Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Lenka Mišániková

PSO-algoritmy a možnosti jejich využití v kryptoanalýze.

Katedra algebry

Vedoucí diplomové práce: doc. RNDr. Jiří Tůma, DrSc.
Studijní program: Matematika

2011

Ďakujem doc. RNDr. Jiřímu Tůmovi, DrSc. za vedenie tejto diplomovej práce, jeho trpezlivosť a cenné rady. Ďakujem tiež Martinovi Svetlíkovi za pomoc s programovaním.

Prehlasujem, že som svoju diplomovú prácu napísala samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce a jej zverejňovaním.

V Prahe dňa 15.4.2011 Lenka Mišániková

Obsah

Názov práce: PSO-algoritmy a možnosti jejich využití v kryptoanalýze.
Autor: Lenka Mišániková
Katedra (ústav): Katedra algebry
Vedúci diplomovej práce: doc. RNDr. Jiří Tůma, DrSc.
e-mail vedúceho: Jiri.Tuma@mff.cuni.cz

Abstrakt: Cieľom diplomovej práce bolo preskúmať možnosti využitia algoritmu PSO v kryptoanalýze. PSO algoritmus sme aplikovali na riešenie problému jednoduchej zámeny a útoku na šifrový systém DES. Pre jednoduchú zámenu sme pomocou modifikácie diskrétnej verzie PSO dosiahli lepšie alebo porovnateľné výsledky než pomocou iných biologicky motivovaných algoritmov. Navrhli sme spôsob ako využiť PSO na útok na DES a zlomili sme 2-kolový DES pri znalosti len 20 ľubovoľných otvorených a im prislúchajúcich šifrových textov. Analyzovali sme, prečo táto metóda nevedie k úspechu pre viac ako 4-kolový DES. V závere práce sme popísali základné princípy diferenčnej kryptoanalýzy pre DES a navrhli sme špecifickú modifikáciu algoritmu PSO na hľadanie optimálnej diferenčnej charakteristiky pre útok na DES. Pre jednoduché problémy PSO algoritmus funguje veľmi efektívne, pre sofistikované systémy ako DES však bez zabudovania hlbokých znalostí o systéme do algoritmu nie je možné dosiahnuť výraznejšie výsledky.
Kľúčové slová: PSO, DES, jednoduchá zámena, diferenčná charaketeristika

Title: PSO-algorithms and possibilities for their use in cryptanalysis.
Author: Lenka Mišániková
Department: Department of algebra
Supervisor: doc. RNDr. Jiří Tůma, DrSc.
Supervisor's e-mail address: Jiri.Tuma@mff.cuni.cz

Abstract: The aim of the thesis was to investigate the usage of PSO algorithm in the area of cryptanalysis. We applied PSO to the problem of simple substitution and to DES attack. By a modified version of PSO algorithm we achieved better or comparable results as by the usage of other biologically motivated algorithms. We suggested a method how to use PSO to attack DES and we were able to break it with the knowledge of only 20 plain texts and corresponding cipher texts. We have analyzed the reasons of failure to break more than a 4 rounds of DES and provided explanation for it. At the end we described the basic principles of differential cryptanalysis for DES and presented a specific modification of PSO for searching optimal differential characteristics for DES. For simple ciphers, PSO is working efficiently but for sophisticated ciphers like DES, without incorporating deep internal knowledge about the

process into the algorithm, we could not expect significant outcomes.

Keywords: particle swarm optimization, data encryption standard, simple substitution, differential characteristics

Kapitola 1

Úvod

V druhej kapitole je stručné zhrnutie známych poznatkov o algoritme PSO. Sú uvedené motívy, ktoré viedli k jeho vzniku, a je popísaný základný - kanonický algoritmus PSO. Pre jednotlivé aspekty algoritmu ako topológia častíc, hodnoty parametrov, účelová funkcia, sú uvedené niektoré výsledky a poznatky dôležité pre túto prácu. Na záver je uvedená modifikácia pre prípad diskrétnej optimalizácie (priestor riešení je tvorený diskrétnymi bodmi), ktorá sa dá využiť aj v prípade útokov na známe šifrovacie algoritmy, a ktorá je základom ďalších častí práce.

Tretia kapitola je venovaná problému jednoduchej zámeny so zreteľom na využitie PSO pri nájdení riešenia pre problém jednoduchej zámeny. Navrhli sme modifikáciu PSO, ktorá bola motivovaná článkom [10], a ukážeme, že výsledky dosiahnuté modifikovaným algoritmom PSO na riešenie jednoduchej zámeny sú porovnateľné alebe lepšie ako výsledky dosiahnuté inými heuristickými algoritmami. Navrhli a vyskúšali sme 3 ďalie modifikácie, ktoré spočívali v rôznej interpretácii rozdielu $\mathbf{pb_i^k} - \mathbf{x_i^k}$. Výsledky dosiahnuté jednotlivými modifikáciami sú v podstate totožné. Je to dané tým, že najdôležitejším faktorom, ktorý ovplyvňuje algoritmus v prípade jednoduchej zámeny je fakt, či medzi dvoma polohami existuje zmena alebo nie, pričom veľkosť tejto zmeny nie je podstatná. Druhým faktorom je „trivialita" problému jednoduchej zámeny, kde sa rozdiely v jednotlivých prístupoch neprejavia.

Vo štvrtej kapitole sa zaoberáme možnosťou použitia PSO na priamy útok na šifrový systém DES. V prvej časti kapitoly je popísaný algoritmus šifrovania pomocou DES a uvedené niektoré známe útoky na DES. V druhej časti ja navrhnutý prístup, ako s použitím algoritmu PSO nájsť kľúč pre DES. Za týmto účelom sme testovali DES s rôznym počtom kôl. V prípade jedno- a dvojkolovej verzie je možné pomocou PSO nájsť kľúč. Pre 3 kolá existuje šanca nájsť klúč ďalším ladením algoritmu, ale pre vyšší počet kôl daný prístup nevedie k úspechu. V závere sme analyzovali správanie sa použitej účelovej

funkcie a uvádzame dôvody, prečo navrhnutý postup nie je úspešný pre viac ako 3-kolový DES.

Piata kapitola je venovaná prehľadu metódy diferenčnej kryptoanalýzy pre DES. Značenie a postup je podľa práce [27]. V závere kapitoly sú uvedené výsledky známe pre diferenčné charakteristiky pre DES.

V šiestej kapitole je popísané, akým spôsobom sa dá aplikovať PSO na nájdenie diferenčnej charakteristiky pre DES. Táto úloha je značne netriviálna vzhľadom na komplexnosť pojmu diferenčnej charakteristiky, definície priestoru riešení a vyžadovala si zásahy aj do algoritmu PSO, kde pri zmene polohy sme nepoužili výpočet rýchlosti. V literatúre existuje variant PSO bez výpočtu rýchlosti [11], ale náš prístup je značne špecifický a bol motivovaný internými špecifikami šifrovacieho procesu DES.

