorithm`

Clusterem je tedy seznam ukazatelů na jedince. Třída `ClusterPair` je pomocná pro výpočet vzdáleností mezi clustery. V ní je kvůli správě paměti příznak zda ještě platí. Pomocný typ `ClusterPairVector` slouží pro uložení platných i neplatných párů pro `ClusterDistances`, který uchovává páry seřazené podle vzdáleností mezi clustery v páru. Klíč typu double slouží k průběžnému seřazení, ne k přístupu k prvkům pomocí typu double. A pomocný typ `PairsForCluster` eviduje všechny páry, ve kterých je cluster obsažen. Celý algoritmus shlukování v metodě `clustering` s použitím těchto typů lze popsat:

1. inicializace *pairsForCluster* typu `PairsForCluster`
   a *clusterDistances* typu `ClusterDistances`
2. nejprve se projde celou elitou, a z každého jedince je vytvořen cluster a přidán mezi ostatní
3. procházení všemi *clusterDistances* od nejmenší vzdálenosti, dokud počet clusterů je větší než požadovaná velikost elity, neplatné páry jsou přeskočeny
   (a) přidání nového clusteru obsahujícího oba jedince ze současného páru
   (b) odebrání obou starých clusterů
4. generování nové elity z jedinců nejbližších středu clusteru

Při přidání clusteru v metodě `addCluster` se spočítají vzdálenosti nového clusteru se všemi existujícími clustery a uloží mezi *clusterDistances*. Každý nový pár je také přidán do *pairsForCluster* k TODO obema klíčům.

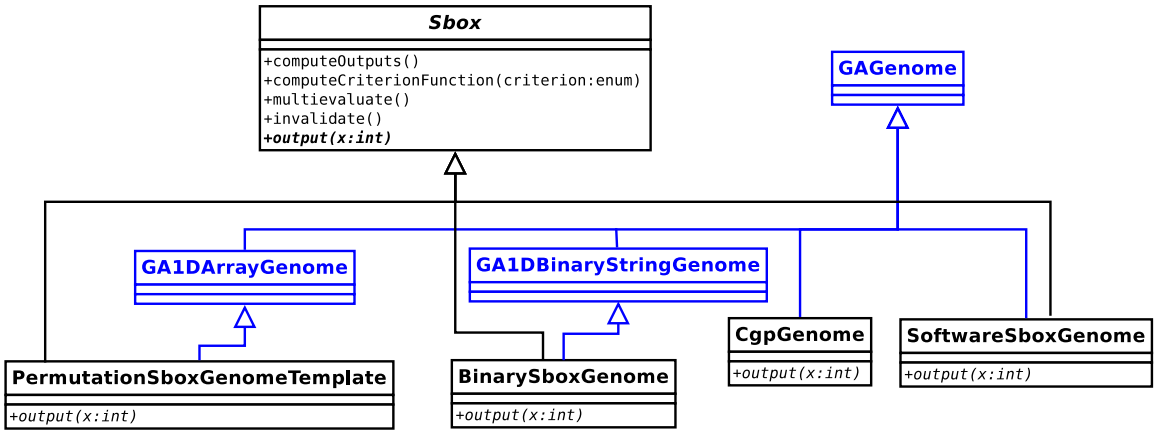
Odebrání clusteru v metodě `removeCluster`: Při odebrání clusteru se zneplatní páry, ve kterých je cluster obsažen a odstraní se jejich seznam pro aktuální cluster v *pairsForCluster*.

Kartézské genetické programování, náhodné prohledávání

Kartézské genetické programování bylo rozděleno na tzv. „Paralelní náhodné prohledávání“ a acyklický graf hradel, jehož implementace je popsána dále, protože v této práci je považována za reprezentaci s-boxu. Při implementaci jak náhodného, tak paralelního náhodného prohledávání bylo zděděno od stejné bázové třídy jako ostatní evoluční algoritmy, z důvodu snadného porovnání díky sběru statistik, jenž provádí Galib. Proto využívají ke generování dalších řešení i mutaci genomu.

6.2 Implementace reprezentací s-boxů

Pro implementaci reprezentací s-boxů bylo využito též specializovaných tříd obsažených v Galibu. Tedy nejen odvozením od `GAGenome`. Na obrázku 6.1 je znázorněn diagram tříd reprezentací a jejich odvození z Galibu, které jsou označeny modře. To samou barvou je označena i větev dědičnosti od `Genome`. Černou jsou označeny vlastní třídy reprezentací substitučních boxů. Tou samou barvou je znázorněno dědění od bázové třídy s-boxu. Z obrázku je patrné, že byla použita vícenásobná dědičnost. Původně byla třída `Sbox` navržena jako čistě virtuální, ale některé funkčnosti, které jsou společné pro všechny s-boxy bylo přehlednější napsat do ní než někam mimo třídu. Mezi tyto metody patří například uložení všech možných vstupů do pole, rozhraní k výpočtu jednotlivých kritérií či provedení multievaluace. Naproti tomu třída `GAGenome` pracuje pouze s jedním kritériem. V ní jsou



Obrázek 6.1: Diagram tříd reprezentací s-boxů s vyznačením napojení do Galibu

definovány jako vlastnost callbacky pro inicializaci, mutaci a křížení genomu. V podtřídách jsou tyto callbacky implementovány a nastaveny.

Zatímco v `PermutationSboxTemplate` jsou výstupem geny na odpovídačích pozicích, u ostatních reprezentací je nutné provést výpočet. Proto je v bázové třídě možnost tyto výsledky (i kritérií) uložit a případně zneplatnit. Kód `CgpGenomu` byl inspirován implementací [20]. Výpočet výstupů reprezentace i686 je uveden v 5.3.4.

6.3 Naimplementovaná kritéria a jejich vyhodnocení

V `multievaluate` (též na obr. 6.1) se napřed uloží výstupy s-boxu do pomocného slovníku a z něho jsou spočítány hodnoty kritérií a poté uloženy v objektu, aby se nemusely počítat znovu. Multikriteriálních algoritmech je tedy nutné navíc po mutaci jedince zavolat metodu `invalidate`, aby se uložené hodnoty kritérií zneplatnily. Některá vypočtená kritéria potřebují před předáním evolučnímu algoritmu upravit, zde jsou uvedeny jejich úpravy:

bent skóre počet w jejichž $|G(w)| = 1$

řád SAC k řádu přičtena jednička, protože Galib nebere záporné hodnoty

bent + SAC pokud je bent skóre rovno 2^n, tak se přičte řád SAC

diferencial probability minimalizace provedená pomocí: $-\log_2(DP_{\max})$

linear bias probability minimalizace provedená pomocí: $-\log_2(LP_{\max})$

bijektivní skóre obdobné bent skóre

Výstupy jsou počítány celočíselnou aritmetikou. Pro Galib, který pracuje s fitness v pohyblivé desetinné čárce se převede až na konec po vypočtení celého kritéria. Výstupy některých kritérií, jak při testování, tak závěrečné výsledky, srovnány s matematickým systémem Sagemath a jeho funkcemi pro s-box[4]. Které nebylo vhodné použít přímo, neboť evoluce z důvodu rychlosti probíhá v C++ a neobsahuje všechna kritéria, kterými se práce zabývá.

[4] `<http://www.sagemath.org/doc/reference/sage/crypto/mq/sbox.html.>`

6.4 Rozhraní komunikace z Pythonem

Původní prototypové schéma na obrázku 5.1 bylo změněno tím, že se třídy pro evoluční algoritmy přesunuly do C++. Pythonu proto zůstává pouze rozhraní pro jejich nadtřídu. Toto rozhraní umožňuje v konstruktoru specifikovat, který algoritmus a genom se má pro prohledávání použít. Při vytváření tohoto rozhraní byla co nejvíce použita knihovna ctypes, která umožňuje spouštět funkce v jazyce C[5] umístěné v dynamicky linkované knihovně. Primitivní datové typy jsou většinou převáděny do Pythonu automaticky. Někdy je nutné v Pythonu specifikovat typ argumentu či návratové hodnoty funkce, ale to není problém, pokud se céčkové funkce příliš nemění. Knihovna ctypes podporuje i struktury jazyka C. Předávání velkých polí dat zajišťují funkce knihovny numpy, která je použita ke zpracování výsledků při experimentech. Pro předávaní řetězců nakonec použito Python C API.

V příloze B je rozhraní rozvedeno trochu podrobněji z pohledu jeho používání.

6.5 Diskuze možností paralelizace výpočtů

Aby se paralelizace vyplatila, nejprve se provedla profilace kódu, při které se našly části kódu, které trvaly nejdéle. Co již nešlo více zrychlit sekvenčně, vybralo se k paralelizaci. Pro orientační odhad, zda se paralelizace vyplatila, bylo využito unixového nástroje `time`, který ukazuje spotřebovaný strojový a reálný čas programu. Strojový čas pro více procesorů (jader) je ve výstupu tohoto nástroje součtem ze všech použitých procesorů. Strojový čas paralelního běhu a sekvenčního běhu museli být přibližně stejné.

Paralelizovat lze na úrovni běhů evoluce, na každém jádře/procesoru pojede jeden běh evoluce. Daly by se řešit i případy, kdy jeden běh skončí dříve a ostatní běhy (většinou poslední) by mohly využít volného času nezaměstnaného jádra, ale tato situace je spíše výjimečná pro případy, kdy bude program použit. Běhy mezi sebou takřka nesdílejí žádná data, a tudíž se paralelizace na víceprocesorovém systému vyplatí. Jediné co sdílejí a bylo třeba ošetřit je stav pseudonáhodného generátoru.

K paralelizaci bylo využito direktiv OpenMP, které jsou obecnější, přenositelnější a mají jednodušší a přehlednější zápis než např. vlákna POSIX. Kód pro POSIX vlákna tam vygeneruje překladač GCC, ostatní překladače tam mohou vygenerovat jiný kód specifický pro určitou architekturu procesoru.

Právě díky OpenMP bylo možné označit jako privátní[6] proměnné, jenž ukládají stav pseudonáhodného generátoru v Galibu:

```
#pragma omp threadprivate(seed, iseed, idum2, iy, iv, idum)
```

Tato úprava nezasáhla do rozhraní galibu a je možné při překladu pro sekvenční provádění použít původní Galib.

Jinou možnou paralelizací by bylo paralelně počítat objektivní funkce jedinců uvnitř populace. Třída `GAPopulation` uchovávající seznam jedinců v populaci by musela být naimplementovaná bezpečně pro použití více vlákny, tj. dodané synchronizace. Uvažovanou pa-

[5] funkce C++ se dají jednoduše deklarovat v `extern "C"` bloku

[6] privátní z pohledu sdílení více procesory

ralelizací se rozumí následující kód:

```
// definice evaluátoru populace
void OmpPopulationEvaluator(GAPopulation & p) {
const int size = p.size();
#pragma omp parallel for shared(p) schedule(guided)
for(int i=0; i< size; i++)
p.individual(i).evaluate();
}

// nastavení evaluátoru např. v tovární metodě objektu GAGeneticAlgorithm
GAPopulation initPopulation(ga.population());
initPopulation.evaluator(OmpPopulationEvaluator);
ga.population(initPopulation);
```

I bez všech potřebných synchronizací stejně nakonec trvalo toto paralelní řešení delší strojový čas než sekvenční řešení, takže by se touto paralelizací moc pro většinu kritérií nezískalo, ledaže by byl výpočet některého kritéria složitější např. $LP_{\max}$.

Kapitola 7

Provedené experimenty a zhodnocení výsledku

Cílem experimentů bylo zjistit zda jsou Evoluční algoritmy vhodné pro pro hledání zvolených velikostí s-boxů. Za tím účelem bylo porovnáno s náhodným prohledáváním. Porovnání běhů dle statistik z Galibu [21] na několika bězích. Experimenty byly naskriptovány pomocí Pythonu. Byly vybrány na základě testovaní programu, kde proběhly jednoběhové zkoušky velikosti populace. Pro dlouhodobější hledání kvalitnějších řešení byly nejprve stanoveny parametry evoluce, zejména pravděpodobnosti mutace a křížení. Poté se tyto parametry užily pro náročnější experimenty.

7.1 Stanovení pravděpodobností mutace a křížení pro jednotlivá kritéria

Nejprve byly stanoveny parametry křížení a mutace pro jednokriteriální optimalizaci pro každé kritérium. Při nich se většinou dospělo k mezním hodnotám. Byly nalezeny nejlepší pravděpodobnosti mutace a křížení pro každé kritérium při pevné velikosti populace 100, počtu generací 1000 a počtu běhů 100. Proti tomu bylo postaveno stejně početné náhodné prohledávání. S-box byl zvolen středně složitý se 4 vstupy a 4 výstupy (tedy byl typu 4×4). Nejprve byly na intervalu $\langle 0, 1)$ zkoušeny všechny možné dvojice pravděpodobností: pro mutaci $0, 0.05, 0.1, 0.2, 0.4, 0.6, 0.8$; pro křížení $0, 0.2, 0.4, 0.6, 0.8$. Protože výsledky u některých kritérií ve vyšších hodnotách byly obdobné a málo slibnější, byl zvolen ještě užší interval $\langle 0, 0.2)$. Zkoušeny byly dvojice pravděpodobností $0, 0.04, 0.08, 0.12, 0.16$ jak u křížení tak u mutace. U každého běhu bylo zaznamenáno uloženo 100 nejlepších jedinců z průběhu celé evoluce a všichni jedinci ze všech běhů pro jednu dvojici (p_{Mut}, p_{Cross}) byly sloučeni dohromady a nad nimy byly spočteny statistiky. Výsledky zaneseny do tabulek, které jsou seřazeny od nejhoršího po nejlepší podle řadícího klíče (maximum, medián, průměr + odchylka, průměr, minimum). Pro rozhodnutí byl proveden orientační dvouvýběrový t-test výsledků náhodného prohledávání a nejlepší kombinace parametrů pro genetický algoritmus.

Bent skóre

V tabulce 7.1 je uvedena pouze varianta bez křížení a mutace, ostatní kombinace parametrů dávaly stejné statistiky a to takové, že se shodovaly i se statistikami náhodného

Parametry	maximum	medián	std. odchylka	průměr	minimum
(0, 0)	32	31	4.07067	29.1836	7
ostatní kombinace	32	32	0	32	32

Tabulka 7.1: Výsledky pro celý interval a kritérium bent skóre + řád SAC

Parametry	maximum	medián	std. odchylka	průměr	minimum
(0, 0)	2	0.830075	0.380956	0.910228	0
ostatní kombinace	2	2	0	2	2
(0, 0.2)	2	2	0.452339	1.7859	0.830075

Tabulka 7.2: Výsledky pro celý interval pro kritérium $LP_{\max}$

prohledávání, u kterého vyšlo maximum 32, medián 32, odchylka 0, průměr 32, minimum 32. Tj. všichni nejlepší členové dosáhli maximálního možného bent skóre pro bijektivní s-box. Pro bijektivní by to bylo 64. U tohoto kritéria tedy nezáleží na parametrech pro velikost populace 100 a počet generací 1000. Výsledky se shodují s náhodným prohledáváním. Na základě toho se kritérium dále pro zúžení intervalu zdálo nezajímavé.

Linear probability

V tabulce 7.2 vychází jako nejvyšší hodnota 2. Maximální odchylka pravděpodobnosti lineárních výrazů $2^{-2} = 0.25$. Vyzkoušeny hodnoty v užším intervalu a zaznamenány do tabulky 7.3. Náhodné prohledávání mělo: maximum 2, medián 2, odchylka 0.365039, průměr 1.87213, minimum 0.830075. Náhodné prohledávání má lepší průměr.

Diferential probability

Výsledky pro celý interval v tabulce 7.4 jsou obdobné výsledkům kritéria $LP_{\max}$. Náhodné prohledávání mělo: maximum 2, medián 2, odchylku 0.29122, průměr 1.73475 a minimum 1.41504. Pro zúžený interval tabulka 7.5 potvrzuje, že pokud je vypnutá mutace a křížení je menší než 0.2 nalezne více slabších výsledků. V jiném případě je lepší než náhodné prohledávání.

Parametry	maximum	medián	std. odchylka	průměr	minimum
(0, 0)	2	0.830075	0.373374	0.905579	0
(0, 0.04)	2	0.830075	0.511175	1.13063	0.830075
(0, 0.08)	2	0.830075	0.577091	1.31957	0.830075
ostatní kombinace	2	2	0	2	2
(0, 0.12)	2	2	0.584362	1.44218	0.830075
(0, 0.16)	2	2	0.549793	1.61486	0.830075

Tabulka 7.3: Výsledky pro zúžený interval pro kritérium $LP_{\max}$

Parametry	maximum	medián	std. odchylka	průměr	minimum
(0, 0)	2	1.41504	0.280306	1.28179	0.415038
ostatní kombinace	2	2	0	2	2
(0, 0.2)	2	2	0.291124	1.73563	1.41504

Tabulka 7.4: Výsledky pro celý interval pro kritérium $DP_{\max}$

Parametry	maximum	medián	std. odchylka	průměr	minimum
(0, 0)	2	1.41504	0.283811	1.28443	0.415038
(0, 0.04)	2	1.41504	0.199666	1.48975	1
(0, 0.08)	2	1.41504	0.23421	1.53232	1.41504
(0, 0.12)	2	1.41504	0.264779	1.58327	1.41504
(0, 0.16)	2	1.41504	0.284352	1.63902	1.41504
ostatní kombinace	2	2	0	2	2

Tabulka 7.5: Výsledky pro zúžený interval pro kritérium $DP_{\max}$

SAC

Samotný řád SAC má malou vypovídací hodnotu o kvalitě řešení, přesto genetický algoritmus nalezl nejvyšší možný řád. Hodnoty v tabulce C.1 jsou zvýšené o jedničku, aby fitness nebyla záporná[1]. Z tabulky vyplývá, že takřka nezáleží na pravděpodobnosti mutace a křížení. Rozdíly mezi jednotlivými dvojicemi parametrů jsou nevýznamné. Naproti tomu náhodné prohledávání dospělo k maximu 1, mediánu 0, odchylce 0.249148, průměru 0.0665 a minimu 0. Tj. našlo jen pár jedinců, kteří vůbec splňují SAC 0. řádu.