Kapitola 2

Particle swarm optimization - PSO

V tejto kapitole urobíme stručné zhrnutie poznatkov o algoritme PSO. Uvedieme motívy, ktoré viedli k jeho vzniku, popíšeme samotný algoritmus, jeho fyzikálnu interpretáciu a vplyv jednotlivých faktorov (topológia častíc, hodnoty parametrov, účelová funkcia), ktoré ovplyvňujú algoritmus. Na záver uvedieme modifikáciu pre prípad diskrétnej optimalizácie (priestor riešení je tvorený diskrétnymi bodmi). Táto modifikácia sa dá použiť aj v prípade útokov na známe šifrovacie algoritmy.

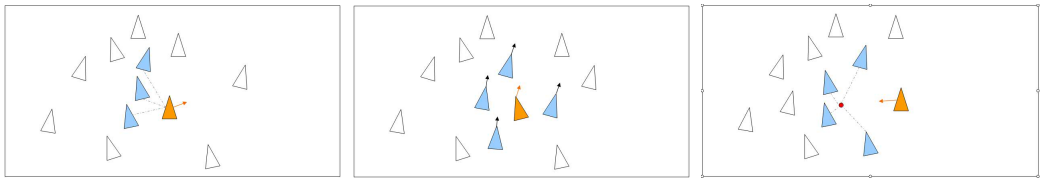
2.1 Charakteristika PSO

Je pozoruhodné, ako kŕdeľ vtákov dokáže synchronizovane letieť v útvare a v momente zmeniť smer, a to bez centrálnej kontroly. Každý vták sa rozhoduje samostatne, a predsa je pohyb kŕdľa dokonale koordinovaný. Tento jav zaujal experta na počíťačovú grafiku Craiga Reynoldsa, ktorý v roku 1986 urobil grafickú simuláciu takéhoto pohybu a nazval ju boids [24]. Každá častica (agent) jeho modelu sa správala podľa troch základných pravidiel.

1. Separácia: každá častica sa snaží vyhnúť sa susedom, ktorí sú príliš blízko.

2. Usmerňovanie: každá častica sa hýbe v priemernom smere pohybu svojich susedov.

3. Kohézia: každá častica sa snaží dostať na priemernú pozíciu svojich susedov.



Obr. 2.1: Separácia,usmerňovanie a kohézia

V roku 1995 Kennedy a Eberhart upravili Reynoldsov model na problém hľadania optima, čím vznikol algoritmus PSO [13].

PSO je výpočtový algoritmus na riešenie optimalizačných problémov. Je to stochastický algoritmus využívajúci heuristické princípy správania sa biologických jedincov zoskupených do celku (napríklad roje, kŕdle, kolónie, ...). V tejto časti popíšeme princíp ako funguje PSO, formálny popis je urobený v časti 2.2

Pre optimalizačné problémy máme danú mmožinu prípustných riešení X a účelovú funkciu $f : X \to R$. Hľadáme také body $x_0 \in X$, pre ktoré účelová funkcia má minimálnu hodnotu. Algoritmus PSO pracuje s množinou častíc - každá častica reprezentuje prípustné riešenie a je charakterizovaná polohou, vektorom rýchlosti, doteraz najlepšou polohou častice a doteraz najlepšou polohou susedných častíc. Pod polohou rozumieme bod v priestore prípustných riešení. Častice sú organizované v určitej topológii, ktorá definuje s ktorými časticami daná častica susedí. Používané topológie a ich vplyv na PSO uvedieme v časti 2.4.

Na začiatku algoritmu si určíme počet častíc a inicializujeme ich. Následne sa vykoná definovaný počet iterácii, pričom v každej iterácii sa všetky častice posunú. Pohyb každej častice je daný vektorom rýchlosti, ktorý je lineárnou kombináciou 3 rôznych vektorov: vektora rýchlosti v predchádzajúcom kroku, vektora smerujúceho z aktuálnej polohy do polohy doterajšej najlepšej pozície a vektora smerujúceho z aktuálnej polohy do najlepšej polohy susedov. Do výpočtu koeficientov lineárnej kombinácie vstupujú pseudonáhodné čísla, ktoré vnášajú stochastickosť do algoritmu. Na základe nového vektora rýchlosti sa aktualizuja poloha jednotlivých častíc. Na záver každej iterácie sa pre všetky častice vypočíta hodnota účelovej funkcie, aktualizuje sa jej dosiaľ najlepšia poloha a najlepšia poloha jej susedov. Celý proces sa končí po vykonaní určitého počtu iterácií alebo po dosiahnutí žiadanej hodnoty účelovej funkcie.

Fascinujúce na PSO a iných biologicky motivovaných algoritmoch je schopnosť celku riešiť zložité úlohy, pričom jednotlivé častice celku majú veľmi limitované schopnosti. Ako neskôr uvidíme, je tu istá podobnosť s DES a inými modernými šiframi, ktoré viacnásobným opakovaním relatívne jednoduchej šifrovacej schémy sú schopné dosiahnuť požadovanú mieru bezpečnosti.

Výhodou a silou PSO je, že nevyužíva gradient pre určenie novej polohy a dokáže tak riešiť úlohy, kde gradient je ťažko vypočítateľný alebo dokonca neexistuje. Cena za túto flexibilitu je v ladení algoritmu, hlavne v nájdení správnych parametrov (počet častíc, hodnoty jednotlivých koeficientov) a určení vhodnej účelovej funkcie pre daný problém, ktoré zabezpečia rýchlu konvergenciu a efektívnosť algoritmu. PSO je vhodné aj na paralelné spracovanie - spojením viacerých výpočtových kapacít je možné dosiahnuť výsledky v kratšom čase [16]. Typicky sa paralelizuje vyhodnotenie účelovej funkcie pre každú časticu.

2.2 Popis základného algoritmu PSO

Algoritmus PSO bol pôvodne navrhnutý na optimalizačné úlohy v reálnom(spojitom) priestore. Definujme problém nasledovne:
Hľadáme množinu $X^* \subseteq X \subseteq \Re^n$ tak, že platí

$$X^* = \arg\min_{\mathbf{x}\in\mathbf{X}} f(x) = \{x^* \in X : f(x^*) \leq f(x) : \forall x \in X\}\,.$$

Poznámka: PSO nie je určené na nájdenie všetkých optimálnych riešení. Pomocou PSO je možné nájsť dobré riešenie v zmysle, že hodnota účelovej funkcie pre riešenie nájdené PSO bude blízka optimálnej hodnote. V prípade viacnásobného spustenia algoritmu je možné, že výsledkom každého behu bude iné riešenie.

Zvoľme si počet častíc, ktoré náhodne a rovnomerne rozmiestnime po celom priestore možných riešení (dolný index i je index častice). Každej častici v každom kroku algoritmu prislúchajú 4 vektory:

1. Jej súčasná poloha $\mathbf{x_i} \in \Re^n$.
2. Vektor rýchlosti $\mathbf{v_i} \in \Re^n$. Obyčajne zložky rýchlosti sa v implementáciách PSO algoritmu limitujú na vopred zvolený interval $[-v_{max}, v_{max}]$.
3. Poloha $\mathbf{pb_i}$, v ktorej mala doteraz najmenšiu hodnotu účelová funkcia pre časticu $\mathbf{x_i}$.
4. Najlepšia poloha $\mathbf{gb_i}$, ktorú nadobudla niektorá častica z definovaného okolia $\mathbf{x_i}$. Okolie je určené topológiou a podrobnejšie ho popíšeme v časti 2.4.

Samotný algoritmus PSO pozostáva z nasledujúcich krokov:

Inicializácia

1. Inicializuj parametre algoritmu (konštanty w, c_1, c_2, m)
2. Pre všetky častice $i = 1, \ldots, m$
   - Náhodne vygeneruj počiatočné hodnoty polohy častíc $\mathbf{x_i^0}$.
   - Inicializuj vektor rýchlosti pre každú časticu $\mathbf{v_i^0}$.
   - Nastav polohy $\mathbf{pb_i^0} = \mathbf{x_i^0}$.
   - Inicializuj $\mathbf{gb_i^0} = \min\{f(\mathbf{x_1^0}), f(\mathbf{x_2^0}), \ldots, f(\mathbf{x_m^0})\}$.

Cyklus. Iteruj dovtedy, kým nie je splnená podmienka ukončenia:

1. Pre každú časticu vypočítaj hodnotu účelovej funkcie $f(\mathbf{x_i})$ a porovnaj s $f(\mathbf{pb_i})$ pre danú časticu. V prípade, $f(\mathbf{x_i}) < f(\mathbf{pb_i})$ aktualizuj $\mathbf{pb_i}$ novou hodnotou. Obdobne pre $\mathbf{gb_i}$.
2. Zmeň rýchlosť a polohu častice podľa nasledujúcich vzťahov

$$\mathbf{v_i^{k+1}} = w\mathbf{v_i^k} + c_1 r_p(\mathbf{pb_i^k} - \mathbf{x_i^k}) + c_2 r_g(\mathbf{gb_i^k} - \mathbf{x_i^k}), \tag{2.1}$$

$$\mathbf{x_i^{k+1}} = \mathbf{x_i^k} + \mathbf{v_i^{k+1}}, \tag{2.2}$$

   kde r_p a r_g sú dve pseudonáhodné čísla, w, c_1 a c_2 sú dopredu určené konštanty a k je index iterácie.

Koniec cyklu.

2.3 Fyzikálna interpretácia PSO a význam jednotlivých parametrov

Rovnice (2.1) a (2.2) môžeme porovnať s fyzikálnou rovnicou pre výpočet posunutia v prípade rovnomerne zrýchleného pohybu

$$\mathbf{x} = \mathbf{x_0} + \mathbf{v_0}t + \frac{1}{2}\mathbf{a}t^2, \tag{2.3}$$

kde $\mathbf{x}$ označuje novú polohu, $\mathbf{x_0}$ východziu polohu, $\mathbf{v_0}$ rýchlosť, t čas a $\mathbf{a}$ označuje zrýchlenie. Pre účely implementácie potrebujeme pracovať s časom ako s diskrétnou veličinou, preto si čas rozdelíme na jednotkové kroky. Dostaneme tak iteratívnu verziu predchádzajúceho vzťahu:

$$\mathbf{x^{k+1}} = \mathbf{x^k} + \mathbf{v^k} + \frac{1}{2}\mathbf{a^k}. \tag{2.4}$$

V PSO sa akcelerácia mení dynamicky v každom kroku a má dve základné zložky:

1. Kognitívna akcelerácia, ktorá je priamo úmerná vzdialenosti častice od osobného optima (tzv. personal best) $(\mathbf{pb} - \mathbf{x})$ s koeficientom kognitívneho zrýchlenia c_1.

2. Sociálna akcelerácia, ktorá je priamo úmerná vzdialenosti častice od globálneho optima (tzv. global best, ktorý reprezentuje najlepšiu hodnotu dosiahnutú ktoroukoľvek časticou z okolia danej častice) $(\mathbf{gb} - \mathbf{x})$ s koeficientom sociálneho zrýchlenia c_2.

Celková akcelerácia je daná súčtom kognitívnej a sociálnej akcelerácie. Zrýchlenie môžeme teda napísať ako

$$\mathbf{a^k} = c_1(\mathbf{pb^k} - \mathbf{x^k}) + c_2(\mathbf{gb^k} - \mathbf{x^k}).$$

Po dosadení do vzťahu (2.4) dostávame

$$\mathbf{x^{k+1}} = \mathbf{x^k} + \mathbf{v^k} + \frac{1}{2}c_1(\mathbf{pb^k} - \mathbf{x^k}) + \frac{1}{2}c_2(\mathbf{gb^k} - \mathbf{x^k}).$$

Namiesto konštanty $\frac{1}{2}$ volíme pseudonáhodné čísla r_p, r_g, ktorých priemerná hodnota je $\frac{1}{2}$, teda sú z intervalu $[0, 1]$, aby sme do algoritmu vniesli stochastickosť. Máme

$$\mathbf{x^{k+1}} = \mathbf{x^k} + \mathbf{v^k} + r_p c_1(\mathbf{pb^k} - \mathbf{x^k}) + r_g c_2(\mathbf{gb^k} - \mathbf{x^k}).$$

Aby sme zabránili nekontrolovateľnému nárastu rýchlosti, bol zavedený koeficient trenia w.

$$\mathbf{x^{k+1}} = \mathbf{x^k} + w\mathbf{v^k} + r_p c_1(\mathbf{pb^k} - \mathbf{x^k}) + r_g c_2(\mathbf{gb^k} - \mathbf{x^k}).$$

Rovnicu sme rozdelili na dve časti:

$$\mathbf{v^{k+1}} = w\mathbf{v^k} + c_1 r_p(\mathbf{pb^k} - \mathbf{x^k}) + c_2 r_g(\mathbf{gb^k} - \mathbf{x^k}), \tag{2.5}$$

$$\mathbf{x^{k+1}} = \mathbf{x^k} + \mathbf{v^{k+1}}, \tag{2.6}$$

Vidíme, že skutočne ide o vzťahy (2.1), (2.2).

Význam jednotlivých parametrov sa dá interpretovať nasledovne.