Branching faktor

Genetický algoritmus s faktorem větvení jako objektivní funkcí v tabulce C.2 dospělo k obdobným statistikám jako náhodné prohledávání, které dospělo k maximu 3, mediánu 2, odchylce 0.0199968, průměru 2.0004 a minimu 2. Pro zúžení byl nezajímavý.

Stupeň algebraického polynomu

Všechny kombinace parametrů v tabulce 7.6 až na variantu bez křížení a mutace se ve výsledcích shodují. Stálo za to, prověřit to i v zúženém intervalu v tabulce 7.7, který zjištění potvrzuje. Náhodné prohledávání dospělo též maximu 3, mediánu 3, odchylce 0, průměru 3 a minimu 3. Pro bijektivní funkce je to stupeň polynomu nejvyšší možný.

[1] Což vyžaduje Galib

Parametry	maximum	medián	std. odchylka	průměr	minimum
ostatní kombinace	3	3	0	3	3
(0, 0)	3	3	0.457029	2.748	1

Tabulka 7.6: Výsledky pro celý interval pro kritérium stupeň algebraického polynomu

Parametry	maximum	medián	std. odchylka	průměr	minimum
ostatní kombinace	3	3	0	3	3
(0, 0)	3	3	0.455533	2.7484	1

Tabulka 7.7: Výsledky pro zúžený interval pro kritérium stupeň algebraického polynomu

Kritérium	pMut	pCross	co je lepší
bent skóre	0.1	0.2	Random
$LP_{\max}$	0.0	0.16	Random
$DP_{\max}$	0.0	0.2	GA
SAC	0.1	0.2	GA
faktor větvení	0.1	0.2	Random
stupeň polynomu	0	0	Random

Tabulka 7.8: Shrnutí Genetického algoritmu vs. náhodného prohledávání na permutaci

V tabulce 7.8 jsou uvedeny nejlepší možné parametry genetického algoritmu. A stanovení, který algoritmus byl pro provedené evoluce lepší. Tam kde se hodnoty statistik shodovaly bylo zvoleno náhodné prohledávání kvůli své jednodušší algoritmické složitosti. Na statistikách evoluce různých jsou vidět jak omezení kritérií, tak i podobný tvar vzorců, kterými jsou k kritéria počítána. To jaký je algoritmus lepší záleží na použitém kritériu. V některých případech má evoluci smysl provést.

7.2 Porovnání výsledků EDA algoritmů s Genetickým algoritmem

Dalším provedeným experimentem bylo ověření na binárním chromozómu, zda EDA algoritmy nacházejí lepší, horší či stejné výsledky v porovnání s nejlepšími parametry předchozího experimentu. Byly provedeny stejné výpočty jako v předešlém experimentu, tentokrát však pro nejlepší parametry Genetického algoritmu vůči algoritmům BMDA a UMDA a náhodnému prohledávání. Narozdíl od předchozího však nebyla hledána bijektivní funkce.

Pro vizualizaci porovnání byly zvoleny tabulky kratší, které ukazují kritéria v jednotlivých algoritmech, a krabicové grafy, které porovnávají algoritmy mezi sebou. Krabicový graf znázorňuje rozložení stejně početných skupin hodnot. Horní a dolní strana modrého obdélníka znázorňuje umístění kvartilů. Prostřední červená vodorovná úsečka přestavuje medián. Nahoru a dolů od obdélníka vede tzv. vous [2]. Horní resp. dolní vous znázorňuje nejvyšší resp. nejnižší hodnotu, na kterou má být brán zřetel. Křížky jsou označeny odlehlé hodnoty. V tabulce 7.9 jsou statistiky pro Genetický algoritmus. Pro SAC se nepodařilo najít nejvyšší řád. Pro Bent skóre a stupeň polynomu již neplatí omezení bijektivitou. Jinak jsou výsledky podobné hledání permutace z předcházejícího experimentu. UMDA algoritmus v tabulce 7.10 neměl žádný problém z $DP_{\max}$ a stupněm polynomu, za to SAC podobně jako u náhodného prohledávání našel jen pár jedinců, kteří splňovali nejnižší stupeň SAC. BDMA v tabulce 7.11 dopadl obdobně, trochu lepší výsledek měl u bent skóre. Náhodné prohledávání (tab. 7.12) mělo stejné výsledky v maximech a mediánech jako BMDA.

[2] z anglického whisker

Kritérium	maximum	medián	std. odchylka	průměr	minimum
$LP_{\max}$	2	1.35614	0.221604	1.44462	1.35614
bent skóre	64	40	3.51839	42.0224	40
$DP_{\max}$	2	2	0.0628992	1.99316	1.41504
SAC	2	1	0.0141407	1.0002	1
BF	2	2	0	2	2
stupeň polynomu	4	3	0.421644	2.9443	0

Tabulka 7.9: Porovnání EDA a GA na binárním chromozómu: kritéria pro GA

Kritérium	maximum	medián	std. odchylka	průměr	minimum
$LP_{\max}$	2	1.35614	0.141776	1.38912	1.35614
bent skóre	47	31	2.23823	31.988	30
$DP_{\max}$	2	2	0	2	2
SAC	1	0	0.0990082	0.0099	0
BF	2	2	0.498952	1.5322	1
stupeň polynomu	4	4	0	4	4

Tabulka 7.10: Porovnání EDA a GA na binárním chromozómu: kritéria pro UMDA

Krabicový graf na obrázku 7.1 porovnává jednotlivé algoritmy pro kritérium bent skóre. V tomto kritériu je zcela jasným vítězem Genetický algoritmus. Náhodné prohledávání je v tomto kritériu srovnatelné s EDA algoritmy. V kritériích $LP_{\max}$ a $DP_{\max}$ (obr. C.2 a C.3) nejsou mezi algoritmy rozdíly. U lavinového efektu, kde má genetický algoritmus svůj průměr i medián, jsou u ostatních odlehlé maximální hodnoty (obr. C.4). Ve faktoru větvení (obr. C.5) je o něco horší UMDA, ale jinak se dospělo ke stejným výsledkům. Za to ve stupni polynomu (obr. C.6) je genetický algoritmus nejhorší. Jen v tomto kritériu vycházejí EDA algoritmy o něco málo lépe než náhodné prohledávání, leč střední hodnota je shodná.

Celkově tedy EDA algoritmy dopadly nejhůř, náhodné prohledávání nalezne obdobné výsledky. Původní motivací pro nasazení EDA algoritmů byly tvary MOSAC Booleovských funkcí, které mají určité pravidelnosti [3]. Dvouvstupová booleovská funkce (půl-bajt) splňující MOSAC obsahuje 3 stejné bity, čtvrtý se liší. Třívstupová (bajt) se skládá z dvouvstupových, tak že pozice lišícího se bitu v půl-bajtu jsou symetrické podle osy vedoucí středem bajtu. U čtyřstup nesmí být dva vedlejší bajty symetrické. Otázkou by bylo zda-li se z toho

[3] viz tabulka 3.1, větší funkce v [8]

Kritérium	maximum	medián	std. odchylka	průměr	minimum
$LP_{\max}$	2	1.35614	0.234303	1.45737	1.35614
bent skóre	48	32	2.98262	33.5576	31
$DP_{\max}$	2	2	0	2	2
SAC	1	0	0.196198	0.0401	0
BF	2	2	0	2	2
stupeň polynomu	4	4	0	4	4

Tabulka 7.11: Porovnání EDA a GA na binárním chromozómu: kritéria pro BMDA

Kritérium	maximum	medián	std. odchylka	průměr	minimum
$LP_{\max}$	2	1.35614	0.184163	1.4141	1.35614
bent skóre	48	32	4.16932	30.4523	24
$DP_{\max}$	2	2	0	2	2
SAC	1	0	0.205328	0.0441	0
BF	2	2	0	2	2
stupeň polynomu	4	4	0.478596	3.6447	3

Tabulka 7.12: Porovnání EDA a GA na binárním chromozómu: kritéria pro Random

Kritérium	pMut	maximum	medián	std. odchylka	průměr	minimum
$LP_{\max}$	0.01	2	2	0	2	2
$DP_{\max}$	0.4	3	2	0.139994	2.02	2
bent skóre	0.01	64	64	0	64	64
SAC	0.01	3	3	1.22189	2.37	0
BF	0.9	4	3	0.511454	2.72	2
stupeň polynomu	0.01	4	4	0	4	4

Tabulka 7.13: Nejlepší pravděpodobnosti mutace pro paralelní náhodné programování nad reprezentací CGP

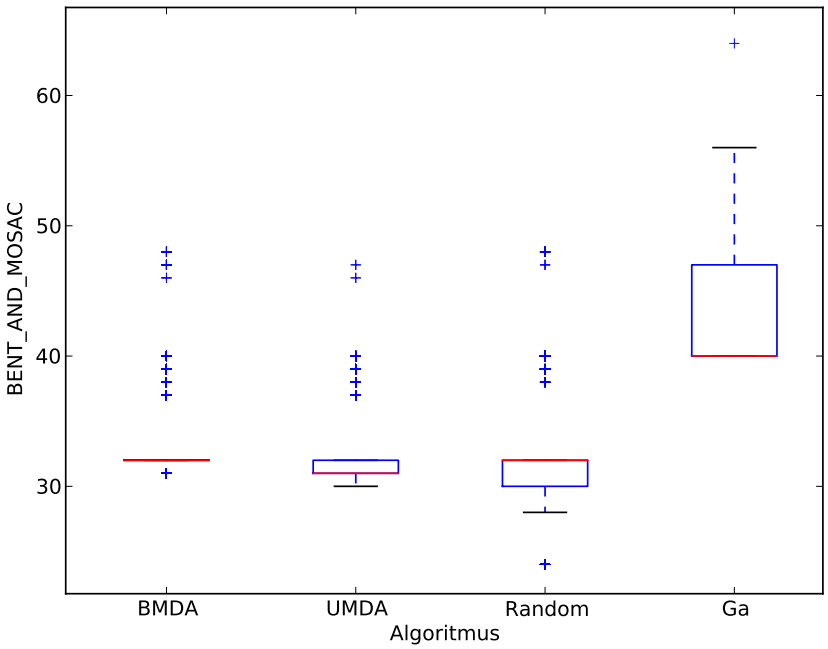
dá vytvořit nějaký pravděpodobnostní model závislostí mezi bity, který by byl použitelný v evoluci pomocí EDA. Proti tomu stojí myšlenka, že statistický model hledající závislosti mezi bity má pro bezpečnost opačný význam, právě, že by bity by měly být vzájemně nezávislé, proto ani nebyl implementován složitější pravděpodobnostní model pro algoritmus BOA.

7.3 Parametry paralelního náhodného prohledávání

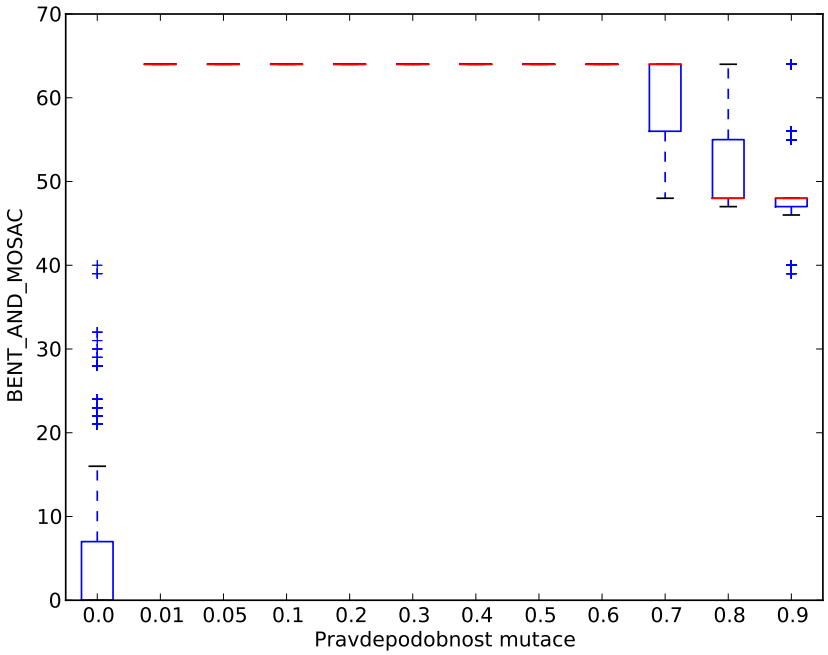
Pro reprezentaci s-boxu jako acyklický graf (Kartézské genetické programování) není definováno křížení. Srovnáno bylo s reprezentací i686 u které též nebylo prováděno křížení. Byly použity dva algoritmy: náhodné prohledávání a paralelní náhodné prohledávání. Pro paralelní náhodné prohledávání vyzkoušeny byly tyto hodnoty mutace: 0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 a 0.9.

Chromozóm pro Kartézské genetické programování zvolen se čtyřmi vstupy, čtyřmi výstupy, čtyřmi sloupci, třemi řádky a s maximálním počtem mutací 6. V tabulce 7.13 jsou shrnuty až výsledné hodnoty s nejlepšími mutacemi. Hodnoty pro jednotlivá kritéria pro každou hodnotu mutace jsou zakreslena do grafů na obrázcích C.7 až C.12. Z nich je patrné, že počet generací je tak velký, že velikost mutace nehraje moc roli. Pro každé kritérium vycházelo lépe paralelní náhodné prohledávání, takže tzv. „paralelnost“[4] prokázala svůj smysl. Pro kritérium $DP_{\max}$ dokonce nalezlo lepší výsledek než předcházející algoritmy. To samé se dá říci i o faktoru větvení.

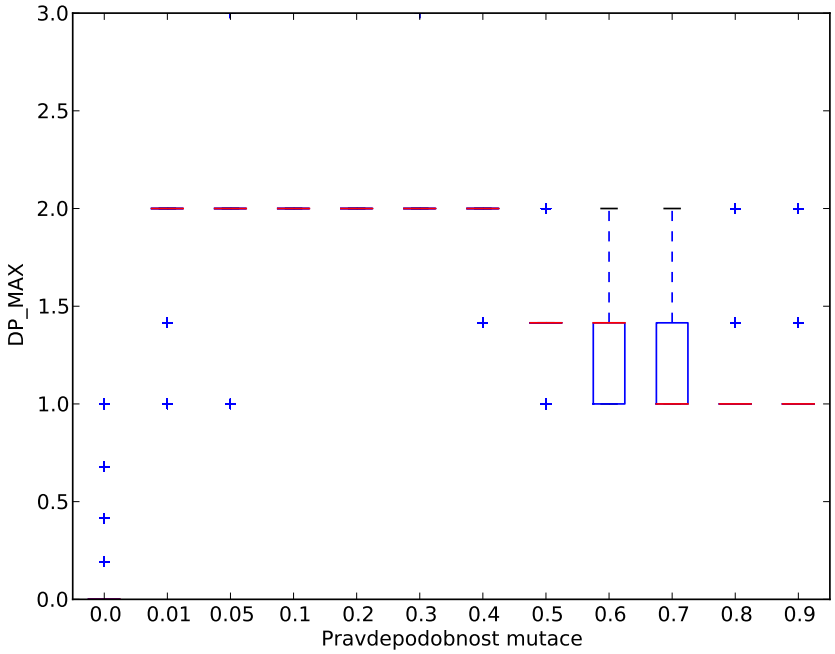
[4] ve smyslu, že je vyhodnoceno více jedinců v populaci



Obrázek 7.1: Porovnání EDA algoritmů na binárním reprezentaci s-boxu, kritérium bent skóre



Obrázek 7.2: Parametry paralelního náhodného prohledávání,bent skóre, reprezentace Luffa



Obrázek 7.3: Parametry paralelního náhodného prohledávání,kritérium DP_MAX, reprezentace Luffa

Pro reprezentaci i686 byly zvolen parametr 1 čtveřice (tj. 1 bit v registru), aby měla počet vstupů i výstupů rovný čtyřem. Z tabulky 7.14 je patrná podobnost výsledků těchto reprezentací. Zde však parametr mutace hraje větší roli. Z krabicového grafu na obrázku 7.2 je patrné, že příliš nízká a příliš vysoká pravděpodobnost mutace zhoršuje výsledek pro bent skóre, hodnoty mezi tím jsou stabilizované. Jako nejlepší parametr však je vybráno 0.7, protože by při některých jiných kritérií, mohly hodnoty u mutací 0.01 až 0.6 znamenat uvíznutí v lokálním extrému. To samé platí pro kritérium $DP_{\max}$. na obrázku 7.3. O ostatních kritériích na obrázcích C.13 až C.16 se dá konstatovat totéž.

Obě reprezentace dospívají k obdobným výsledkům s tím rozdílem, že u každé jsou jiné hodnoty mutace. To je zřejmě dané strukturou chromozómů a různými způsoby mutování.

Kritérium	pMut	maximum	medián	std. odchylka	průměr	minimum
$LP_{\max}$	0.5	2	2	0.263511	1.86451	0.830075
$DP_{\max}$	0.05	3	2	0.141421	2	1
bent skóre	0.7	64	64	4.53159	59.8606	48
SAC	0.05	3	3	0.729079	2.78	0
BF	0.05	4	4	0.42989	3.7919	2
stupeň polynomu	0.01	4	4	0	4	4

Tabulka 7.14: Nejlepší pravděpodobnosti mutace pro paralelní náhodné programování nad reprezentací i686

7.4 Parametry multikriteriálního vyhledávání

Pro jedno kritérium se tedy dá dospět k praktickým mezním hodnotám rychle a ne moc náročně. Ovšem jak vypadají výsledky a jaké parametry vyšly nejlépe, když bude třeba splnit všechna kritéria najednou, probere následující podkapitola. Nejprve bylo nutné stanovit jak vůbec výsledky pro každou dvojici pravděpodobností mutace a křížení mezi sebou porovnávat. Nabízel se počet ve všech nadprůměrných jedinců. Průměrný jedinec byl vypočten ze všech běhů pro všechny parametry, tj. po provedení všech potřebných evolucí byly vypočteny průměrné hodnoty pro každé kritérium a z nich byl vytvořen vektor o stejných dimenzích jako ohodnocení kteréhokoliv skutečného jedince. Pro každou dvojici pravděpodobností mutace a křížení byli spočteni jedinci, kteří tomuto „průměrnému jedinci“ slabě dominují. Jako kritéria byla použita všechna, která byla zkoumána v předcházejících podkapitolách, tedy bent skóre, $LP_{\max}$, $DP_{\max}$, SAC, faktor větvení a stupeň polynomu. Použitými algoritmy byly VEGA a SPEA.