- Parameter w: tento parameter zaviedli v roku 1998 Shi a Eberhart [26] a dá sa interpretovať ako koeficient zotrvačnosti a $(1 - w)$ ako koeficient trenia. Pri hodnotách parametra w blízkych k 1 prevláda tendencia prehľadať čo najväčšiu časť priestoru, kým pri menších hodnotách algoritmus spôsobí, že častice „sa sústredia" na prehľadanie okolia momentálne najlepšej polohy **pb** resp. **gb**. V niektorých modifikáciách PSO sa hodnota parametru w nastaví na vyššiu hodnotu (okolo 1) a postupne sa znižuje.

- Parametre r_p, r_g sú pseudonáhodné čísla z intervalu $[0, 1]$ a vnášajú do procesu stochastickosť. Generujú sa nanovo v každom kroku a pre každú časticu. V pôvodnom algoritme sa generovalo iné pseudonáhodné číslo pre každú súradnicu.

- Parametre c_1, c_2: sú koeficienty kognitívnej akcelerácie a sociálnej akcelerácie. Udávajú, či častica má tendenciu ísť k svojmu doterajšiemu najlepšiemu výsledku $\mathbf{pb_i}$, alebo k doteraz najlepšiemu výsledku niektorej častice zo svojho okolia $\mathbf{gb_i}$.

V prvej verzii algoritmu bolo nutné obmedziť dynamiku častíc limitovaním rýchlosti, inak rýchlosť neobmedzene narastala. Preto sa hľadal vzťah, ktorý by zabezpečil konvergenciu aj bez umelého obmedzenia rýchlosti. V práci [5] sa namiesto vzťahu (2.1) používa algebraicky ekvivalentný vzťah

$$\mathbf{v_i^{k+1}} = \chi(\mathbf{v_i^k} + c_1 r_p(\mathbf{pb_i^k} - \mathbf{x_i^k}) + c_2 r_g(\mathbf{gb_i^k} - \mathbf{x_i^k})), \text{ kde} \tag{2.7}$$

$$\chi = \frac{2}{|\phi - 2 + \sqrt{\phi^2 - 4\phi}|}, \tag{2.8}$$

$$\phi = c_1 + c_2 > 4. \tag{2.9}$$

Obyčajne sa volia nasledovné hodnoty parametrov $\phi = 4.1$, $c_1 = c_2$, $\chi = 0.7298$ a algoritmus konverguje aj bez obmedzenia rýchlosti parametrom v_{max}. Analýzy, ktoré viedli k týmto voľbám boli urobené zjednodušeniami v PSO (sú to hlavne deterministické modely bez stochastickosti). Správanie PSO závisí od parametra ϕ. Mohan a Ozcan urobili analýzu zjednodušeného PSO, ktorý naviac redukovali na jednu časticu. Vyšlo im, že v prípade $\phi > 4$ oscilácia častice narastá. Pre $\phi < 4$ častica vykonáva periodický pohyb [21]. V praxi sa spolu so vzťahom (2.7) naďalej používa obmedzenie na interval [-v_{max}, v_{max}], lebo táto kombinácia má stále zmysel a vykazuje lepšie výsledky než metóda bez v_{max}.

V [22] sa použili optimalizačné metódy na nájdenie hodnôt pre parametre w, c_1, c_2 a určenie počtu častíc m. Uvedené parametre sa menia v závislosti od dimenzie priestoru riešenia a povoleného počtu vyhodnotení účelovej funkcie. Publikovaná tabuľka 2.1 uvádza odporúčané hodnoty jednotlivých parametrov pre danú dimenziu a počet vyhodnotení funkcie. Pre praktické použitie je to dobrý východiskový bod, ako nastaviť jednotlivé parametre.

Napriek mnohým článkom a rozsiahlemu výskumu, neexistuje konsolidovaný pohľad na voľbu parametrov w, c_1, c_2, ktorý by bol platný pre širokú

Dimenzia problému	Počet vyhodnotení účelovej funkcie	PSO parametre			
		m	w	c_1	c_2
2	400	25	0.3925	2.5586	1.3358
		29	-0.4349	-0.6504	2.2073
2	4,000	156	0.4091	2.1304	1.0575
		237	-0.2887	0.4862	2.5067
5	1,000	63	-0.3593	-0.7238	2.0289
		47	-0.1832	0.5287	3.1913
5	10,000	223	-0.3699	-0.1207	3.3657
		203	0.5069	2.5524	1.0056
10	2,000	63	0.6571	1.6319	0.6239
		204	-0.2134	-0.3344	2.3259
10	20,000	53	-0.3488	-0.2746	4.8976
20	40,000	69	-0.4438	-0.2699	3.3950
20	400,000	149	-0.3236	-0.1136	3.9789
		60	-0.4736	-0.9700	3.7904
		256	-0.3499	-0.0513	4.9087
30	600,000	95	-0.6031	-0.6485	2.6475
50	100,000	106	-0.2256	-0.1564	3.8876
100	200,000	161	-0.2089	-0.0787	3.7637

Tabuľka 2.1: Rôzne nastavenia parametrov PSO ako ich uvádza Pedersen[22]. Parametre volíme podľa dimenzie problému a povoleného počtu vyhodnotení účelovej funkcie. V niektorých prípadoch je uvedených viac možných nastavení, keďže majú skoro totožnú úspešnosť. Parameter m predstavuje odporúčaný počet častíc použitých v algoritme PSO.

triedu problémov a zabezpečoval efektivitu PSO. Čiastkové výsledky sú známe pre konkrétne typy úloh, existujú matematické modely zjednodušenej PSO a odporúčané hodnoty pre rôzne dimenzie problému. Sú to však skôr návody a odporúčania vychádzajúce zo skúsenosti než z presných matematických dôkazov.

2.4 Vplyv topológie častíc - spôsoby určenia susedov

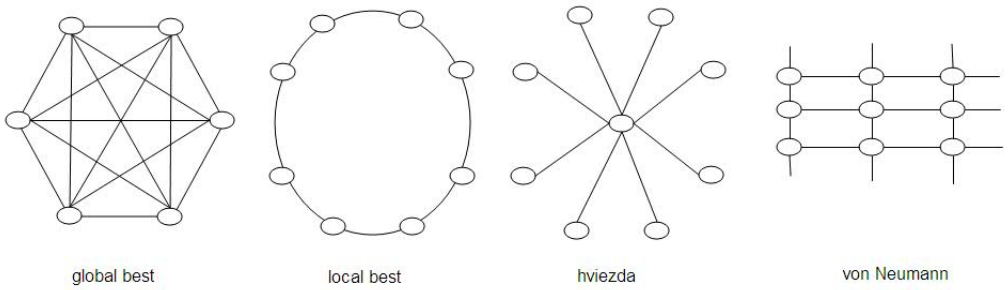
Pri algoritme PSO sa jednotlivé častice pohybujú jednak smerom k svojmu osobnému optimu a jednak smerom ku globálnemu optimu, ktoré si častice

medzi sebou zdieľajú. Zaujímavou otázkou je, čo sa stane, ak nebudú všetky častice komunikovať so všetkými, ale len so svojimi najbližšími susedmi. Každá častica bude mať svojich susedov a len týmto bude posielať svoju najlepšiu pozíciu. Susedstvo je vždy symetrické. Existujú rôzne metódy ako definovať topológiu častíc, ktoré sa nazývajú populačné topológie. Najčastejšie sa rozlišuje medzi

1. statickými topológiami, ktoré sa nemenia počas behu algoritmu a
2. dynamickými topológiami, ktoré sa menia počas jednotlivých iterácií.