Parametry pro permutaci

Nejprve evoluce proběhla při velikosti populace 100 při 1000 generacích pro permutaci se čtyřmi vstupy a čtyřmi výstupy.

Algoritmus VEGA dospěl k průměrnému jedinci[5] s bent skóre 29.61, $-log_2 LP_{\max} = 0.99$ tj. $LP_{\max} = 0.5$, $-log_2 DP_{\max} = 1.31$ tj. $DP_{\max} = 0.4$, SAC 0.95, faktor větvení 2 a stupeň polynomu 2.77.

Algoritmus SPEA dospěl k bent skóre 31.50, $-log_2 LP_{\max} = 1.91$ tj. $LP_{\max} = 0.27$, $-log_2 DP_{\max} = 1.93$ tj. $DP_{\max} = 0.26$, SAC 0.98, faktor větvení 2.04 a stupeň polynomu 2.96.

Žádný algoritmus nedospěl k žádnému parametru, co by měl nějaké nadprůměrné jedince. Proto byly i vzhledem k časové náročnosti algoritmu Spea a na základě výsledků z testovací fáze programu zvoleny jiné parametry na počet generací na 200 a velikost populace 200. Počet běhů zůstal na 100. Hodnoty průměrného jedince byly ještě před stanovením počtů nadprůměrných zaokrouhleny dolů na celé desetiny.

Algoritmus VEGA dospěl k tomuto průměrnému jedinci s bent skóre 29.1, $-log_2 LP_{\max} = 1.7$ tj. $LP_{\max} = 0.30778610333622908$, $-log_2 DP_{\max} = 1.5$ tj. $DP_{\max} = 0.35355339059327379$, SAC 0.2, faktor větvení 2.0 a stupeň polynomu 2.7. V tabulce C.3 jsou uvedeny výsledky pro jednotlivé kombinace pravděpodobností. Nejlépe vyšla kombinace s $pMut = 0.3$ a $pCross = 0.8$, počty však byly docela vyrovnané.

Algoritmus SPEA dospěl k o něco lepšímu průměrnému jedinci, podobně jak tomu bylo u vyššího počtu jedinců v populaci, s bent skóre 31.6, $-log_2 LP_{\max} = 1.9$ tj. $LP_{\max} = 0.26794336563407328$, $-log_2 DP_{\max} = 1.9$ tj. $DP_{\max} = 0.26794336563407328$, SAC 0.9, faktor větvení 2.0 a stupeň polynomu 2.9. V tabulce C.4 jsou uvedeny výsledky pro jednotlivé kombinace pravděpodobností. Nejlépe vyšla kombinace s $pMut = 0.9$ a $pCross = 0.05$.

Zvýšení velikosti populace tedy populace zlepšilo průměry jednotlivých kritérií.

Parametry pro i686

Dále proveden předchozí pokus, ale z reprezentací i686 se čtyřmi vstupy a čtyřmi výstupy, navíc přibylo kritérium bijektivní skóre, neboť tato reprezentace nezaručuje bijektivitu.

[5] při sázení textu všechny hodnoty zaokrouhleny na dvě desetinná místa

Algoritmus VEGA dospěl k průměrnému jedinci s bent skóre 14.4, $-log_2 LP_{\max} = 0.1$ tj. $LP_{\max} = 0.93303299153680741$, $-log_2 DP_{\max} = 0.6$ tj. $DP_{\max} = 0.659753955386447$, bijektivní skóre 8.8, SAC 0.0, faktor větvení 1.0 a stupeň polynomu 1.5. V tabulce C.5 jsou uvedeny výsledky pro jednotlivé kombinace pravděpodobností. Nejlépe vyšla kombinace s $pMut = 0.01$ a $pCross = 0.09$.

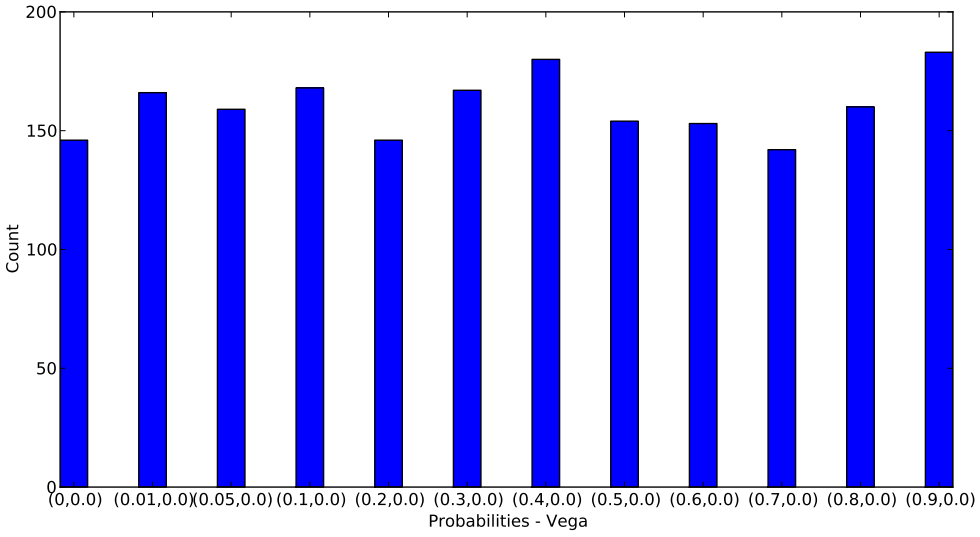
Algoritmus SPEA dospěl k průměrnému jedinci s bent skóre 21.5, $-log_2 LP_{\max} = 0.4$ tj. $LP_{\max} = 0.75785828325519899$, $-log_2 DP_{\max} = 0.8$ tj. $DP_{\max} = 0.57434917749851744$, bijektivní skóre 12.2, SAC 0.4, faktor větvení 1.5 a stupeň polynomu 2.2. V tabulce C.6 jsou uvedeny výsledky pro jednotlivé kombinace pravděpodobností. Nejlépe vyšla kombinace s $pMut = 0.05$ a $pCross = 0.7$

Přidání bijektivního skóre má vliv na degradaci ostatních parametrů.

Parametry pro CGP

A pokus byl proveden ještě s reprezentací CGP se 4 vstupními bity, 4 výstupními bity, 4 sloupci, 3 řádky a 6 maximálními mutacemi). Samozřejmě, že tentokrát nebylo užito křížení, proto je ve výsledcích méně kombinací.

Algoritmus VEGA dospěl k průměrnému jedinci s bent skóre 8.1, $-log_2 DP_{\max} = 0.1$ tj. $LP_{\max} = 0.93303299153680741$, $-log_2 DP_{\max} = 0.5$ tj. $DP_{\max} = \frac{\sqrt{2}}{2}$, bijektivní skóre 10.3, SAC 0.0, faktor větvení 1.0 a stupeň polynomu 1.1. V tabulce C.7 jsou uvedeny výsledky pro jednotlivé pravděpodobnosti mutace. V grafu na obrázku 7.4 jsou porovnány počty nadprůměrných . Nejlépe vyšla pravděpodobnost mutace $pMut = 0.9$. Na této reprezentaci vyšlo lépe bijektivní skóre, avšak o to jsou horší ostatní parametry.



Obrázek 7.4: Parametry cgp pro VEGA

Algoritmus SPEA dospěl k průměrnému jedinci s bent skóre 21.8, $-log_2 LP_{\max} = 0.3$ tj. $LP_{\max} = 0.81225239635623558$, $-log_2 DP_{\max} = 0.8$ tj. $DP_{\max} = 0.57434917749851744$, bijektivní skóre 12.7, SAC 0.3, faktor větvení 1.4 a stupeň polynomu 2.1. Algoritmus Spea bez křížení nedospěl k žádné nadprůměrné kombinaci. Na této reprezentaci je o něco lepší

bijektivní skóre, a o to jsou horší ostatní parametry.

7.5 Hledání složitějších s-boxů

Bylo provedeno i hledání s-boxů o rozměrech 6×6, 6×4, 8×8 algoritmy VEGA a SPEA, jak na permutacích tak na reprezentacích CGP, i686. Opět se potvrdilo, že pokud je bijektivita zaručena reprezentací, dosáhlo se vysokých hodnot kritérií, s tím, že vysoce hodnocených bylo několik a každý jedinec byl lepší v něčem jiném (pareto optimum). Při bijektivitě jako zvláštní kritérium byly hodnoty všech kritérií malé. Již na 30 generacích a velikosti populace 30 a počtu běhů 4, kdy z jednotlivých běhů byla vytvořena sada nedominovaných výsledků. Pro všechny výsledky v této nedominované sadě platilo pro rozměry 6×6 při reprezentaci permutace, že $-log_2 LP_{\max} = 2.38529015$ nebo 2.83007503 pro algoritmus SPEA, u algoritmu VEGA mělo několik jedinců 2. Stupeň polynomu byl 4 nebo 5 u obou algoritmů. Oba nalezly různé jedince s $-log_2 DP_{\max} = 3.41503739$.

U algoritmu SPEA měl tento jedinec chromozóm:

```
5 49 43 44 3 30 35 59 40 29 13 53 11 4 2 18
39 41 38 10 57 36 7 20 17 37 47 24 5 55 62 51
22 60 26 16 14 0 21 63 48 1 34 19 27 52 42 61
12 50 46 54 32 31 58 8 15 6 9 56 33 28 23 45
```

U algoritmu VEGA měl tento jedinec chromozóm:

```
50 35 21 23 3 38 15 25 26 11 8 62 32 42 47 51
57 28 24 48 58 54 19 56 55 13 60 2 43 0 52 20
46 29 14 59 31 17 9 41 30 22 40 4 49 10 37 27
63 45 5 33 6 39 44 61 18 16 34 53 1 36 7 12
```

Obdobné výsledky byly i u počtu generací 200 a velikosti populace 200. U s-boxů 6×4 byly všechny $LP_{\max} = 1$. Některé s-boxy 8×8 docela rychle splnily podmínky[6] pro výběr $LP_{\max} \leq 0.0625$ a $DP_{\max} \leq \frac{10}{256}$ při hledání pomocí algoritmu SPEA, počtu generací 30 a velikosti populace 30. Na obr. C.1 je uvedený jeden z nich, který má bent skóre 250, $-log_2 LP_{\max} = 4.18621874$ tj. $LP_{\max} = 0.05493164$, $-log_2 DP_{\max} = 4.67807198$ tj. $DP_{\max} \doteq \frac{10}{256}$, bijektivní skóre 255, SAC 0, faktor větvení 2 a stupeň polynomu 7. Zobrazení nuly na nulu je spíše nežádoucí, avšak to se týká jen u tohohle konkrétního jedince.

Ostatní řešení, co měla $-log_2 DP_{\max} = 4$, měla o něco málo vyšší bent skóre. K obdobným výsledkům dojde i náhodné prohledávání, ale potřebuje více běhů. Náhodné permutace větších velikostí totiž mají nízké pravděpodobnosti $LP_{\max}$ a $DP_{\max}$, jak je uvedeno v [6]. A hodně náhodných permutací má $-log_2 DP_{\max} = 4.67807198$ a $-log_2 DP_{\max} = 4$. K jedinci s $-log_2 LP_{\max} = 4.18621874$[7] však evoluce dospěje evoluce s méně běhy, jinak řečeno je nutné méně náhodně vygenerovaných jedinců.

[6] které jsou uvedeny v [16]

[7] a dalšími parametry výše

7.6 Dvoufázové hledání

Kvůli zaručení bijektivity u reprezentací CGP a i686 se jako další možnost[8] řešení nabízí rozdělit evoluci na dvě fáze. V první fázi se na reprezentaci permutace najde nejlepší řešení. To je pak použito jako trénovací množina pro symbolickou regresi pomocí reprezentací CGP či i686. Prvotně nastává problém s výběrem vhodného kandidáta, který postoupí do druhé fáze. Symbolická regrese bude muset být provedena několikrát pro více jedinců. Dále se bude muset přijmout fakt, že i symbolická regrese má svá omezení a problém se na ni jen přenáší. Symbolická regrese byla provedena na reprezentacích CGP a i686 o rozměrech 2×2, 4×4, 6×6, 8×8 na různých permutacích jako trénovací množiny. Trénovací množina je úplná, tj. obsahuje všechny vstupy a výstupy.

Parametry symbolické regrese

Již při počtu generací 50 a velikosti populace 1000 se v reprezentaci jako acyklický graf s 3 sloupci, 3 řádky a 3 maximálními mutacemi, nalezne 3×3 za nutnosti několika běhů. U složitějších s-boxů již záleží na tvaru funkce, např. pro $f(x) = x$ tj. $[1, 2, \cdots, n-1]$, je bez problému nalezeno řešení i pro 8×8.

Vedlejším produktem tohoto rozdělení může být optimální realizace, avšak základním kritériem musí být přesnost. Nejoptimálnější se vybere až z několika přesných řešení.

7.7 Celkové zhodnocení výsledku

Při jednokriteriální optimalizaci na provedených experimentech dochází jak náhodné prohledávání tak genetický algoritmus k téměř obdobným výsledkům. Pouze v některých případech je výrazněji lepší buď jedno nebo druhé. V multikriteriálním prohledávání pomocí algoritmů VEGA a SPEA lze snadněji zajistit lepší hodnoty více kritérií souběžně než u náhodného prohledávání. Zatímco náhodné prohledávání je algoritmicky méně složitější, genetický algoritmus nepotřebuje tolik běhů (náhodných inicializací jedinců na počátku). Tedy jsou případy reprezentací jako permutace, kdy lze evoluční algoritmy použít. Zaručit bijektivitu lze spolehlivě jen u permutací. U ostatních reprezentací se při multikriteriálním prohledávání po přidání bijektivního skóre výrazně zhoršují hodnoty ostatních kritérií. A pokud pro všechny nebijektivní jedince bude hodnota každého kritéria nulová, či-li napřed je proveden test na bijektivitu a poté je teprve proveden výpočet kritéria, poskytuje ještě horší výsledky. Nejlépe by bylo, kdyby pro reprezentace CGP a i686 existovaly genetické operátory, které bijektivitu zaručí. U těchto reprezentací se projevuje složitost jejich struktur (mnoho možností pro mutace a křížení a častost opakování toho samého jedince), a tím pádem evoluční algoritmus hledá řešení obtížněji [18]. Obdobné závěry lze konstatovat i u hledání minimální realizace (podle počtu použitých hradel) u reprezentace CGP, kdy je populace zanesena minimálními nefunkčními obvody. Rozdělení na dvě fáze je pouhé přenesení na problém symbolické regrese. Minimální realizace musí být vybrána, až po splnění kritérií bezpečnosti. V nalezených zdrojích se nezabývají multikriteriálním prohledáváním s více kritérii bezpečnosti. V [13] se zabývají optimální strukturou hardware, avšak hodnoty kritérií bezpečnosti nejsou zmíněny.

[8] po bijektivním skóre jako kritérium a omezení bijektivity v objektivní funkci

Kapitola 8

Závěr

Bylo navrženo jednoduché rozhraní pro popis parametrů evoluce, aby jej bylo možné v budoucnu použít na další experimenty. Ve fázi implementace muselo být naprogramováno efektivní provedení různých evolučních algoritmů. K tomu bylo využito nástrojů valgrind, který pomáhá odhalit chyby práce s pamětí, a oprofile, hledající části kódu, které trvají nejdéle. Kvůli tomu, aby Python rozhraní mělo možnost jednoduše popsat parametry evoluce, musela být provedena práce na úrovní C++, která se navrátila jednodušším popisem experimentů a zpracováním výsledků. Experimenty ukázaly, které algoritmy, a jaké jejich použité parametry jsou lepší při určitých kritériích na zvolených reprezentacích, a to jak pro jednotlivá kritéria bezpečnosti substitučních boxů, tak i pro multikriteriální optimalizaci. V [17] je zmíněno, že bylo provedeno málo porovnaní Genetických algoritmů a algoritmů EDA. I z toho důvodu bylo provedeno srovnání na tomto konkrétním problému. Nebylo možné provést všechny možné kombinace algoritmů, reprezentací, kritérií a parametrů. K experimentům byly vybrány ty, které se zdály být zajímavé.

Náměty pro další rozvoj

Pro s-boxy složitější, než se kterými byly provedeny experimenty v této práci, je už nutné zvážit hardwarovou akceleraci výpočtů jednotlivých kritérií a prohledávacích algoritmů. Dalším kritériem by mohl být řád polynomu v $GF(2^m)$. Vyšší řád [15, 3] polynomu znamená vyšší odolnost vůči útoku pomocí interpolace Langrangeovým polynomem. Jedná se o zobecnění lineární kryptoanalýzy. Právě pomocí Lagrangeovy interpolace se získá polynom vyjadřující s-box. Jeho stupeň je dále použitelný k maximalizaci. Existuje ovšem několik variant Galois polí, ze kterých si útočník může vybrat, tudíž je nutná odolnost vůči všem.

Literatura

[1] Adams, C.; Tavares, S.: Good S-Boxes Are Easy To Find. 1990.