Populačnú (statickú)topológiu môžeme znázorňovať pomocou grafu, kde častice sú vrcholy a výmena informácii je znázornená obojstrannou hranou.

V počiatkoch PSO susedili častice podľa ich euklidovskej vzdialeností v priestore riešení. Táto štruktúra sa však neosvedčila nielen preto, že bola výpočtovo náročná, ale hlavne pre svoje slabé konvergenčné vlastnosti [23]. Typickými statickými topológiami sú nasledujúce topológie:



Obr. 2.2: Rôzne typy populačných topológií.

Global best, kde každá častica má informáciu o doteraz najlepšej dosiahnutej hodnote účelovej funkcie, spolu s polohou, v ktorej bola dosiahnutá. Inými slovami, vymieňa sa informácia o najlepšiej hodnote medzi všetkými časticami navzájom.

Local best, kde každá častica susedí s práve dvoma ďalšími časticmi (posledná častica je prepojená s predposlednou a prvou) a častice sú pospájané do kruhu.

Von Neumannova topológia, ak si predstavíme rovinu, každá častica má 4 susedov, jedného na každú svetovú stranu - jedná sa o ekvivalent pravidelnej štvorcovej mriežky. Existujú varianty aj v dimenziách vyšších než 2.

iné topológie ako pyramída, hviezda, ...

V prípade global best topológie sa všetky častice dozvedia o aktuálnom minime a sú k tejto polohe „priťahované". V local best sa môžu vytvoriť lokálne klastre častíc v okolí viacerých miním s podobnou hodnotou účelovej funkcie a preskúmať tieto okolia podrobnejšie. Z tohto dôvodu global best konverguje rýchlejšie, ale je väčšia šanca zablúdenia do lokálneho minima [18].

Dynamické topológie zvyčajne v prvej fáze využívajú topológiu, kde každá častica má malý počet susedov (napríklad local best) a počet susedov sa postupne zvyšuje. Existuje veľké množstvo topológií a v závislosti od jej zvolenia sa signifikantne mení výkon PSO algoritmu. Voľba vhodnej topológie závisí od účelovej funkcie, ktorú testujeme, avšak dodnes nie je známe, že by nejaká topólogia bola všeobecne lepšia od ostatných [14].

2.5 Určenie účelovej funkcie

Keď chceme vyriešiť nejaký problém pomocou algoritmu PSO, potrebujeme ho previesť na optimalizačnú úlohu. Uveďme konkrétny príklad, na ktorom chceme demonštrovať dôležitosť a netriviálnosť určenia vhodnej účelovej funkcie:

Zadanie úlohy:
Majme trojrozmerné teleso umiestnené v jednotkovej kocke $Q = [0,1] \times [0,1] \times [0,1]$. Úlohou je nájsť súradnice ťažiska telesa.

Prevod úlohy na optimalizačný problém pre PSO
Priestor prípustných riešení bude jednotková kocka. Pre každý bod jednotkovej kocky je potrebné definovať účelovú funkciu, ktorá by v určitom zmysle merala blízkosť bodu k ťažisku. Súradnice ťažiska nepoznáme, takže musíme nájsť funkciu, ktorá nevyužíva túto informáciu, ale minimum nadobudne v ťažisku.

Vhodná účelová funkcia by mala spĺňať nasledovné podmienky (nájsť účelovú funkciu, ktorá tieto podmienky spĺňa môže byť v niektorých prípadoch značne netriviálna úloha):

- musí mať silnú koreláciu s pôvodným problémom v tom zmysle, že ak ρ je metrika nad priestorom riešení X a x_1, x_2 sú 2 prípustné riešenia z definovaného okolia optimálneho riešenia x_0 , také že $\rho(x_1, x_0) < \rho(x_2, x_0)$, tak $f(x_1) < f(x_2)$. Čiže účelová funkcia musí pre „lepšie" riešenie priradiť nižšiu hodnotu než pre „horšie" riešenie.

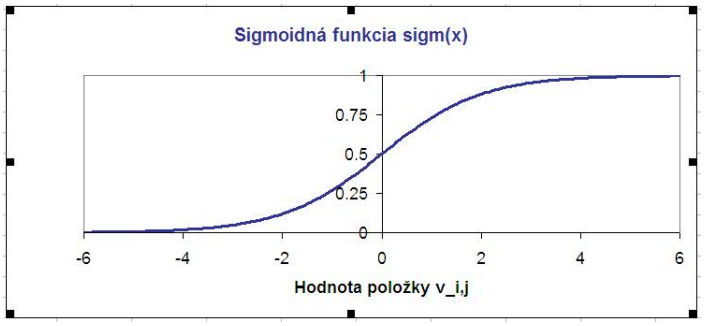
- výpočet účelovej funkcie musí byť rýchly a efektívny, nakoľko PSO algoritmus musí vyhodnotiť účelovú funkciu v každej iterácii pre všetky častice.

2.6 Diskrétna verzia PSO

Mnohé optimalizačné úlohy sú definované v diskrétnom priestore - napríklad $(Z_2)^n$, kde jednotlivé súradnice nadobúdajú len určité vopred stanovené hodnoty. Je to príklad kombinatorických úloh alebo úloh v kryptografii.

V klasickej PSO sa v každom kroku aktualizuje rýchlosť a poloha častíc. V diskrétnom priestore môžu jednotlivé súradnice stavového vektora nadobúdať len konečný počet hodnôt a vzťah (2.2) sa nedá priamočiaro aplikovať. Riešením je interpretovať súradnice vektora rýchlosti ako pravdepodobnosti, že príslušná súradnica vektora polohy nadobudne hodnotu 0 alebo 1 [12]. Inými slovami d-tá súradnica pre časticu i vo vektore rýchlosti $v_{i,d}$ určuje pravdepodobnosť, s akou bude mať častica i na d-tej súradnici svojej polohy $x_{i,d}$ hodnotu 0 alebo 1. Keďže zložky vektora $\mathbf{v_i^k}$ udávajú pravdepodobnosti, musíme ich obmedziť na interval $[0, 1]$. Použijeme funkciu

$$s(v_{i,d}^k) = v_{i,d}^{*k} = \frac{1}{1 + \exp(-v_{i,d}^k)}. \tag{2.10}$$



Obr. 2.3: Priebeh funkcie s, ktorá zobrazuje R na inetrval $[0, 1]$

Častica i zmení hodnotu bitu d podľa nasledujúceho vzťahu, kde pod $rnd()$ chápeme pseudonáhodné číslo zvolené z rovnomerného rozdelenia intervalu $[0, 1]$.

$$x_{i,d}^{k+1} = \begin{cases} 1 & \text{if } rnd() \leq s(v_{i,d}^{k+1}), \\ 0 & \text{inak.} \end{cases}$$

Uveďme príklad. Nech $v_{i,5}^{*(k+1)} = 0.8$. To znamená, že s pravdepodobnosťou 80% piata súradnica vektora x_i^{k+1} bude mať hodnotu 1 a s pravdepodobnosťou 20% hodnotu 0.