[2] Canniére, C. D.; Sato, H.; Watanabe, D.: Hash Function Luffa – Specification. Technická zpráva, Systems Development Laboratory, Hitachi, Ltd., 2008.

[3] Courtois, N. T.: The Inverse S-box, Non-linear Polynomial Relations and Cryptanalysis of Block Ciphers. Technická zpráva, Axalto Cryptographic Research & Advanced Security, 2007.

[4] Daemen, J.; Rijmen, V.: AES Proposal: Rijndael. Technická zpráva, 1999.

[5] Delman, B.: *Genetic Algorithms in Cryptography*. Diplomová práce, Rochester Institute of Technology, Kate Gleason College of Engenineering, July 2004.

[6] Gordon, J. A.; Retkin, H.: Are Big S-Boxes best? 1998.

[7] Heys, H. M.: A tutorial on linear and differential cryptanalysis. *Cryptologia*, ročník 26, č. 3, 2002: s. 189–221, ISSN 0161-1194.
URL `<http://www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.pdf>`

[8] KIM, K.: *A Study on the Construction and Analysis of Substitution Boxes for Symmetric Cryptosystems*. Dizertační práce, Yokohama National University, December 1990.

[9] Mařík, V.; Štěpánková, O.; Lažanský, J.; aj.: *Umělá inteligence (1)*. Academia Praha, 1993.

[10] Mařík, V.; Štěpánková, O.; Lažanský, J.; aj.: *Umělá inteligence (4)*. Academia Praha, 2003.

[11] Miller, J. F.; Thomson, P.: Cartesian Genetic Programing. 2000.

[12] Mister, S.; Adams, C.: Practical S-Box Design. 2000.
URL `<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.40.7715&rep=rep1&type=pdf>`

[13] Nedjah, N.; de Macedo Mourelle, L.: Pareto-Optimal Hardware for Substitution Boxes. *Journal of Universal Computer Science*, ročník 12, č. 4, 2006: s. 395–407.

[14] Pelikan, M.: Implementation of the Dependency-Tree Estimation of Distribution Algorithm in C++. Technická zpráva, Missouri Estimation of Distribution Algorithms Laboratory (MEDAL), 2006.

[15] RezaeiPour, D.; Said, M. R. M.: New Directions in Cryptanalysis of Block Ciphers. *Journal of Computer Science 5*, ročník 12, 2009: s. 1094–1097.

[16] Schneier, B.; Kelsey, J.; Whiting, D.; aj.: Twofish: A 128-Bit Block Cipher. June 1998.
URL <http://www.counterpane.com/twofish.html>

[17] Schwarz, J.; Lukáš, S.: Aplikované evoluční algoritmy - studijní opora. 2006.

[18] Sekanina, L.: Přednášky předmětu Biologií inspirované počítače. 2008.
URL <https://www.fit.vutbr.cz/study/courses/BIN/private/>

[19] Thiele, L.: Evolutionary Methods in Multi-Objective Optimization – Why do they work ? –. ONLINE, 2. května 2010.
URL <http://www.slidefinder.net/E/Evolutionary_Methods_Multi_Objective_Optimization/10784121>

[20] Vašíček, Z.: Nástroje pro kartézské genetické programování Tools4CGP. 2008.
URL <http://www.fit.vutbr.cz/~vasicek/cgp/>

[21] Wall, M.: GAlib – A C++ Library of Genetic Algorithm Components – version 2.4.7. ONLINE, March 2007.
URL <http://lancet.mit.edu/ga/>

[22] Wikipedia: Bent function. ONLINE, April, 23.4.2010.
URL <http://en.wikipedia.org/wiki/Bent_function>

[23] Wikipedia: CBC-MAC. ONLINE, 13.5.2010.
URL <http://en.wikipedia.org/wiki/File:CBC-MAC_structure_(en).svg>

[24] Wikipedia: Feistel Cipher. ONLINE, 14.5.2010.
URL <http://en.wikipedia.org/wiki/Feistel_cipher>

[25] Wikipedia: Rijndael S-box. ONLINE, 3.5.2010.
URL <http://en.wikipedia.org/wiki/Rijndael_S-box>

[26] Wikipedia: S-box. ONLINE, 3.5.2010.
URL <http://en.wikipedia.org/wiki/S-box>

Seznam použitých zkratek

AES Advanced Encryption Standard – šifra Rijndael schválená jako standard

BF Branching factor – faktor větvení (viz podkapitola 3.2.5 na str. 17)

BMDA Bivariate Marginal Distribution Algorithm – druh EDA (str. 25)

BOA Bayesian Optimization Algorithm – druh EDA používající Bayesovské sítě jako pravděpodobnostní model

CGP Cartesian Genetic Programing – viz str. 23

CREW Concurent Read Exclusive Write – jeden z možných modelů paralelizmu

DES Data Encryption Standard – šifra s 56-bitovým klíčem vyvinutá v 70. letech, v původní formě dnes již nespolehlivá

DSA Digital Signature Algorithm – algoritmus pro elektronický podpis založený na diskrétních logaritmech

EA Evoluční Algoritmus – nadtřída prohledávacích algoritmů zmíněných v této práci

EDA Estimation of Distribution Algorithm – druh EA založený na pravděpodobnostním modelu (str. 24)

GA Genetický algoritmus – v klasické podobě (viz str. 21)

GaLib Genetic Algorithm Library – knihovna pro Genetický algoritmus a genetické programování [21]

GCC Gnu Compiler Collection – sada překladačů různých jazyků pro UNIXové systémy `<http://gcc.gnu.org/>`

GP Genetické programování – evoluční algoritmus pracující rovnou se spustitenými strukturami místo číselného chromozomu

MD5 Message Digest 5 – jeden z hashovacích algoritmů Rona Rivesta, který mj. spolupracoval na RSA

MOSAC maximal order SAC – SAC nejvyššího možného řádu

POSIX Portable Operating System Interface – standardizované přenositelné rozhranní pro operační systémy

RC4 Rivest Cipher 4 – jedna ze symetrických proudových šifer Rona Rivesta, který mj. spolupracoval na RSA

RSA Rivest – Shamir – Adleman – nejznámější asymetrická šifra, založená na faktorizaci součinu dvou velkých prvočísel, princip vymysleli dříve v Británii avšak jej tajili, nezávisle na to samé přišla trojice Američanů

SAC Strict Avalanche Criterion – viz str. 14

SHA Secure Hash Algorithm – standardizovaný hashovací algoritmus, v současnosti SHA2, SHA3 ve vývoji

SPEA Strength Pareto Evolutionary Algorithm – multikriteriální GA s elitou obsahující nashlukované Pareto-optimální jedince (str. 27)

UMDA Univariate Marginal Distribution Algorithm – druh EDA (str. 25)

UML Unified Modeling Language – sada zformalizovaných grafických reprezentací návrhu/modelu programu

VEGA Vector Evaluated Genetic Algorithm – jednoduchý multikriteriální GA (str. 27)

Seznam příloh

Součástí odevzdané práce jsou tyto přílohy:

A Chromozóm v Kartézském genetickém programování

B Přehled příkazů pro skriptování experimentů

C Podrobné tabulky a grafy

DVD DVD nosič

DVD nosič obsahuje:

- kompletní zdrojové kódy programu
- kompletní zdrojové kódy tohoto textu
- výstupy programu při jednotlivých experimentech

Příloha A

Chromozóm v Kartézském genetickém programování

Tato příloha vysvětluje značení v chromozómu kartézského genetického programování. Notace byla použita ve shodě s CGPTools [20]

Tvar chromozómu

Hlavička obsahuje údaje v tomto pořadí:

```
{počet_vstupů, počet_výstupů, sloupce, řádky, vstupů_do_bloku, l_back,
 počet_použitých_bloků}
```

Následují popisy jednotlivých bloků (za vstupní blok se považuje i vstup celého obvodu, bloky se číslují od počtu vstupů):

```
([číslo bloku] vstupní_blok_a, vstupní_blok_b, funkce)
```

a na konec jsou výstupy některých bloků (uvedena čísla) prohlášeny za výstupy celého obvodu:

```
(seznam bloků)
```

Přiřazení funkcí

pro vstupní proměnné a a b

číslo funkce	výraz
0	a
1	a AND b
2	a OR b
3	a XOR b
4	NOT a
5	NOT b
6	a AND (NOT b)
7	a NAND b
8	a NOR b

Příloha B

Přehled příkazů pro skriptování experimentů

Tento krátký manuál je určen těm, kteří budou chtít provádět vlastní experimenty v evolučním hledání s-boxů. Pro správný (paralelní) chod je nutné mít zkompilovanou knihovnu Galib v upravené podobě. Pro sekvenční byla odzkoušena i knihovna původní. Po instalaci knihovny je možné pomocí příkazu *make* zkompilovat dynamickou knihovnu pro hledání s-boxů. Pokud nejde z nejakeho důvodu Galib nainstalovat a jen zkompilovat v adresáři, musí se v makefile pomocí přepínačů -I a -L nastavit cesta k této knihovně. (V odevzdávané podobě jsou zdrojové kódy tak jak běžely na serveru merlin, takže makefile je nastavený).

Dále popisované objekty jsou v (i)pythonu přístupné pomocí příkazu:

```
from evolution import *
```

tak jak vidět v souboru experiments.py. V tomto souboru jsou též provedené experimenty jako příklady.

Definování genomu

Pro zvolení reprezentace slouží konstruktor objektu Genome, kterému je třeba předat typ, objektivní funkci pro jednokriteriální optimalizaci:

```
Genome(chromozomeType, criterion, [specific])
```

kde `chromozomeType` může být
`ChromozomeType.BINARY` pro binární reprezentaci; `specific` jsou počet vstupních bitů, počet výstupních bitů
`ChromozomeType.PERMUTATION` pro reprezentaci jako seznam výstupů; `specific` vyžadování permutace, počet vstupních bitů, počet výstupních bitů
`ChromozomeType.CGP` pro reprezentaci jako acyklický graf hradel; `specific` počet vstupních bitů, počet výstupních bitů, počet sloupců, počet řádků, maximální počet mutací jednoho prvku
`ChromozomeType.SOFTWARE_IMPL` pro reprezentaci jako posloupnost trojinstrukcí i686; `specific` počet čtveřic bitů na vstupu a výstupu zároveň

a `criterion`
(pro multikriteriální optimalizaci ignorován, lze však zadat `CriterionFunction.NONE_FITNESS`)
`CriterionFunction.BENT_AND_MOSAC` pro zvolení kritéria bent skóre,

`CriterionFunction.LP_MAX` pro linear probability,
`CriterionFunction.DP_MAX` pro differential probability,
`CriterionFunction.BIJECTIVE_SCORE` pro bijektivní skóre,
`CriterionFunction.SAC` pro řád Strict avalanche criterion,
`CriterionFunction.BF` pro faktor větvení,
`CriterionFunction.POLYNOMIAL_DEGREE` pro stupeň algebraického polynomu

Definování parametrů hledání, výběr algoritmu

Vytvoření nového hledání se provede pomocí zavolání kostruktoru:

```
Searching(algorithm, genome, popSize, generations, [specific])
```

`algorithm` je
`"Ga"` pro jednokriteriální genetický algoritmus (genetické programování);
`specific` pravděpodobnost mutace, pravděpodobnost křížení, zapnutí elitismu
`"Vega"` pro algoritmus VEGA; `specific` pravděpodobnost mutace, pravděpodobnost křížení, seznam kritérií
`"Spea"` pro algoritmus SPEA; `specific` pravděpodobnost mutace, pravděpodobnost křížení, seznam kritérií
`"Eda"` pro algoritmus EDA; `specific` typ (`False` pro UMDA, `True` BMDA)
`"Random"` pro náhodné prohledávání; velikost populace znamená počet uchovávaných nejlepších výsledků
`"ParallelRandom"` pro paralelní náhodné prohledávání; `specific` pravděpodobnost mutace

`genome` je reprezentace vytvořená podle pravidel výše a `popSize` resp. `generations` je velikost populace resp. počet generací

Spuštění evoluce a získání výsledku

Třída Searching obsahuje:

- instanční metodu `simpleEvolve()` – umožňuje spustit jeden experiment
- třídní metodu `parallelSearching(list)` – provede seznam `list` objektů typu Searching paralelně
- instanční metodu `statistics()` – po provedení evoluce vrátí object TStatistics
- instanční metodu `bestPopulationStrings()` – vrátí seznam řetězcových reprezentací všech `statistics().nBestGenomes` nejlepších jedinců za celý průběh evoluce

Třída TStatistics obsahuje:

- `online` – průměr skóre za celý průběh evoluce
- `offlineMax` – průměr minimálních skóre
- `offlineMin` – průměr maximálních skóre
- `bestPopulationScores` – skóre všech `nBestGenomes` nejlepších jedinců za celý průběh evoluce

- `bestPopulationCriterionsValues` – hodnoty jednotlivých kritérií všech `nBestGenomes` nejlepších jedinců za celý průběh evoluce
- `bestPopulationOutputs` – výstupy všech `nBestGenomes` nejlepších jedinců za celý průběh evoluce
- `maxEver` – největší dosažené skóre,
- `minEver` – nejmenší dosažené skóre,
- `generation` – poslední generace,
- `convergence` – konvergence evoluce,
- `selections` – počet provedených selekcí za celý průběh evoluce,
- `crossovers` – počet provedených křížení za celý průběh evoluce,
- `mutations` – počet provedených mutací za celý průběh evoluce,
- `replacements` – počet provedených nahrazení za celý průběh evoluce,
- `nBestGenomes` – počet uchovávaných nejlepších jedinců,

Položky `bestPopulationScores` ap. jsou typu `numpy.Array`

Provedení symbolické regrese z výsledku předchozího hledání

Pro symbolickou regresi je třeba vytvořit specializovaný genom

```
SymbolicalRegresionGenome(chromozomeType, list, [specific])
```

kde `list` je seznam požadovaných výstupů, musí odpovídat parametrům boxu ostatní parametry jsou shodné s třídou `Genome`

Příloha C

Podrobné tabulky a grafy

V této příloze jsou uvedeny tabulky a grafy ze 7. kapitoly, které jsou příliš podrobné na to, aby byly v té kapitole obsaženy. Kapitola 7 se na ně dovolává pomocí křížových odkazů a tam je též k nim umístěn komentář. Hodnoty v tabulkách ve sloupci „Parametry“ jsou ve tvaru (p_{Mut}, p_{Cross}), kde p_{Mut} je pravděpodobnost mutace a p_{Cross} pravděpodobnost křížení. Krabicové grafy na obrázcích C.2 až C.16 znázorňují rozložení stejně početných skupin hodnot. Horní a dolní strana modrého obdélníka znázorňuje umístění kvartilů. Prostřední červená vodorovná úsečka přestavuje medián. Nahoru a dolů od obdélníka vede tzv. vous. Horní resp. dolní vous znázorňuje nejvyšší resp. nejnižší hodnotu, na kterou má být brán zřetel. Křížky jsou označeny odlehlé hodnoty.

Tabulka C.1: Výsledky pro celý interval pro kritérium SAC při hledání parametrů GA

Parametry	maximum	medián	std. odchylka	průměr	minimum
(0, 0)	1	0	0.0809699	0.0066	0
(0, 0.4)	2	1	0.0509265	1.0026	1
(0, 0.6)	2	1	0.0623285	1.0039	1
(0.4, 0.4)	2	1	0.0809684	1.0066	1
(0.05, 0.4)	2	1	0.0845514	1.0072	1
(0, 0.2)	3	0	0.441642	0.263	0
(0.4, 0)	3	1	0.0785067	1.006	1
(0, 0.8)	3	1	0.0803715	1.0063	1
(0.6, 0.4)	3	1	0.0815945	1.0065	1
(0.8, 0)	3	1	0.082812	1.0065	1
(0.4, 0.2)	3	1	0.0845636	1.007	1
(0.6, 0.6)	3	1	0.0857259	1.0072	1
(0.6, 0)	3	1	0.0863206	1.0069	1
(0.05, 0.6)	3	1	0.0863215	1.0069	1
(0.6, 0.8)	3	1	0.08631	1.0071	1
(0.1, 0.4)	3	1	0.0868963	1.007	1
(0.8, 0.6)	3	1	0.0874718	1.0069	1
(0.8, 0.8)	3	1	0.0874622	1.0071	1
(0.05, 0)	3	1	0.0874461	1.0073	1
(0.05, 0.8)	3	1	0.0885629	1.0075	1
(0.8, 0.4)	3	1	0.088563	1.0075	1
(0.1, 0)	3	1	0.0896852	1.0075	1
(0.6, 0.2)	3	1	0.0902371	1.0076	1
(0.2, 0.2)	3	1	0.0902182	1.0078	1
(0.1, 0.8)	3	1	0.0902183	1.0078	1
(0.05, 0.2)	3	1	0.0907574	1.0079	1
(0.4, 0.8)	3	1	0.0913398	1.0076	1
(0.2, 0.6)	3	1	0.0918783	1.0077	1
(0.1, 0.6)	3	1	0.091852	1.0079	1
(0.2, 0.4)	3	1	0.0929026	1.0083	1

Tabulka C.1: Výsledky pro celý interval pro kritérium SAC při hledání parametrů GA

Parametry	maximum	medián	std. odchylka	průměr	minimum
(0.2, 0.8)	3	1	0.0934659	1.008	1
(0.4, 0.6)	3	1	0.0939728	1.0083	1
(0.8, 0.2)	3	1	0.0945171	1.0082	1
(0.2, 0)	3	1	0.0945181	1.0082	1
(0.1, 0.2)	3	1	0.0985637	1.0092	1

Tabulka C.2: Výsledky pro celý interval pro kritérium Branching Factor při hledání parametrů GA

Parametry	maximum	medián	std. odchylka	průměr	minimum
(0, 0)	2	2	0	2	2
(0, 0.2)	3	2	0.0528395	2.0028	2
(0, 0.4)	3	2	0.0879735	2.0078	2
(0, 0.6)	3	2	0.102886	2.0107	2
(0, 0.8)	3	2	0.125482	2.016	2
(0.8, 0)	3	2	0.134745	2.0185	2
(0.4, 0.2)	3	2	0.135111	2.0186	2
(0.05, 0.4)	3	2	0.137571	2.0193	2
(0.05, 0.2)	3	2	0.138615	2.0196	2
(0.8, 0.8)	3	2	0.138963	2.0197	2
(0.6, 0)	3	2	0.139993	2.02	2
(0.05, 0.6)	3	2	0.139993	2.02	2
(0.8, 0.6)	3	2	0.14171	2.0205	2
(0.6, 0.6)	3	2	0.142047	2.0206	2
(0.4, 0.4)	3	2	0.142375	2.0207	2
(0.4, 0.8)	3	2	0.142375	2.0207	2
(0.1, 0)	3	2	0.142708	2.0208	2
(0.6, 0.4)	3	2	0.142709	2.0208	2
(0.2, 0.4)	3	2	0.143388	2.021	2
(0.1, 0.6)	3	2	0.144042	2.0212	2
(0.1, 0.4)	3	2	0.144043	2.0212	2
(0.4, 0.6)	3	2	0.144386	2.0213	2
(0.8, 0.4)	3	2	0.144713	2.0214	2
(0.1, 0.2)	3	2	0.144713	2.0214	2
(0.6, 0.8)	3	2	0.14504	2.0215	2
(0.2, 0)	3	2	0.145706	2.0217	2
(0.05, 0)	3	2	0.147008	2.0221	2
(0.6, 0.2)	3	2	0.147008	2.0221	2
(0.05, 0.8)	3	2	0.147329	2.0222	2
(0.2, 0.6)	3	2	0.147329	2.0222	2
(0.2, 0.8)	3	2	0.147664	2.0223	2
(0.2, 0.2)	3	2	0.148297	2.0225	2
(0.4, 0)	3	2	0.148615	2.0226	2
(0.8, 0.2)	3	2	0.148615	2.0226	2
(0.1, 0.8)	3	2	0.156425	2.0251	2

Tabulka C.3: Průměry kritérií, permutace, algoritmus Vega

Parametry	Počet nadprůměrných	Bent skóre	$LP_{\max}$	$DP_{\max}$	SAC	faktor větvení	stupeň polynomu
(0, 0)	4	30.552	1.3164	1.50276	0.04155	2	2.86885
(0, 0.01)	129	30.0604	1.62747	1.60609	0.06055	2	2.80555
(0, 0.05)	136	29.5706	1.90148	1.63849	0.1023	2.00005	2.74485
(0, 0.1)	126	29.2994	1.87887	1.61536	0.1273	2	2.7172
(0, 0.2)	199	29.1722	1.84383	1.59467	0.15375	2	2.70585
(0, 0.3)	218	29.1935	1.8149	1.59044	0.184	2	2.7079
(0, 0.4)	197	29.1935	1.80199	1.58468	0.19355	2	2.7082
(0, 0.5)	209	29.1345	1.78896	1.57468	0.20615	2	2.703
(0, 0.6)	262	29.1197	1.78238	1.57491	0.2143	2	2.70505

Tabulka C.3: Průměry kritérií, permutace, algoritmus Vega

Parametry	Počet nadprů-měrných	Bent skóre	$LP_{\max}$	$DP_{\max}$	SAC	faktor větvení	stupeň po-lynomu
(0, 0.7)	258	29.0871	1.7782	1.5706	0.218	2	2.70025
(0, 0.8)	249	29.1013	1.77824	1.5717	0.2158	2	2.6989
(0, 0.9)	240	29.0779	1.77244	1.56821	0.21885	2	2.699
(0.01, 0)	248	29.0996	1.76672	1.56907	0.2283	2	2.6997
(0.01, 0.01)	255	29.0616	1.77139	1.56605	0.22305	2	2.6984
(0.01, 0.05)	262	29.114	1.76524	1.56694	0.22655	2	2.7039
(0.01, 0.1)	268	29.1092	1.76679	1.56885	0.22685	2	2.7032
(0.01, 0.2)	233	29.0781	1.77317	1.56899	0.21925	2	2.69985
(0.01, 0.3)	238	29.1072	1.77676	1.56757	0.21845	2	2.7015
(0.01, 0.4)	248	29.0669	1.76429	1.56859	0.228	2	2.6973
(0.01, 0.5)	206	29.13	1.77068	1.56634	0.21975	2	2.70365
(0.01, 0.6)	261	29.0946	1.7786	1.57167	0.217	2	2.7009
(0.01, 0.7)	219	29.1311	1.77289	1.56736	0.2188	2	2.7033
(0.01, 0.8)	261	29.0912	1.77051	1.57232	0.2249	2	2.70225
(0.01, 0.9)	258	29.0919	1.76983	1.56683	0.2234	2.00005	2.70005
(0.05, 0)	223	29.0863	1.76957	1.56649	0.2208	2	2.69985
(0.05, 0.01)	244	29.0906	1.77501	1.56698	0.21865	2	2.69955
(0.05, 0.05)	254	29.1067	1.77121	1.56666	0.22145	2	2.70145
(0.05, 0.1)	229	29.0779	1.77573	1.57002	0.21775	2	2.6979
(0.05, 0.2)	244	29.1192	1.76646	1.56778	0.22535	2	2.7026
(0.05, 0.3)	239	29.0565	1.77324	1.56822	0.2203	2.00005	2.6976
(0.05, 0.4)	272	29.1361	1.77011	1.56607	0.2251	2.00005	2.7074
(0.05, 0.5)	222	29.0983	1.77542	1.57057	0.2167	2	2.70055
(0.05, 0.6)	253	29.1275	1.77448	1.57069	0.22015	2	2.70645
(0.05, 0.7)	241	29.1365	1.77207	1.56696	0.22255	2	2.7064
(0.05, 0.8)	262	29.0948	1.77016	1.56599	0.2232	2	2.7012
(0.05, 0.9)	249	29.1069	1.76569	1.565	0.22815	2	2.7032
(0.1, 0)	231	29.093	1.77454	1.56968	0.2181	2	2.7
(0.1, 0.01)	248	29.1061	1.76797	1.56408	0.2265	2	2.70365
(0.1, 0.05)	256	29.1627	1.77034	1.57059	0.22375	2	2.7062
(0.1, 0.1)	261	29.0423	1.77288	1.56311	0.22235	2	2.69625
(0.1, 0.2)	231	29.1102	1.77219	1.56624	0.2217	2	2.70205
(0.1, 0.3)	225	29.0767	1.76711	1.56655	0.22265	2	2.6999
(0.1, 0.4)	245	29.0702	1.77286	1.5682	0.2223	2	2.69635
(0.1, 0.5)	261	29.0927	1.77346	1.56885	0.22065	2	2.7004
(0.1, 0.6)	240	29.1341	1.77146	1.5684	0.2194	2	2.7036
(0.1, 0.7)	240	29.1109	1.76861	1.56929	0.2238	2	2.7013
(0.1, 0.8)	234	29.0678	1.76585	1.56734	0.22435	2	2.69915
(0.1, 0.9)	281	29.0788	1.77457	1.56801	0.22135	2	2.69745
(0.2, 0)	228	29.1306	1.76486	1.56346	0.22585	2	2.70625
(0.2, 0.01)	226	29.0886	1.77171	1.56707	0.22105	2	2.6987
(0.2, 0.05)	234	29.1051	1.77616	1.56693	0.2184	2	2.70085
(0.2, 0.1)	252	29.0842	1.7668	1.56559	0.22755	2	2.6979
(0.2, 0.2)	239	29.0771	1.77097	1.56613	0.2217	2	2.6974
(0.2, 0.3)	264	29.1199	1.76927	1.57026	0.224	2.00005	2.7022
(0.2, 0.4)	233	29.073	1.76872	1.56128	0.22325	2	2.70025
(0.2, 0.5)	229	29.0126	1.77426	1.56785	0.2192	2	2.6949
(0.2, 0.6)	237	29.1238	1.77062	1.5673	0.22075	2	2.7054
(0.2, 0.7)	251	29.0671	1.76796	1.56645	0.22545	2	2.6992
(0.2, 0.8)	258	29.1424	1.77071	1.56934	0.2238	2	2.70635
(0.2, 0.9)	231	29.1014	1.76713	1.56669	0.22555	2	2.70365
(0.3, 0)	236	29.1088	1.77388	1.56934	0.2208	2	2.70135
(0.3, 0.01)	217	29.1404	1.77088	1.56812	0.22285	2	2.7039
(0.3, 0.05)	244	29.112	1.77694	1.56865	0.21895	2	2.70245
(0.3, 0.1)	255	29.1596	1.77655	1.56981	0.2185	2	2.7056
(0.3, 0.2)	242	29.1304	1.77211	1.57029	0.22315	2	2.70515
(0.3, 0.3)	249	29.0796	1.77033	1.56663	0.22305	2.00005	2.70015
(0.3, 0.4)	238	29.0876	1.77565	1.56972	0.2193	2	2.70005
(0.3, 0.5)	251	29.0746	1.77583	1.56927	0.21865	2	2.69835
(0.3, 0.6)	236	29.114	1.77167	1.56531	0.2232	2	2.7039
(0.3, 0.7)	210	29.0565	1.76512	1.56588	0.22415	2	2.69745

Tabulka C.3: Průměry kritérií, permutace, algoritmus Vega

Parametry	Počet nadprů-měrných	Bent skóre	$LP_{\max}$	$DP_{\max}$	SAC	faktor větvení	stupeň po-lynomu
(0.3, 0.8)	277	29.0954	1.76736	1.5647	0.2267	2	2.69965
(0.3, 0.9)	235	29.0928	1.7759	1.56908	0.21905	2	2.70135
(0.4, 0)	219	29.0484	1.77142	1.56632	0.2219	2.00005	2.69585
(0.4, 0.01)	257	29.1129	1.77039	1.56786	0.2231	2	2.7027
(0.4, 0.05)	224	29.1243	1.76633	1.57012	0.22635	2	2.7034
(0.4, 0.1)	259	29.0592	1.77032	1.56891	0.2227	2	2.69935
(0.4, 0.2)	245	29.1536	1.77521	1.5745	0.2187	2	2.7034
(0.4, 0.3)	252	29.0361	1.77575	1.5706	0.22065	2	2.69595
(0.4, 0.4)	256	29.0898	1.76491	1.56697	0.22935	2	2.70105
(0.4, 0.5)	242	29.0861	1.77253	1.57361	0.2204	2	2.70045
(0.4, 0.6)	261	29.087	1.77186	1.56675	0.22215	2	2.70045
(0.4, 0.7)	250	29.1431	1.77025	1.56947	0.2221	2	2.70645
(0.4, 0.8)	220	29.0956	1.76727	1.56475	0.22495	2	2.70315
(0.4, 0.9)	247	29.0397	1.77132	1.56845	0.22035	2	2.6967
(0.5, 0)	275	29.1199	1.76707	1.56903	0.228	2	2.7036
(0.5, 0.01)	213	29.0694	1.77092	1.56328	0.22235	2	2.7014
(0.5, 0.05)	228	29.0649	1.7743	1.56664	0.2191	2	2.6969
(0.5, 0.1)	235	29.0551	1.77212	1.56835	0.2232	2	2.6967
(0.5, 0.2)	213	29.0601	1.77445	1.57218	0.218	2	2.69595
(0.5, 0.3)	222	29.0524	1.76865	1.5628	0.2242	2	2.6977
(0.5, 0.4)	255	29.0839	1.76888	1.5698	0.2256	2	2.70095
(0.5, 0.5)	235	29.1096	1.76619	1.56762	0.22465	2	2.7003
(0.5, 0.6)	242	29.1059	1.77424	1.56962	0.22	2	2.70325
(0.5, 0.7)	258	29.1245	1.77612	1.57207	0.2191	2	2.70455
(0.5, 0.8)	242	29.1207	1.7639	1.56289	0.2285	2.00005	2.70485
(0.5, 0.9)	266	29.0924	1.7734	1.56674	0.22175	2	2.69995
(0.6, 0)	240	29.1114	1.77178	1.56324	0.2216	2	2.70395
(0.6, 0.01)	248	29.1267	1.77343	1.56535	0.22165	2	2.705
(0.6, 0.05)	247	29.1002	1.77204	1.56817	0.22185	2	2.69955
(0.6, 0.1)	258	29.1029	1.77823	1.56927	0.2165	2	2.6999
(0.6, 0.2)	246	29.1208	1.77087	1.56964	0.2227	2	2.7009
(0.6, 0.3)	226	29.1152	1.77853	1.56981	0.21675	2	2.70185
(0.6, 0.4)	220	29.0742	1.77199	1.56875	0.22055	2	2.69845
(0.6, 0.5)	239	29.0876	1.77541	1.56875	0.22015	2	2.69995
(0.6, 0.6)	243	29.0766	1.7695	1.56548	0.2242	2	2.69935
(0.6, 0.7)	256	29.0662	1.77282	1.56497	0.2227	2	2.6982
(0.6, 0.8)	247	29.0761	1.76264	1.56502	0.22975	2	2.7003
(0.6, 0.9)	227	29.0834	1.77362	1.56767	0.22	2	2.70015
(0.7, 0)	204	29.1338	1.77233	1.56672	0.2214	2	2.7052
(0.7, 0.01)	228	29.1622	1.77348	1.56871	0.21715	2	2.7057
(0.7, 0.05)	241	29.1212	1.77025	1.56654	0.2226	2	2.70335
(0.7, 0.1)	243	29.1109	1.77005	1.5675	0.2211	2	2.704
(0.7, 0.2)	254	29.0954	1.77017	1.56734	0.22285	2	2.70115
(0.7, 0.3)	248	29.074	1.7714	1.57196	0.22425	2	2.69885
(0.7, 0.4)	274	29.1005	1.77064	1.56925	0.22505	2	2.7017
(0.7, 0.5)	232	29.0354	1.76491	1.56384	0.22475	2	2.6954
(0.7, 0.6)	249	29.08	1.77292	1.56674	0.22175	2	2.7007
(0.7, 0.7)	235	29.1014	1.77275	1.56925	0.2193	2	2.70145
(0.7, 0.8)	251	29.0169	1.77403	1.56329	0.2191	2	2.69325
(0.7, 0.9)	241	29.0801	1.76964	1.56498	0.2233	2	2.702
(0.8, 0)	237	29.1153	1.76765	1.56664	0.22455	2	2.70165
(0.8, 0.01)	235	29.1161	1.77202	1.56669	0.2212	2	2.70415
(0.8, 0.05)	241	29.0526	1.76918	1.56262	0.2253	2	2.69785
(0.8, 0.1)	235	29.0664	1.77194	1.56641	0.21835	2	2.6978
(0.8, 0.2)	231	29.1066	1.77144	1.56535	0.2218	2	2.7021
(0.8, 0.3)	254	29.1132	1.77434	1.57228	0.2214	2	2.70245
(0.8, 0.4)	225	29.0748	1.77358	1.57036	0.2195	2	2.69795
(0.8, 0.5)	232	29.1287	1.76912	1.56596	0.22365	2	2.70605
(0.8, 0.6)	234	29.0373	1.77109	1.5688	0.22295	2	2.6963
(0.8, 0.7)	215	29.0447	1.77829	1.56951	0.2146	2	2.69565
(0.8, 0.8)	211	29.1233	1.77392	1.57304	0.2198	2	2.70305

Tabulka C.3: Průměry kritérií, permutace, algoritmus Vega

Parametry	Počet nadprů-měrných	Bent skóre	$LP_{\max}$	$DP_{\max}$	SAC	faktor větvení	stupeň po-lynomu
(0.8, 0.9)	241	29.0578	1.77422	1.56762	0.21975	2	2.69705
(0.9, 0)	248	29.0844	1.76714	1.56366	0.2251	2	2.70015
(0.9, 0.01)	242	29.0771	1.7692	1.56371	0.22285	2	2.69875
(0.9, 0.05)	260	29.1542	1.77186	1.57026	0.22455	2	2.705
(0.9, 0.1)	227	29.0964	1.76547	1.56702	0.2274	2	2.6995
(0.9, 0.2)	230	29.1508	1.76577	1.56529	0.22525	2	2.70745
(0.9, 0.3)	241	29.0558	1.77701	1.56706	0.218	2	2.69545
(0.9, 0.4)	235	29.0477	1.76832	1.56422	0.2262	2	2.6959
(0.9, 0.5)	233	29.0645	1.77625	1.56985	0.2181	2	2.69915
(0.9, 0.6)	248	29.0957	1.77071	1.56735	0.22295	2	2.7
(0.9, 0.7)	237	29.1671	1.77576	1.56927	0.2171	2	2.7071
(0.9, 0.8)	250	29.0908	1.76655	1.56632	0.2272	2	2.70085
(0.9, 0.9)	224	29.1147	1.77638	1.5717	0.21855	2	2.69975