Pre diskrétnu verziu sa teda vzťahy [2.1] a [2.2] zmenia nasledovne

$$\mathbf{v_i^{k+1}} = w\mathbf{v_i^k} + c_1 r_p(\mathbf{pb_i^k} - \mathbf{x_i^k}) + c_2 r_g(\mathbf{gb_i^k} - \mathbf{x_i^k}) \tag{2.11}$$

$$x_{i,d}^{k+1} = \begin{cases} 1 & \text{if } rnd() \leq s(v_{i,d}^{k+1}), \\ 0 & \text{inak.} \end{cases} \tag{2.12}$$

Hlavný rozdiel diskrétnej verzie PSO voči pôvodnej spojitej verzii je v interpretácii vektora rýchlosti. V diskrétnej verzii sa vektor rýchlosti transformuje na stochastický vektor, ktorý určuje pravdepodobnosť s akou sa častica nachádza v nejakom stave. Všimnime si, že zo vzťahu (2.12) vyplýva, že k zmene stavu môže dôjsť aj v prípade, keď sa vektor rýchlosti nezmení medzi iteráciami.

Kapitola 3

Aplikácia PSO na jednoduchú zámenu

V tejto kapitole popíšeme ako sa diskrétna verzia PSO dá modifikovať na riešenie jednoduchej zámeny. Ukážeme, že výsledky dosiahnuté s algoritmom PSO na riešenie jednoduchej zámeny sú porovnateľné alebe lepšie ako výsledky dosiahnuté inými heuristickými algoritmami. Navrhli sme a vyskúšali 3 variácie algoritmu PSO, ktoré spočívali v rôznej interpretácii rozdielu $\mathbf{pb_i^k} - \mathbf{x_i^k}$. Dosiahli sme veľmi podobné výsledky. Je to dané tým, že jediným faktorom, ktorý ovplyvňuje algoritmus, je skutočnosť, či medzi dvoma polohami existuje zmena alebo nie, pričom veľkosť tejto zmeny nie je podstatná. Druhým faktorom je „trivialita" problému jednoduchej zámeny aplikovanej na úrovni znakov, kde sa rozdiely v jednotlivých prístupoch neprejavia.

3.1 Jednoduchá zámena

Jednoduchá zámena patrí medzi klasické šifry. Je známa uz niekoľko storočí a teší sa veľkej obľube aj medzi amatérskymi kryptografmi, či lúštiteľmi hádaniek. Častý je aj jej výskyt v detektívkach, napríklad „Tancujúce figúrky" od Arthura Conana Doyla. V tomto príbehu vyrieši Sherlock Holmes vraždu po tom, čo odhalí, že reťazce čudných postavičiek sú v skutočnosti správy šifrované (jednoduchou) substitučnou šifrou.

Obr. 3.1: šifrový text z knihy Artura Conana Doyla: Tancujúce figúrky

Ako vyplýva už z názvu, jednoduchá zámena je metóda, kde každý znak

(písmeno) otvoreného textu je zamenené iným znakom (písmenom) v šifrovom texte. Všeobecne môže mať otvorený a šifrový text každý svoju(inú) abecedu, tak ako je to v prípade Sherlocka Holmesa, kde sa v otvorenom texte používa anglická abeceda a v šifrovom texte ide o 26 znakov - postavičiek. Častý je však prípad, keď sú tieto abecedy zhodné.

Hoci je táto šifra už dávno prelomená, je stále dôležitá pre kryptografiu. Substitúcia je komponentom mnohých moderných šifier, aj keď dnes sa už používajú blokové šifry (blokové šifry nepracujú nad znakmi, ale nad blokmi textu). Hoci je DES bloková šifra, v podstate je to stále substitúcia, aj keď pravidlá substitúcie sú značne komplikované a navrhnuté tak, aby odolali kryptoanalytickým útokom [20]. Naďalej sa však mnoho nových útokov testuje na jednoduchej zámene alebo iných šifrách klasickej kryptografie, keďže sú komponentami sofistikovaných a moderných šifier.

Jednoduchá zámena sa môže javiť na prvý pohľad ako ťažký problém, opak je však pravdou. Hlavným nedostatkom tejto šifry je skutočnosť, že každý znak sa vždy zobrazí na ten istý znak. Z toho vyplýva, že jednoduchá zámena zachováva nielen rozdelenie frekvencií jednotlivých znakov, ale aj ich rozmiestnenie. Frekvencie jednotlivých hlások v jazyku sú výrazne odlišné, napríklad v angličtine je najfrekventovanejšia hláska „e" s výskytom vyše 9%. To isté platí aj pre dvojice(bigramy), trojice(trigramy), či n-tice hlások. Práve na frekvenčnej analýze je založená metóda, ktorá úspešne rieši jednoduchú zámenu.

3.2 Ako aplikovať diskrétnu PSO na jednoduchú zámenu

Definícia 1. Nech M je množina {1,2,...,n}. Permutácia nad M je každé bijektívne zobrazenie $p : M \to M$. Množinu všetkých permutácií nad M budeme označovať S_n.

Očíslujme písmená anglickej abecedy (plus medzera) $1, .., 27$ ('a' bude 1, 'b' bude 2, až ␣bude 27). Potom kľúč pre jednoduchú zámenu v prípade, že vstupná aj výstupná abeceda je abeceda anglického jazyka, je ľubovoľná permutácia $k \in S_{27}$.

Značenie. *Majme permutáciu $k \in S_n$, definovanú nasledovne: $k(1) = s_1, k(2) = s_2, \ldots, k(n) = s_n$. Permutáciu k zapíšeme nasledovne*

$$k = \begin{pmatrix} 1 & 2 & \ldots & n \\ s_1 & s_2 & \ldots & s_n \end{pmatrix} \tag{3.1}$$

alebo v zjednodušenom jednoriadkovom zápise ako $(s_1, s_2, \ldots, s_n)$*, kde uvedieme len druhý riadok vzťahu (3.1)*

Definícia 2. Nech M je množina {1,2,...,n} a pre $k \geq 2$ $\{r_1, r_2, \ldots, r_k\} \subset M$. Permutáciu $c \in S_n$ takú, že

$$c(r_1) = r_2,\ c(r_2) = r_3,\ \ldots, c(r_{k-1}) = r_k,\ c(r_k) = r_1$$
$$c(m) = m\ \forall m \in M - \{r_1, r_2, \ldots, r_k\}$$

budeme nazývať cyklom dĺžky k a označovať $(r_1, r_2, \ldots, r_k)$. Cyklus dĺžky 2 nazývame transpozícia.