Tabulka C.4: Průměry kritérií, permutace, algoritmus Spea

Parametry	Počet nadprů-měrných	Bent skóre	$LP_{\max}$	$DP_{\max}$	SAC	faktor větvení	stupeň po-lynomu
(0, 0)	6	31.8152	1.86648	1.90963	0.137763	2.00162	2.98379
(0, 0.01)	18	31.933	1.91158	1.94681	0.100812	2.00068	2.99256
(0, 0.05)	50	31.9452	1.95575	1.97402	0.0732861	2.00039	2.99448
(0, 0.1)	75	31.9258	1.96241	1.97169	0.122249	2.00081	2.99104
(0, 0.2)	94	31.8022	1.96327	1.96813	0.262222	2.00889	2.98222
(0, 0.3)	98	31.8311	1.96851	1.96739	0.387417	2.00662	2.98013
(0, 0.4)	102	31.7593	1.91963	1.94531	0.981481	2.03704	2.99074
(0, 0.5)	178	31.4703	1.95043	1.96665	0.989189	2.02162	2.96757
(0, 0.6)	131	31.3869	1.92701	1.94723	0.985401	2.0292	2.92701
(0, 0.7)	255	31.5204	1.93056	1.94058	0.996283	2.02974	2.95539
(0, 0.8)	256	31.834	1.97793	1.98583	0.973585	2.02642	2.98491
(0, 0.9)	188	31.8442	1.9304	1.95941	0.964824	2.03518	2.9799
(0.01, 0)	3620	31.8773	1.97965	1.98323	1.01873	2.00326	2.9905
(0.01, 0.01)	3279	31.9511	1.98315	1.9902	1.0018	2.01171	2.9955
(0.01, 0.05)	3498	31.9411	1.98537	1.98948	1.00535	2.0062	2.99549
(0.01, 0.1)	3271	31.9376	1.98403	1.98922	1.00151	2.00724	2.99457
(0.01, 0.2)	3455	31.9313	1.98115	1.98724	1.00371	2.0077	2.99401
(0.01, 0.3)	3249	31.9069	1.96494	1.97644	1.01981	2.00961	2.99189
(0.01, 0.4)	3325	31.8443	1.967	1.97938	1.00558	2.0097	2.98971
(0.01, 0.5)	3115	31.8915	1.9694	1.97885	1.00031	2.01501	2.98968
(0.01, 0.6)	3112	31.8939	1.96874	1.97806	1.00626	2.01628	2.99061
(0.01, 0.7)	3200	31.899	1.98107	1.98629	1.00983	2.00768	2.99109
(0.01, 0.8)	2958	31.905	1.97481	1.97977	0.99868	2.01386	2.99241
(0.01, 0.9)	2914	31.7382	1.96012	1.97462	1.01269	2.00835	2.98197
(0.05, 0)	1897	31.7976	1.94876	1.96668	0.984779	2.0274	2.98021
(0.05, 0.01)	1903	31.7406	1.93595	1.95939	0.999496	2.02217	2.97229
(0.05, 0.05)	2040	31.7459	1.9421	1.96271	0.988213	2.02593	2.97265
(0.05, 0.1)	1993	31.6684	1.9334	1.95616	0.9885	2.03019	2.96933
(0.05, 0.2)	1828	31.6467	1.91618	1.94595	0.990201	2.03507	2.96854
(0.05, 0.3)	1778	31.7066	1.9398	1.96284	0.982172	2.02971	2.96813
(0.05, 0.4)	1788	31.753	1.9486	1.96503	0.994606	2.02104	2.97843
(0.05, 0.5)	1723	31.6501	1.92824	1.95088	0.99005	2.0293	2.96794
(0.05, 0.6)	1679	31.5711	1.92401	1.94907	0.988669	2.03059	2.96261
(0.05, 0.7)	1649	31.4917	1.90959	1.93868	0.986294	2.03712	2.95545
(0.05, 0.8)	1571	31.5943	1.91954	1.94602	0.986136	2.03496	2.96745
(0.05, 0.9)	1530	31.6511	1.9275	1.95281	0.980757	2.03662	2.96276
(0.1, 0)	1209	31.472	1.90428	1.9359	0.974379	2.04503	2.95186
(0.1, 0.01)	1192	31.5212	1.90281	1.93413	0.96305	2.05031	2.95283
(0.1, 0.05)	1163	31.6326	1.91623	1.94288	0.977291	2.03974	2.96675
(0.1, 0.1)	1151	31.2343	1.88231	1.91345	0.969502	2.05377	2.9382
(0.1, 0.2)	1143	31.4258	1.9164	1.93895	0.971831	2.0406	2.95278

Tabulka C.4: Průměry kritérií, permutace, algoritmus Spea

Parametry	Počet nadprů-měrných	Bent skóre	$LP_{\max}$	$DP_{\max}$	SAC	faktor větvení	stupeň po-lynomu
(0.1, 0.3)	1147	31.4044	1.88997	1.92272	0.958671	2.05592	2.94246
(0.1, 0.4)	1084	31.1228	1.87289	1.90822	0.957374	2.06138	2.92157
(0.1, 0.5)	1192	31.2624	1.88072	1.91706	0.96972	2.0528	2.93634
(0.1, 0.6)	1175	31.4022	1.8963	1.92831	0.963434	2.05246	2.94674
(0.1, 0.7)	1166	31.3077	1.88688	1.92071	0.963521	2.0563	2.95004
(0.1, 0.8)	1125	31.3018	1.89336	1.92616	0.970978	2.04809	2.94196
(0.1, 0.9)	1170	31.1571	1.86779	1.9091	0.976434	2.05263	2.93166
(0.2, 0)	1118	31.3183	1.89549	1.92632	0.969089	2.05013	2.93651
(0.2, 0.01)	1152	31.4182	1.88962	1.92422	0.966962	2.05238	2.94762
(0.2, 0.05)	1192	31.3483	1.88055	1.91525	0.959846	2.061	2.94131
(0.2, 0.1)	1118	31.4456	1.89403	1.92145	0.9734	2.04988	2.94597
(0.2, 0.2)	1202	31.5047	1.9124	1.93968	0.979624	2.03997	2.96003
(0.2, 0.3)	1177	31.3183	1.89327	1.92562	0.975455	2.04751	2.94141
(0.2, 0.4)	1232	31.273	1.89018	1.91904	0.975867	2.04751	2.93891
(0.2, 0.5)	1094	31.1804	1.881	1.91507	0.962901	2.05818	2.92327
(0.2, 0.6)	1104	31.4801	1.8997	1.92904	0.974598	2.04572	2.95089
(0.2, 0.7)	1157	31.1958	1.8756	1.9187	0.971223	2.05356	2.93125
(0.2, 0.8)	1114	31.3358	1.89965	1.92611	0.97076	2.04929	2.934
(0.2, 0.9)	1134	31.294	1.89084	1.92192	0.971335	2.05078	2.93694
(0.3, 0)	1166	31.3838	1.89385	1.92105	0.977707	2.04777	2.94825
(0.3, 0.01)	1217	31.4131	1.90156	1.92869	0.98	2.04231	2.94923
(0.3, 0.05)	1189	31.332	1.89872	1.92824	0.976341	2.04338	2.94006
(0.3, 0.1)	1148	31.3317	1.90318	1.92892	0.97718	2.04401	2.94132
(0.3, 0.2)	1190	31.2781	1.89118	1.92136	0.967187	2.05234	2.94375
(0.3, 0.3)	1132	31.3344	1.89477	1.9258	0.980182	2.04377	2.9455
(0.3, 0.4)	1240	31.4909	1.90309	1.92999	0.972054	2.04683	2.95242
(0.3, 0.5)	1180	31.3792	1.88792	1.92317	0.974036	2.04957	2.94335
(0.3, 0.6)	1138	31.4495	1.89336	1.92767	0.964667	2.05012	2.95234
(0.3, 0.7)	1150	31.3041	1.8819	1.92008	0.979887	2.04827	2.94047
(0.3, 0.8)	1109	31.2116	1.89271	1.92308	0.969773	2.04954	2.93871
(0.3, 0.9)	1180	31.3006	1.88922	1.91978	0.966168	2.0535	2.94099
(0.4, 0)	1112	31.1746	1.87882	1.9137	0.954281	2.06234	2.92685
(0.4, 0.01)	1198	31.3898	1.89663	1.92612	0.968168	2.05124	2.94255
(0.4, 0.05)	1242	31.4286	1.91317	1.93788	0.980243	2.03799	2.95517
(0.4, 0.1)	1270	31.4398	1.90444	1.93519	0.97561	2.04361	2.94826
(0.4, 0.2)	1207	31.3866	1.90782	1.93418	0.992206	2.03507	2.95246
(0.4, 0.3)	1229	31.3791	1.90112	1.93088	0.977879	2.04424	2.94813
(0.4, 0.4)	1206	31.3364	1.88944	1.92485	0.972222	2.04861	2.94522
(0.4, 0.5)	1188	31.3806	1.89877	1.92674	0.977147	2.04492	2.94799
(0.4, 0.6)	1175	31.3958	1.90335	1.93149	0.984038	2.0399	2.94812
(0.4, 0.7)	1180	31.4024	1.89607	1.93054	0.988095	2.03889	2.94683
(0.4, 0.8)	1226	31.2851	1.89562	1.92528	0.978659	2.04421	2.94893
(0.4, 0.9)	1142	31.3107	1.89009	1.92298	0.9861	2.0417	2.94031
(0.5, 0)	1198	31.3307	1.90036	1.92622	0.989054	2.03753	2.94605
(0.5, 0.01)	1246	31.4548	1.91715	1.9399	0.977186	2.03954	2.95817
(0.5, 0.05)	1296	31.5169	1.92417	1.9452	0.979472	2.03592	2.95821
(0.5, 0.1)	1188	31.4805	1.9161	1.93762	0.981732	2.03813	2.95631
(0.5, 0.2)	1201	31.6643	1.92919	1.94721	0.980998	2.03484	2.96912
(0.5, 0.3)	1194	31.5024	1.91171	1.93751	0.977883	2.03949	2.95419
(0.5, 0.4)	1219	31.5304	1.90625	1.93694	0.969254	2.04766	2.95465
(0.5, 0.5)	1182	31.3419	1.88434	1.91654	0.982786	2.04695	2.94836
(0.5, 0.6)	1191	31.3333	1.89491	1.92344	0.972613	2.04851	2.94444
(0.5, 0.7)	1079	31.264	1.88832	1.92186	0.980155	2.04573	2.93788
(0.5, 0.8)	1258	31.4571	1.90272	1.93526	0.977629	2.04251	2.95227
(0.5, 0.9)	1199	31.386	1.9099	1.93475	0.978774	2.04009	2.94969
(0.6, 0)	1321	31.4156	1.91278	1.93722	0.98927	2.03362	2.95136
(0.6, 0.01)	1289	31.6132	1.92355	1.94515	0.993382	2.03088	2.97279
(0.6, 0.05)	1349	31.571	1.93303	1.94656	0.985159	2.0311	2.96184
(0.6, 0.1)	1248	31.5201	1.9111	1.93969	0.979561	2.03861	2.95761
(0.6, 0.2)	1353	31.4485	1.92027	1.93994	0.984583	2.03434	2.95655
(0.6, 0.3)	1224	31.5633	1.91631	1.94209	0.977623	2.04012	2.95988

Tabulka C.4: Průměry kritérií, permutace, algoritmus Spea

Parametry	Počet nadprů-měrných	Bent skóre	$LP_{\max}$	$DP_{\max}$	SAC	faktor větvení	stupeň po-lynomu
(0.6, 0.4)	1244	31.3489	1.90257	1.93138	0.976639	2.04446	2.94725
(0.6, 0.5)	1262	31.521	1.91983	1.94269	0.98051	2.03748	2.95652
(0.6, 0.6)	1233	31.3275	1.89923	1.92835	0.984043	2.04027	2.94453
(0.6, 0.7)	1255	31.4695	1.91051	1.94033	0.976639	2.03919	2.94876
(0.6, 0.8)	1175	31.5242	1.91499	1.94143	0.97504	2.03945	2.95491
(0.6, 0.9)	1231	31.5244	1.91364	1.93784	0.981679	2.03969	2.95725
(0.7, 0)	1407	31.6166	1.93728	1.95446	0.993882	2.02515	2.96329
(0.7, 0.01)	1327	31.5836	1.9184	1.93982	0.997865	2.0306	2.9694
(0.7, 0.05)	1396	31.6241	1.92999	1.95614	0.990398	2.02675	2.96845
(0.7, 0.1)	1359	31.5182	1.92207	1.94466	1	2.02657	2.96224
(0.7, 0.2)	1388	31.5889	1.92693	1.94827	0.987646	2.0302	2.96843
(0.7, 0.3)	1383	31.5296	1.92416	1.94695	0.989684	2.03095	2.9608
(0.7, 0.4)	1343	31.529	1.92143	1.94319	0.994342	2.029	2.96393
(0.7, 0.5)	1306	31.388	1.90225	1.93263	0.983441	2.03816	2.9532
(0.7, 0.6)	1307	31.4701	1.90812	1.93634	0.979122	2.04104	2.95392
(0.7, 0.7)	1256	31.3555	1.91108	1.93166	0.974551	2.04266	2.9491
(0.7, 0.8)	1251	31.3588	1.90053	1.93052	0.982772	2.04045	2.95206
(0.7, 0.9)	1293	31.4668	1.91019	1.93954	0.978086	2.04091	2.95909
(0.8, 0)	1801	31.7368	1.95591	1.96802	0.990291	2.01942	2.97411
(0.8, 0.01)	1627	31.7011	1.95094	1.96417	0.999406	2.01901	2.97683
(0.8, 0.05)	1703	31.7633	1.95699	1.96996	0.996577	2.01654	2.98118
(0.8, 0.1)	1741	31.7273	1.94279	1.96169	0.998894	2.01991	2.97677
(0.8, 0.2)	1679	31.7367	1.95568	1.96827	0.998268	2.0179	2.97806
(0.8, 0.3)	1694	31.6034	1.93853	1.95405	0.994915	2.02599	2.97119
(0.8, 0.4)	1566	31.662	1.94247	1.9589	0.993252	2.02454	2.97117
(0.8, 0.5)	1579	31.7066	1.93197	1.9523	0.998185	2.0248	2.97822
(0.8, 0.6)	1525	31.4651	1.91457	1.93625	0.998765	2.03212	2.96109
(0.8, 0.7)	1450	31.3748	1.90303	1.92908	0.981935	2.04323	2.94774
(0.8, 0.8)	1459	31.6573	1.93029	1.95186	0.988251	2.03003	2.96932
(0.8, 0.9)	1446	31.5485	1.91768	1.94061	0.985583	2.03408	2.96527
(0.9, 0)	3925	31.8829	1.97836	1.98408	1.00527	2.00752	2.99072
(0.9, 0.01)	3905	31.9025	1.97919	1.98484	1.00884	2.00555	2.99268
(0.9, 0.05)	3967	31.8824	1.97258	1.98166	1.01114	2.0047	2.99282
(0.9, 0.1)	3939	31.8473	1.97581	1.98005	1.00522	2.0092	2.98707
(0.9, 0.2)	3885	31.8229	1.97426	1.97774	1.0078	2.0083	2.98792
(0.9, 0.3)	3413	31.9105	1.97617	1.98352	0.997985	2.01123	2.99223
(0.9, 0.4)	3420	31.7659	1.95236	1.9672	0.998867	2.01757	2.98158
(0.9, 0.5)	3308	31.7604	1.96513	1.97512	1.00471	2.01384	2.98293
(0.9, 0.6)	3030	31.7452	1.96138	1.96905	0.99872	2.01697	2.98079
(0.9, 0.7)	2776	31.7516	1.94276	1.96347	1.00521	2.01841	2.97985
(0.9, 0.8)	2732	31.6774	1.95268	1.96482	0.996105	2.01912	2.97415
(0.9, 0.9)	2574	31.7431	1.95125	1.96487	1.00038	2.01727	2.98198

Tabulka C.5: Průměry kritérií, i686, algoritmus Vega

Parametry	Počet nadprů-měrných	Bent skóre	$LP_{\max}$	$DP_{\max}$	SAC	faktor větvení	stupeň po-lynomu	bijektivní skóre
(0, 0)	419	12.6303	0.0493762	0.324886	0.0036	1.0078	1.2864	8.66591
(0, 0.01)	841	12.8836	0.0737946	0.374788	0.0011	1.0024	1.38895	8.65791
(0, 0.05)	1320	14.013	0.137044	0.487981	0.00465	1.0038	1.52515	8.81425
(0, 0.1)	1577	14.535	0.146307	0.505345	0.00315	1.00315	1.5358	8.86941
(0, 0.2)	1383	14.5378	0.164639	0.535781	0.0049	1.00385	1.6117	8.80803
(0, 0.3)	1432	14.4263	0.147757	0.524942	0.006	1.0066	1.5205	8.8153
(0, 0.4)	1388	14.6973	0.146998	0.530909	0.00825	1.0058	1.5472	8.74919
(0, 0.5)	1466	14.496	0.167305	0.541135	0.0071	1.0044	1.6223	8.76936
(0, 0.6)	1449	14.5032	0.167018	0.552414	0.0097	1.00635	1.59815	8.84758
(0, 0.7)	1183	14.214	0.143136	0.526117	0.0074	1.0045	1.55945	8.72951
(0, 0.8)	1492	14.462	0.164935	0.57263	0.0066	1.00505	1.59535	8.85272
(0, 0.9)	1207	14.2762	0.134465	0.543235	0.008	1.0055	1.5516	8.71266

Tabulka C.5: Průměry kritérií, i686, algoritmus Vega

Parametry	Počet nadprů-měrných	Bent skóre	$LP_{\max}$	$DP_{\max}$	SAC	faktor větvení	stupeň po-lynomu	bijektivní skóre
(0.01, 0)	2090	14.8595	0.230006	0.582026	0.01855	1.00945	1.68065	8.93769
(0.01, 0.01)	2235	15.2115	0.250771	0.61361	0.0176	1.0081	1.6947	8.94246
(0.01, 0.05)	2245	14.9729	0.232193	0.59613	0.01265	1.0064	1.6858	8.93843
(0.01, 0.1)	2019	14.9431	0.222862	0.594266	0.0186	1.0089	1.66	8.91286
(0.01, 0.2)	2119	14.7499	0.239441	0.609186	0.0172	1.00815	1.68565	8.96119
(0.01, 0.3)	2241	14.5308	0.258395	0.613337	0.01625	1.00845	1.7059	8.98077
(0.01, 0.4)	2094	14.6706	0.235223	0.611662	0.01515	1.0071	1.6532	8.96148
(0.01, 0.5)	2111	14.4443	0.25501	0.611243	0.01235	1.00595	1.68625	8.93314
(0.01, 0.6)	1916	14.3773	0.226488	0.60775	0.01605	1.00795	1.62675	8.90944
(0.01, 0.7)	2100	14.2713	0.244984	0.616289	0.01775	1.0076	1.65745	8.94812
(0.01, 0.8)	2062	14.2765	0.244592	0.626198	0.0158	1.00755	1.6475	8.97274
(0.01, 0.9)	2267	14.3286	0.25056	0.632649	0.0161	1.0079	1.6675	8.97101
(0.05, 0)	1792	14.7377	0.200645	0.612951	0.0205	1.00815	1.5946	8.8841
(0.05, 0.01)	1824	14.612	0.207684	0.619157	0.01455	1.0079	1.59835	8.90077
(0.05, 0.05)	1788	14.4837	0.201327	0.610494	0.0158	1.0082	1.5895	8.91742
(0.05, 0.1)	1737	14.7148	0.197101	0.622911	0.0145	1.007	1.5964	8.86941
(0.05, 0.2)	1647	14.4923	0.195063	0.612602	0.0154	1.00735	1.60205	8.83579
(0.05, 0.3)	1632	14.3138	0.189944	0.627507	0.0182	1.0093	1.58875	8.89408
(0.05, 0.4)	1658	14.2334	0.195787	0.632367	0.0151	1.00725	1.5808	8.88947
(0.05, 0.5)	1690	14.4731	0.192377	0.627101	0.01485	1.00725	1.5927	8.87096
(0.05, 0.6)	1693	14.3915	0.191093	0.630964	0.01475	1.0069	1.5909	8.8617
(0.05, 0.7)	1729	14.5275	0.196836	0.636156	0.0144	1.0069	1.59445	8.87511
(0.05, 0.8)	1585	14.3299	0.18495	0.628685	0.0156	1.00805	1.5512	8.84313
(0.05, 0.9)	1744	14.2747	0.193142	0.645326	0.01575	1.0071	1.5568	8.87166
(0.1, 0)	1614	14.4416	0.185894	0.631482	0.01575	1.00715	1.5739	8.86976
(0.1, 0.01)	1582	14.3235	0.178188	0.624482	0.01525	1.0082	1.53745	8.86184
(0.1, 0.05)	1541	14.2303	0.176346	0.622984	0.01555	1.00785	1.5369	8.87199
(0.1, 0.1)	1582	14.3299	0.177478	0.6283	0.0149	1.0085	1.53985	8.87216
(0.1, 0.2)	1578	14.3955	0.17727	0.626047	0.01695	1.00805	1.54185	8.85862
(0.1, 0.3)	1492	14.2537	0.171157	0.622081	0.0153	1.00785	1.5371	8.84724
(0.1, 0.4)	1482	14.2317	0.169439	0.626285	0.01595	1.00785	1.5142	8.84923
(0.1, 0.5)	1583	14.3382	0.179165	0.639447	0.01515	1.0079	1.56405	8.89026
(0.1, 0.6)	1543	14.1934	0.178254	0.631593	0.01405	1.00855	1.54885	8.83656
(0.1, 0.7)	1601	14.2867	0.176871	0.636985	0.01685	1.00785	1.55505	8.8421
(0.1, 0.8)	1440	14.2545	0.169715	0.633629	0.0159	1.00705	1.52435	8.84951
(0.1, 0.9)	1461	14.2223	0.170525	0.624661	0.01495	1.00755	1.53165	8.82479
(0.2, 0)	1403	14.22	0.163277	0.625398	0.01545	1.0079	1.5257	8.82286
(0.2, 0.01)	1391	14.195	0.162187	0.628851	0.01445	1.0079	1.51985	8.82728
(0.2, 0.05)	1409	14.1328	0.16257	0.626793	0.0155	1.0073	1.5255	8.85045
(0.2, 0.1)	1381	14.2402	0.158545	0.631879	0.0155	1.0075	1.5028	8.83442
(0.2, 0.2)	1466	14.2734	0.160531	0.627505	0.01635	1.009	1.5193	8.8304
(0.2, 0.3)	1406	14.187	0.158971	0.624876	0.01525	1.00775	1.5105	8.82286
(0.2, 0.4)	1339	14.3216	0.155971	0.628508	0.016	1.0078	1.50255	8.7868
(0.2, 0.5)	1350	14.295	0.15446	0.628197	0.0161	1.008	1.4987	8.82314
(0.2, 0.6)	1382	14.3344	0.160738	0.62611	0.0161	1.00835	1.5064	8.84212
(0.2, 0.7)	1406	14.375	0.157418	0.62417	0.01565	1.0073	1.5086	8.82532
(0.2, 0.8)	1392	14.188	0.159276	0.628969	0.01405	1.00665	1.51225	8.82403
(0.2, 0.9)	1417	14.4213	0.161452	0.629545	0.01525	1.00785	1.5192	8.8228
(0.3, 0)	1351	14.1558	0.154054	0.627052	0.015	1.00755	1.48405	8.82434
(0.3, 0.01)	1341	14.2216	0.156109	0.626192	0.01385	1.00715	1.5026	8.81295
(0.3, 0.05)	1418	14.273	0.159782	0.629774	0.0129	1.007	1.5081	8.82405
(0.3, 0.1)	1324	14.2154	0.154222	0.625236	0.01615	1.00815	1.4993	8.80203
(0.3, 0.2)	1327	14.2341	0.154023	0.625816	0.01585	1.0075	1.4934	8.82301
(0.3, 0.3)	1331	14.1708	0.157187	0.625539	0.01415	1.0076	1.48815	8.82147
(0.3, 0.4)	1341	14.2731	0.155317	0.62035	0.0124	1.0066	1.5043	8.81062
(0.3, 0.5)	1348	14.2345	0.151656	0.630097	0.01795	1.00895	1.48555	8.81899
(0.3, 0.6)	1368	14.3742	0.153289	0.629305	0.01585	1.008	1.50555	8.80544
(0.3, 0.7)	1308	14.2416	0.15493	0.625636	0.015	1.00785	1.4898	8.82554
(0.3, 0.8)	1269	14.1848	0.147629	0.623236	0.01615	1.0083	1.494	8.80599
(0.3, 0.9)	1269	14.2253	0.152548	0.625079	0.0142	1.00805	1.49945	8.84018
(0.4, 0)	1334	14.1849	0.151281	0.620415	0.01665	1.0083	1.4893	8.79426

Tabulka C.5: Průměry kritérií, i686, algoritmus Vega

Parametry	Počet nadprů-měrných	Bent skóre	$LP_{\max}$	$DP_{\max}$	SAC	faktor větvení	stupeň po-lynomu	bijektivní skóre
(0.4, 0.01)	1285	14.3567	0.148638	0.621856	0.01595	1.008	1.49015	8.77446
(0.4, 0.05)	1272	14.213	0.148006	0.622738	0.0154	1.00835	1.4853	8.78487
(0.4, 0.1)	1294	14.3264	0.148964	0.62541	0.01675	1.0079	1.48885	8.8087
(0.4, 0.2)	1302	14.3165	0.151179	0.625127	0.01455	1.0077	1.47735	8.80916
(0.4, 0.3)	1307	14.1397	0.148805	0.622818	0.0151	1.00785	1.49745	8.80978
(0.4, 0.4)	1375	14.2296	0.15179	0.62416	0.016	1.00815	1.4965	8.79763
(0.4, 0.5)	1233	14.2593	0.144546	0.622016	0.01605	1.00815	1.46785	8.81326
(0.4, 0.6)	1214	14.2238	0.150097	0.621659	0.01605	1.0079	1.4861	8.8119
(0.4, 0.7)	1319	14.4045	0.150333	0.626036	0.01575	1.00735	1.4974	8.78802
(0.4, 0.8)	1302	14.3603	0.146846	0.626248	0.014	1.0071	1.49025	8.792
(0.4, 0.9)	1258	14.4016	0.147335	0.62891	0.0153	1.0068	1.4788	8.80851
(0.5, 0)	1346	14.2633	0.150643	0.624045	0.014	1.00765	1.4785	8.81854
(0.5, 0.01)	1299	14.2453	0.149206	0.624082	0.0142	1.00765	1.47835	8.78869
(0.5, 0.05)	1255	14.3086	0.147116	0.61959	0.01535	1.0075	1.4727	8.78971
(0.5, 0.1)	1244	14.2477	0.145269	0.621021	0.0173	1.0084	1.48305	8.83297
(0.5, 0.2)	1311	14.3256	0.147214	0.62033	0.0163	1.0075	1.46935	8.80645
(0.5, 0.3)	1383	14.4265	0.152191	0.622405	0.0127	1.0072	1.485	8.81154
(0.5, 0.4)	1275	14.2286	0.150129	0.626889	0.0153	1.00665	1.48695	8.80644
(0.5, 0.5)	1330	14.2497	0.145191	0.622602	0.01435	1.0073	1.47135	8.7791
(0.5, 0.6)	1264	14.37	0.143063	0.615714	0.01565	1.00755	1.4887	8.78249
(0.5, 0.7)	1314	14.3779	0.1454	0.62166	0.01625	1.00755	1.4843	8.79776
(0.5, 0.8)	1304	14.3387	0.151205	0.624093	0.0174	1.0087	1.47645	8.80134
(0.5, 0.9)	1283	14.227	0.145791	0.626205	0.0172	1.00845	1.48005	8.80302
(0.6, 0)	1246	14.224	0.146275	0.622952	0.01705	1.00775	1.4781	8.80704
(0.6, 0.01)	1307	14.3003	0.146432	0.619713	0.0166	1.0082	1.4806	8.82179
(0.6, 0.05)	1319	14.4542	0.145305	0.627825	0.016	1.00755	1.4815	8.79312
(0.6, 0.1)	1277	14.2807	0.145953	0.624647	0.0174	1.00815	1.4834	8.80084
(0.6, 0.2)	1287	14.4293	0.146025	0.623657	0.01485	1.00755	1.48965	8.79081
(0.6, 0.3)	1261	14.2053	0.146788	0.628021	0.01535	1.0071	1.4799	8.80842
(0.6, 0.4)	1272	14.3864	0.147385	0.62776	0.0128	1.00635	1.47	8.81028
(0.6, 0.5)	1293	14.2973	0.147572	0.628363	0.01555	1.0078	1.47485	8.81709
(0.6, 0.6)	1215	14.2805	0.144294	0.624686	0.01455	1.00705	1.47675	8.78228
(0.6, 0.7)	1306	14.4218	0.142166	0.625978	0.01565	1.0074	1.4738	8.80265
(0.6, 0.8)	1215	14.4564	0.142099	0.621734	0.01545	1.00705	1.4825	8.78064
(0.6, 0.9)	1264	14.2551	0.143658	0.624504	0.01485	1.00785	1.47235	8.78722
(0.7, 0)	1211	14.1601	0.141552	0.617201	0.01355	1.00695	1.46735	8.78319
(0.7, 0.01)	1216	14.2234	0.144596	0.620401	0.01525	1.0084	1.48005	8.77432
(0.7, 0.05)	1273	14.3476	0.145157	0.622029	0.01645	1.00835	1.4943	8.83127
(0.7, 0.1)	1211	14.1821	0.142052	0.618485	0.0173	1.0083	1.4769	8.78492
(0.7, 0.2)	1306	14.3417	0.147134	0.623205	0.01405	1.00825	1.4882	8.82108
(0.7, 0.3)	1192	14.2772	0.14129	0.621637	0.01445	1.0067	1.47265	8.79581
(0.7, 0.4)	1239	14.3508	0.144837	0.624406	0.01695	1.0082	1.47655	8.82624
(0.7, 0.5)	1254	14.4152	0.142327	0.617883	0.01315	1.0062	1.46595	8.79552
(0.7, 0.6)	1246	14.1953	0.14084	0.619501	0.0161	1.0083	1.4772	8.8171
(0.7, 0.7)	1245	14.2963	0.143429	0.617321	0.0138	1.00755	1.48405	8.78786
(0.7, 0.8)	1272	14.2516	0.145	0.617426	0.0165	1.0096	1.46865	8.80404
(0.7, 0.9)	1282	14.2597	0.143759	0.618734	0.01705	1.00865	1.48645	8.80268
(0.8, 0)	1253	14.2392	0.146115	0.624325	0.01645	1.0081	1.4831	8.80181
(0.8, 0.01)	1313	14.2161	0.142104	0.620041	0.01455	1.0079	1.46565	8.77654
(0.8, 0.05)	1279	14.3654	0.143273	0.623946	0.01485	1.00795	1.471	8.7856
(0.8, 0.1)	1256	14.3656	0.145869	0.623651	0.0135	1.0062	1.47475	8.77674
(0.8, 0.2)	1324	14.3084	0.146631	0.627616	0.01595	1.00785	1.47475	8.8041
(0.8, 0.3)	1287	14.2041	0.144229	0.617816	0.014	1.0068	1.4831	8.7809
(0.8, 0.4)	1220	14.3475	0.141765	0.623154	0.0173	1.0086	1.4815	8.78926
(0.8, 0.5)	1308	14.4075	0.144431	0.625365	0.01575	1.00755	1.4834	8.82091
(0.8, 0.6)	1256	14.2881	0.142976	0.618915	0.0157	1.008	1.47865	8.80071
(0.8, 0.7)	1292	14.3297	0.146885	0.62325	0.0123	1.007	1.4772	8.80384
(0.8, 0.8)	1267	14.2539	0.146672	0.621688	0.0148	1.0087	1.47445	8.79241
(0.8, 0.9)	1275	14.2749	0.142833	0.619507	0.01615	1.00735	1.47675	8.79309
(0.9, 0)	1236	14.3315	0.142124	0.623103	0.0158	1.0088	1.4757	8.79284
(0.9, 0.01)	1321	14.2367	0.139682	0.62572	0.017	1.0079	1.47695	8.77304

Tabulka C.5: Průměry kritérií, i686, algoritmus Vega

Parametry	Počet nadprůměrných	Bent skóre	$LP_{\max}$	$DP_{\max}$	SAC	faktor větvení	stupeň polynomu	bijektivní skóre
(0.9, 0.05)	1230	14.3538	0.140955	0.619987	0.016	1.0074	1.4835	8.78567
(0.9, 0.1)	1268	14.296	0.144826	0.624311	0.0135	1.007	1.4788	8.79916
(0.9, 0.2)	1271	14.3572	0.141796	0.620855	0.0156	1.00705	1.4693	8.80086
(0.9, 0.3)	1277	14.3651	0.142911	0.617651	0.0164	1.00775	1.47215	8.77289
(0.9, 0.4)	1362	14.2927	0.146754	0.623784	0.01575	1.00765	1.4743	8.81749
(0.9, 0.5)	1259	14.3322	0.141612	0.618476	0.0149	1.0069	1.4732	8.79166
(0.9, 0.6)	1270	14.1509	0.143165	0.62204	0.01605	1.0084	1.4746	8.80181
(0.9, 0.7)	1208	14.2693	0.14155	0.621056	0.01515	1.00775	1.47505	8.78262
(0.9, 0.8)	1327	14.3679	0.146031	0.621656	0.0167	1.00845	1.47285	8.82384
(0.9, 0.9)	1226	14.3595	0.142036	0.624768	0.0158	1.0074	1.4741	8.79375

Tabulka C.6: Průměry kritérií, i686, algoritmus Spea

Parametry	Počet nadprůměrných	Bent skóre	$LP_{\max}$	$DP_{\max}$	SAC	faktor větvení	stupeň polynomu	bijektivní skóre
(0.01, 0)	1	23.1407	0.459712	0.887365	0.430258	1.53872	2.3412	12.4926
(0.01, 0.01)	2	23.2024	0.489704	0.927982	0.429515	1.56948	2.4177	12.5127
(0.01, 0.05)	9	23.6577	0.500453	0.939769	0.572829	1.65366	2.45818	12.7218
(0.01, 0.1)	9	23.4524	0.539483	0.974619	0.5258	1.668	2.4548	12.764
(0.01, 0.2)	15	24.6757	0.5833	1.02837	0.571972	1.6959	2.47728	12.8469
(0.01, 0.3)	28	25.1036	0.583768	1.04205	0.660177	1.72782	2.49338	12.8637
(0.01, 0.4)	28	25.1598	0.575463	1.06998	0.602123	1.74089	2.4968	12.9703
(0.01, 0.5)	35	25.4302	0.611593	1.0735	0.633086	1.73263	2.53555	12.9575
(0.01, 0.6)	32	25.3597	0.573307	1.07224	0.685548	1.74299	2.54984	12.9711
(0.01, 0.7)	30	25.2828	0.592483	1.06864	0.65716	1.73649	2.56477	12.9891
(0.01, 0.8)	53	25.8222	0.57996	1.07395	0.7102	1.7604	2.5212	13.0057
(0.01, 0.9)	48	25.9057	0.588738	1.08988	0.696575	1.77068	2.52794	12.9785
(0.05, 0)	16	24.5612	0.571332	1.01913	0.6992	1.7176	2.4788	12.7454
(0.05, 0.01)	13	25.0454	0.57721	1.03009	0.6618	1.7188	2.474	12.8005
(0.05, 0.05)	17	24.9994	0.573894	1.04027	0.735925	1.76037	2.48988	12.8415
(0.05, 0.1)	17	25.0764	0.593161	1.05641	0.696139	1.74415	2.5225	12.8445
(0.05, 0.2)	26	25.1342	0.61102	1.06547	0.699579	1.73603	2.51472	12.8714
(0.05, 0.3)	35	25.4356	0.582555	1.04099	0.7818	1.7924	2.502	12.954
(0.05, 0.4)	36	25.3735	0.606482	1.07914	0.786833	1.78563	2.555	12.9708
(0.05, 0.5)	27	25.5046	0.596639	1.06928	0.768815	1.81125	2.51721	13.0421
(0.05, 0.6)	48	25.3573	0.614404	1.08047	0.755351	1.78196	2.54731	12.9793
(0.05, 0.7)	57	25.5078	0.620943	1.10155	0.772964	1.78119	2.56418	13.0121
(0.05, 0.8)	47	25.675	0.612794	1.09884	0.712136	1.77914	2.55466	13.0128
(0.05, 0.9)	45	25.0076	0.596171	1.09263	0.755956	1.7984	2.57698	13.0482
(0.1, 0)	21	24.8294	0.585685	1.04014	0.680736	1.71614	2.4767	12.8006
(0.1, 0.01)	15	24.7961	0.57756	1.0334	0.690967	1.73363	2.46765	12.8234
(0.1, 0.05)	15	24.3133	0.590666	1.01708	0.733534	1.73574	2.49109	12.799
(0.1, 0.1)	20	25.025	0.574518	1.02612	0.711627	1.73004	2.48289	12.8265
(0.1, 0.2)	22	24.6366	0.589777	1.03463	0.693726	1.73401	2.48226	12.7964
(0.1, 0.3)	13	24.6806	0.597795	1.05038	0.723034	1.75365	2.4949	12.8562
(0.1, 0.4)	17	24.3795	0.582715	1.04237	0.695452	1.73793	2.49209	12.8431
(0.1, 0.5)	26	25.0884	0.590032	1.05572	0.682211	1.73307	2.50955	12.8342
(0.1, 0.6)	12	24.9128	0.597823	1.0541	0.683303	1.7278	2.4985	12.8415
(0.1, 0.7)	24	24.8841	0.587284	1.05087	0.700665	1.74037	2.49163	12.8483
(0.1, 0.8)	10	24.9044	0.593647	1.05048	0.705433	1.72113	2.52133	12.8631
(0.1, 0.9)	21	23.8061	0.579902	1.03163	0.697797	1.73499	2.48211	12.8959
(0.2, 0)	6	24.2799	0.553198	0.987834	0.585923	1.65992	2.43313	12.632
(0.2, 0.01)	3	24.283	0.563439	0.998164	0.59434	1.6668	2.42794	12.6714
(0.2, 0.05)	5	24.0034	0.545091	0.988422	0.618189	1.67748	2.43229	12.6434
(0.2, 0.1)	11	23.832	0.540897	0.992221	0.597868	1.66506	2.45102	12.6165
(0.2, 0.2)	3	24.2929	0.555065	1.00783	0.582212	1.64706	2.43987	12.6269
(0.2, 0.3)	7	23.7788	0.542314	0.993429	0.559767	1.64942	2.4294	12.6189
(0.2, 0.4)	3	23.9016	0.53083	0.98022	0.566975	1.63516	2.42961	12.5847
(0.2, 0.8)	6	23.6941	0.505579	0.964395	0.532638	1.60221	2.39104	12.5211