Priestor kľúčov pre jednoduchú zámenu pre anglickú abecedu je veľkosti $27! \approx 10^{28} \approx 2^{93}$. Podotknime, že na dešifrovanie správy sa použije inverzná permutácia k^{-1}.

Pomocou PSO hľadáme kľúč - permutáciu, podľa ktorého bola urobená jednoduchá zámena. Priestor riešení je teda S_{27} a poloha častice je určená niektorou permutáciou. Keď meníme polohu častice podľa vzťahu (2.12), nemôžeme túto zmenu urobiť len mechanicky, lebo výsledok zmeny musí byť permutácia. Akonáhle zmeníme hodnotu jednej súradnice vo vektori $\mathbf{x_i^{k+1}}$, musíme zmeniť aspoň jednu ďalšiu, aby sme zachovali definíciu permutácie. Priestor riešení pri jednoduchej zámene je tvorený inými objektami - permutáciami, a musíme modifikovať vzťahy (2.11) a (2.12), a predefinovať zmenu polohy častice a definovať operáciu „odčítania" vo vzťahu (2.11).

Pri návrhu modifikovaného algoritmu sme vychádzali z článku [10], v ktorom sa aplikoval algoritmus PSO na problém rozmiestnenia kráľovien na šachovnici, ktorého riešením je tiež permutácia.

Počas inicializácie je potrebné nastavit počiatočné polohy častíc, ktoré podľa základného algoritmu sa rozmietnia náhodne v priestore. V našom prípade je potrebné vygenorovať náhodné permutácie. Tieto sme generovali podľa Durstenfeldovho algoritmu. Spôsob, akým sa mení rýchlosť a poloha častice i je nasledovný.

- Pri výpočte vektoru rýchlosti $\mathbf{v_i^{k+1}}$ budeme s vektorm $\mathbf{x_i^k}, \mathbf{pb_i^k}, \mathbf{lb_i^k}$ pracovať ako s vektormi v R^n a nie ako s permutáciami. Pri výpočte rozdielov $\mathbf{gb_i^k} - \mathbf{x_i^k}$ a $\mathbf{lb_i^k} - \mathbf{x_i^k}$ sme pre každú súradnicu priradili absolútnu hodnotu rozdielu. Následne nový vektor rýchlosti normalizujeme do rozpätia 0 až 1 vydelením najvyššou hodnotou zo súradnic rýchlosti (maximovou normou) a označíme $\mathbf{u_i^{k+1}}$. Každá súradncia vektora $\mathbf{u_i^{k+1}}$ udáva tú pravdepodobnosť, s ktorou v prislúchajúcej súradnici polohy častice $\mathbf{x_i^{k+1}}$ dôjde k zmene.

- V prípade, že častica sa už nachádza v pozícii local best, urobíme jednu náhodnú transpozíciu. Inak postupujeme nasledovne. Na začiatku položíme $\mathbf{x_i^{k+1}} = \mathbf{x_i^k}$. Postupne prechádzame všetky súradnice $u_{i,d}^{k+1}$ normalizovaného vektora rýchlosti $\mathbf{u_i^{k+1}}$. Pre každú súradnicu d vygenerujeme pseudonáhodné číslo $rnd()$ z intervalu $[0, 1]$. Ak $rnd() < u_{i,d}^{k+1}$, potom v súradnici $x_{i,d}^{k+1}$ dôjde k zmene. Hodnota $x_{i,d}^{k+1}$ sa vymení za hodnotu $lb_{i,d}^k$. Ak si uvedomíme, že každá poloha predstavuje permutáciu z S_{27} a výmenu d-tej súradnice vo vektori $\mathbf{x_i^{k+1}}$ môžeme formálne vyjadriť ako transpozíciu $t_{i,d}^k = (x_{i,d}^k, lb_{i,d}^k)$, tak nová permutácia $\mathbf{x_i^{k+1}}$ sa dá vyjadriť ako zloženie série transpozícií s pôvodnou permutáciou $\mathbf{x_i^k}$.

Teda vzťah (2.11)sa zmení nasledovne

$$\begin{aligned} v_{i,d}^{k+1} &= wv_{i,d}^k + c_1 r_p(\left|pb_{i,d}^k - x_{i,d}^k\right|) + c_2 r_g(\left|lb_{i,d}^k - x_{i,d}^k\right|) \\ \mathbf{u_i^{k+1}} &= \mathbf{v_i^{k+1}} / \|\mathbf{v_i^{k+1}}\|_{max} \end{aligned} \tag{3.2}$$

a pre zmenu polohy (2.12) budeme používať nasledovný vzťah

$$\begin{aligned} &if\ (\mathbf{x^k} \neq \mathbf{lb^k}\ ) \qquad then \\ &\qquad \mathbf{x_i^{k+1}} = t_{i,27}^k \circ t_{i,26}^k \circ \ldots \circ t_{i,1}^k \circ \mathbf{x_i^k} \\ &else \\ &\qquad \mathbf{x_i^{k+1}} = t^{rnd} \circ \mathbf{x_i^k} \end{aligned} \tag{3.3}$$

kde

$$t_{i,d}^k = \begin{cases} (x_{i,d}^k, lb_{i,d}^k) & if\ (rnd() < u_{i,d}^{k+1}) \\ \text{Identita} & inak \end{cases} \tag{3.4}$$

t^{rnd} je náhodná transpozícia

Navrhli sme 3 modifikácie tohto algoritmu. Modifikácie sa týkali výpočtu vzdialenosti medzi dvoma polohovými vektormi. Treba si uvedomiť, že polohový vektor reprezentuje permutáciu a musíme definovať, čo znamená rozdiel dvoch permutácií. Rozdiel dvoch permutácií (alebo dvoch polohových vektorov) budeme definovať ako rozdiel dvoch vektorov, teda urobáme operáciu rozdielu po súradniciach. Každá súradnica predstavuje poradie znaku, teda rozdiel súradníc je rozdielom poradia znakov. na Z toho okamžite vzniká otázka, aký vplyv má voľba poradia znakov na algoritmus. Nami navrhnuté zmeny sa týkali aj tejto otázky a boli motivované predpokladom, že použitie metriky, ktorá lepšie zodpovedá podstate problému, bude vykazovať lepšie výsledky (napríklad rýchlejšia konvergencia, presnejšie riešenie, ...).