Tabulka C.6: Průměry kritérií, i686, algoritmus Spea

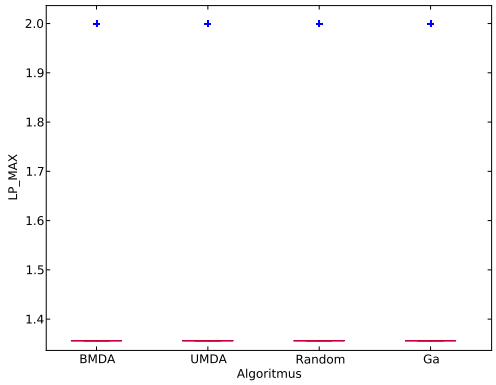
Parametry	Počet nadprů-měrných	Bent skóre	$LP_{\max}$	$DP_{\max}$	SAC	faktor větvení	stupeň po-lynomu	bijektivní skóre
(0.2, 0.9)	2	22.901	0.501721	0.943332	0.497857	1.60441	2.38906	12.5291
(0.3, 0)	2	23.1338	0.488028	0.940361	0.45643	1.57034	2.36104	12.4331
(0.3, 0.01)	1	22.9849	0.472914	0.925429	0.474457	1.58005	2.36685	12.4339
(0.3, 0.05)	1	23.3536	0.482627	0.937762	0.467004	1.57773	2.37146	12.448
(0.3, 0.2)	1	22.9106	0.465335	0.903351	0.458719	1.57652	2.33689	12.4239
(0.3, 0.4)	4	23.0447	0.473823	0.909574	0.450112	1.54643	2.32331	12.3595
(0.3, 0.8)	1	21.8465	0.426339	0.854633	0.400373	1.52133	2.29287	12.3116
(0.4, 0)	1	22.2263	0.424751	0.855939	0.378791	1.50113	2.28781	12.24
(0.4, 0.3)	2	21.1613	0.42104	0.829777	0.375284	1.50031	2.25046	12.2171
(0.4, 0.5)	1	21.3234	0.403927	0.825037	0.366805	1.47344	2.23278	12.169
(0.4, 0.6)	1	21.1718	0.403739	0.820775	0.33995	1.46526	2.22725	12.179
(0.5, 0)	1	21.0312	0.386396	0.794699	0.325821	1.44566	2.22072	12.0761
(0.5, 0.9)	1	18.9958	0.359611	0.742724	0.284351	1.41052	2.13528	11.9901

Tabulka C.7: Průměry při Vega pro reprezentaci CGP

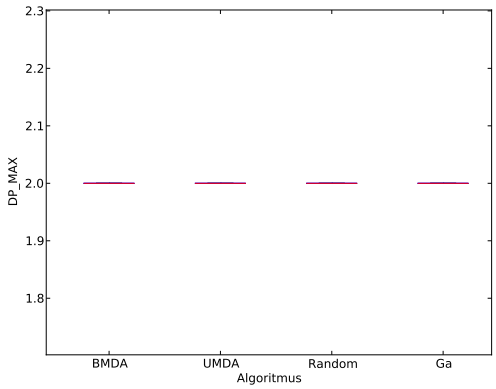
Parametry	Počet nadprů-měrných	Bent skóre	$LP_{\max}$	$DP_{\max}$	SAC	faktor větvení	stupeň po-lynomu	bijektivní skóre
(0, 0)	146	8.15495	0.106639	0.526983	0.0016	1.00335	1.12175	10.3927
(0.01, 0)	166	8.08845	0.107557	0.525988	0.00245	1.00355	1.1267	10.3955
(0.05, 0)	159	8.2103	0.113246	0.527512	0.00145	1.00295	1.1347	10.3882
(0.1, 0)	168	8.15995	0.111118	0.530028	0.0024	1.0038	1.1237	10.3758
(0.2, 0)	146	8.0598	0.11081	0.528936	0.0007	1.003	1.12945	10.399
(0.3, 0)	167	8.2046	0.107698	0.518769	0.0023	1.0036	1.12875	10.4018
(0.4, 0)	180	8.08095	0.114341	0.533752	0.00145	1.0031	1.1328	10.4012
(0.5, 0)	154	8.2248	0.108894	0.519965	0.0021	1.00345	1.12835	10.4013
(0.6, 0)	153	8.10495	0.110147	0.532479	0.00185	1.003	1.13215	10.3891
(0.7, 0)	142	8.1753	0.10966	0.522231	0.0016	1.004	1.1297	10.3837
(0.8, 0)	160	8.14045	0.111748	0.525935	0.0018	1.003	1.1267	10.3777
(0.9, 0)	183	8.0458	0.109198	0.528072	0.002	1.0034	1.12695	10.3822

0 66 46 159 61 240 133 86 188 171 37 123 81 100 151 216 200 193 109 253 45 84 59 10 82 49 220 106 164 57 70 29 245 191 205 91 225 25 125 58 246 195 36 233 247 147 69 217 99 118 114 241 157 96 103 198 212 175 255 202 150 51 80 254 184 28 90 196 237 129 136 213 31 39 72 64 176 215 2 79 33 158 7 148 227 24 172 197 93 22 43 194 169 181 210 199 55 62 160 250 207 105 201 65 214 163 135 74 52 174 124 15 60 8 87 20 238 224 67 166 40 121 223 144 138 116 143 107 48 56 12 115 85 128 27 127 244 242 167 189 16 130 75 236 149 95 32 229 30 102 120 208 47 185 146 211 252 108 203 155 243 78 9 11 89 111 94 141 126 88 134 190 63 50 222 4 206 226 83 180 97 221 153 104 41 18 71 161 228 182 35 248 3 218 231 145 156 183 19 170 187 162 178 1 251 122 117 68 142 186 6 77 5 235 234 119 21 98 139 38 113 132 249 154 165 17 140 177 209 14 173 73 230 219 23 26 76 168 92 131 232 112 44 53 239 101 42 192 152 137 110 13 179 204 54 34

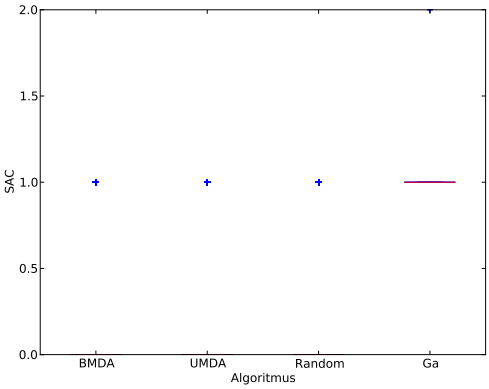
Obrázek C.1: Nejlepší nalezený s-box 8×8



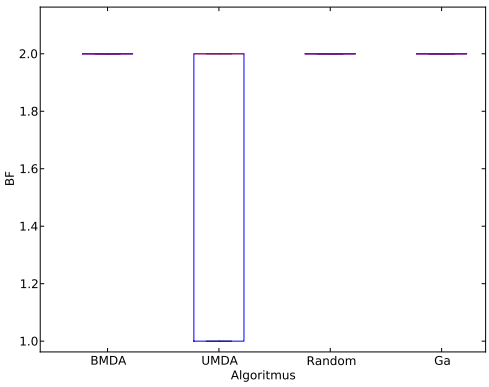
Obrázek C.2: Porovnání EDA algoritmů na binárním reprezentaci s-boxu, kritérium LP_MAX



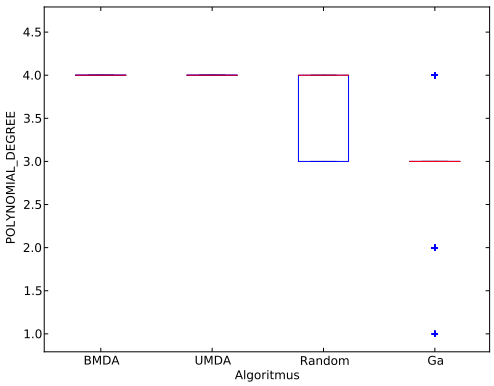
Obrázek C.3: Porovnání EDA algoritmů na binárním reprezentaci s-boxu, kritérium DP_MAX



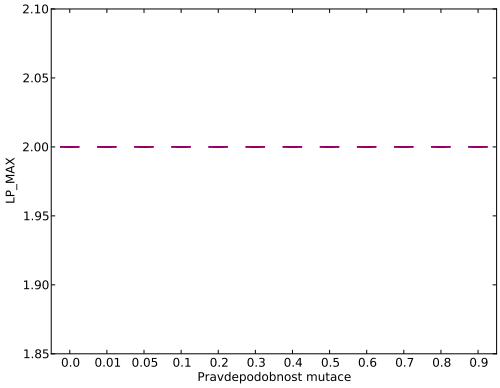
Obrázek C.4: Porovnání EDA algoritmů na binárním reprezentaci s-boxu, kritérium SAC



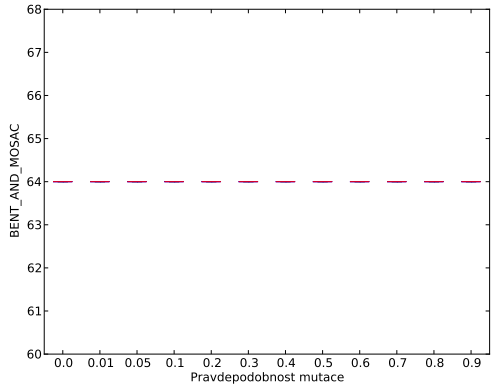
Obrázek C.5: Porovnání EDA algoritmů na binárním reprezentaci s-boxu, branching faktor



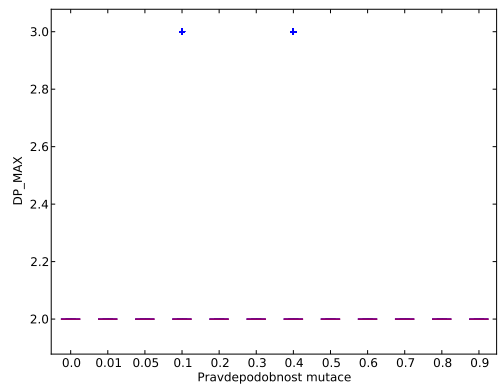
Obrázek C.6: Porovnání EDA algoritmů na binárním reprezentaci s-boxu, stupeň alg. polynomu



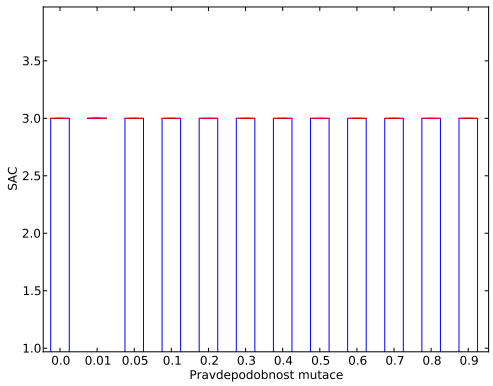
Obrázek C.8: Parametry paralelního náhodného prohledávání, kritérium LP_MAX, reprezentace CGP



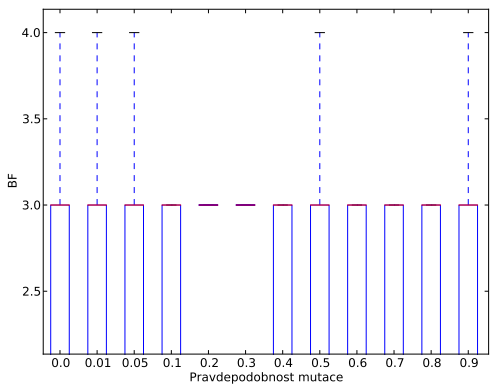
Obrázek C.7: Parametry paralelního náhodného prohledávání,kritérium bent skóre, reprezentace CGP



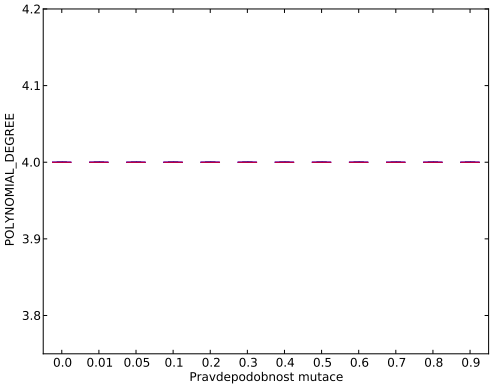
Obrázek C.9: Parametry paralelního náhodného prohledávání, kritérium DP_MAX, reprezentace CGP



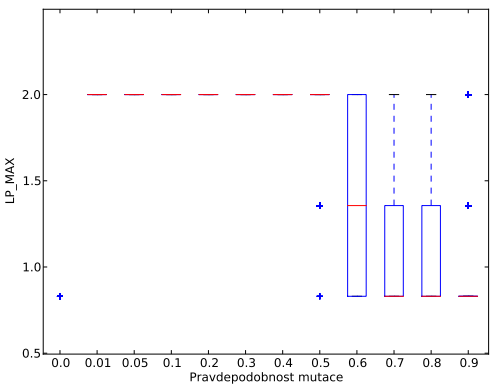
Obrázek C.10: Parametry paralelního náhodného prohledávání, kritérium SAC, reprezentace CGP



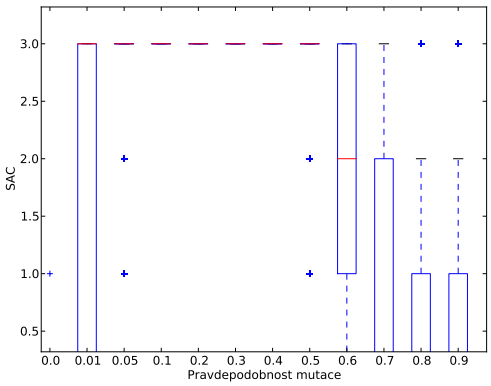
Obrázek C.11: Parametry paralelního náhodného prohledávání, faktor větvení, reprezentace CGP



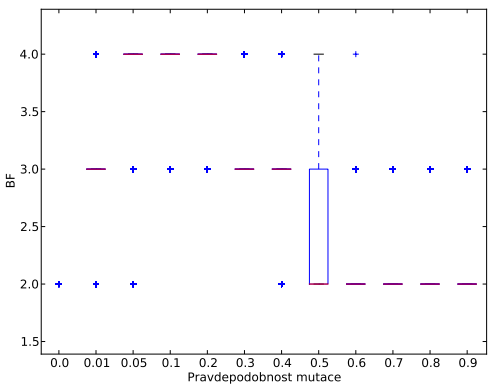
Obrázek C.12: Parametry paralelního náhodného prohledávání, stupeň polynomu, reprezentace CGP



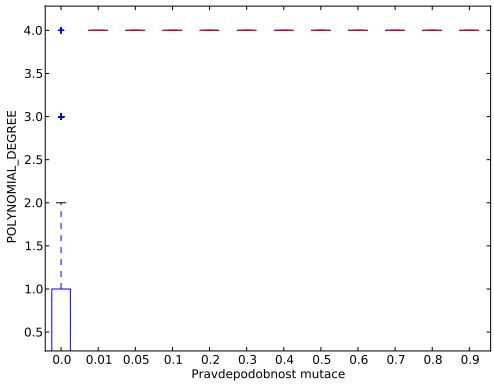
Obrázek C.13: Parametry paralelního náhodného prohledávání,kritérium LP_MAX, reprezentace Luffa



Obrázek C.14: Parametry paralelního náhodného prohledávání,kritérium SAC, reprezentace Luffa



Obrázek C.15: Parametry paralelního náhodného prohledávání, faktor větvení, reprezentace Luffa



Obrázek C.16: Parametry paralelního náhodného prohledávání,stupeň polynomu, reprezentace Luffa