V algoritme sa vzdialenosť medzi dvoma znakmi vypočíta ako absolútna hodnota rozdielu poradia znakov zoradených podľa abecedy.

$$d(char_i, char_j) = |i - j| \tag{3.5}$$

Na veľkosť rýchlosti má vplyv rozdiel $|pb - x|$ resp. $|lb - x|$. Čím väčší je tento rozdiel, tým väčšiu rýchlosť častica nadobúda a tým väčšia je pravdepodobnosť zmeny.

Príklad 3. *Majme usporiadané znaky abecedne (teda 'a' = 1, 'b' = 2, .., ' ' = 27) a pozrime sa na druhú súradnicu vektora rýchlosti (teda permutáciu znaku 'b').*

1. *Nech $pb^k_{i,2} = 24$, teda permutácia pb zamieňa 'b' v otvorenom texte za 'y' v šifrovom texte. Nech $x^k_{i,2} = 7$, teda permutáca x zamieňa 'b' s 'h'. Potom $\left|pb^k_{i,2} - x^k_{i,2}\right| = |24 - 7| = 17$.*

2. *Nech $pb^k_{i,2} = 4$, teda permutácia pb zamieňa 'b' v otvorenom texte za 'd' v šifrovom texte. Nech $x^k_{i,2} = 7$, teda permutácia x zamieňa 'b' s 'h'. Potom $\left|pb^k_{i,2} - x^k_{i,2}\right| = |4 - 7| = 3$.*

Rozdiel $\left|pb^k_{i,2} - x^k_{i,2}\right|$ bude v prvom prípade väčší. Keďže sa tak stane iba vďaka tomu, že 'y' je v abecede až ďaleko za 'd', a nie nutne z hlbšej príčiny súvisiacej s podstatou jednoduchej zámeny, hľadali sme možnosti, ktoré by lepšie reflektovali podstatu problému.

Testovali sme nasledovné varianty, ktoré sa líšia spôsobom výpočtu rozdielu medzi **gb** a **x** (indexy i, k kvôli jednoduchosti vynechávame)

1. $d(char_i, char_j) = |i - j|$ i, j určujú poradie znaku v abecednom zozname znakov. Na túto verziu sa budeme odvolávať ako na „pôvodnú" verziu PSO pre jednoduchú zámenu

2. preusporiadanie znakov podľa frekvenčnej analýzy

   $$d(char_i, char_j) = |\psi(i) - \psi(j)|,$$

   funkcia $\psi(i)$ určuje poradie znaku i v zozname znakov zoradených podľa frekvenčnej analýzy. Aby sme zabránili tomu, že indexovanie znakov podľa abecedy ovplyvňuje výpočet rýchlosti nelogickým spôsobom, zmenili sme indexovanie. Vzdialenosť dvoch znakov nebude závisieť na ich vzdialenosti v abecede, ale bude závisieť od rozdielu poradia ich frekvencíi v jazyku otvoreného textu. Inými slovami, rozdiel $|pb - x|$ bude

nadobúdať väčšie hodnoty pre tie súradnice, keď permutácia pb bude zobrazovať daný znak na znak so značne inou frekvenciou ako permutácia x. Naopak, keď obe permutácie pb aj x zobrazujú znak na znaky s veľmi podobnými frekvenciami, v tejto súradnici bude rozdiel malý. Funkcia ψ indexuje znaky abecedy podľa frekvenčnej analýzy znakov v danom jazyku, napr. podľa tabuľky 3.1. Najfrekventovanejší znak ␣bude mať index 1, druhý najfrekventovanejší (znak 'e') bude mať 2 atď.

3. rozdiel definovaný ako rozdiel medzi frekvenciami výskytu a nie ako rozdiel medzi poradiami,

$$d(char_i, char_j) = |freq(i) - freq(j)|,$$

funkcia $freq(i)$ vracia relatívny výskyt daného znaku v jazyku v percentách. Myšlienka je rovnaká ako v predchádzajúcom prípade. Nebude však rozhodovať poradie znakov v tabuľke usporiadanej podľa ich frekvencíi (funkcia ψ), ale priamo hodnoty týchto frekvencíi (funkcia $freq(i)$).

4. digitálny model vzdialenosti,

$$d(char_i, char_j) = \delta_{i,j},$$

kde $\delta_{i,j}$ je Kroneckerovo delta. Pri výpočte rýchlosti by sme chceli, aby sa častica rýchlejšie dostala zo svojej súčasnej pozície do doteraz svojej najlepšej pozície (a obdobne zo súčasnej pozície do lokálne najlepšej pozície). Pri takto zvolenej metrike každý rozdiel medzi dvoma rôznymi znakmi bude mať vzdialenosť 1.

Parametre PSO sme nastavili tak, aby boli porovnateľné s prístupom v článku [4], kde riešili problém jednoduchej zámeny pomocou troch rôznych algoritmov: simulované zažíhanie, genetický algoritmus a zakázané prehľadávanie (tabu search). Naším cieľom bolo porovnať PSO s uvedenými algoritmami na tom istom probléme a s rovnakými alebo obdobnými nastaveniami.

Rozmiestnili sme 500 častíc, ktoré spravili 200 iterácií. Frekvenčnú analýzu znakov sme robili z knihy Pes Baskervillský od Arthura Conana Doyla (stiahnutej zo stránky
http://www.gutenberg.org.
Pracovali sme s 27 znakmi (26 znakov anglickej abecedy a medzera). Všekty interpunkčné znamienka sme ignorovali. Každý útok sme robili na 200 rôznych, náhodne vybraných textoch rôznej dĺžky z už spomínanej knihy Pes

Baskervillský. Na každý z týchto 200 textov sme algoritmus spustili 3-krát a pracovali sme len s najlepším z týchto 3 pokusov. Teda s tým, v ktorom mala účelová funkcia najlepšiu hodnotu. Účelová funkcia bola zvolená nasledovne:

$$f = \sum_{c \in \{A..Z\}} \left| F^U_{(c)} - K^U_{(c)} \right| + \sum_{c,d \in \{A..Z\}} \left| F^B_{(c,d)} - K^B_{(c,d)} \right| \tag{3.6}$$

$F^U_{(c)}$ je frekvencia výskytu znaku c v otvorenom texte
$K^U_{(c)}$ je frekvencia výskytu znaku c v texte, ktorý vznikne odšifrovaním pomocou konkrétnej permutácie
$F^B_{(c,d)}$ je frekvencia bigramu (c,d) v otvorenom texte
$K^B_{(c,d)}$ je frekvencia výskytu bigramu v texte, ktorý vznikne odšifrovaním pomocou konkrétnej permutácie

Frekvencie jednotlivých znakov v anglickej abecede je v tabuľke 3.1, frekvencia 10 najčastejších bigramov je v tabuľke 3.2.

poradie	znak	frekvencia	poradie	znak	frekvencia
1	␣	19.8198%			
2	e	9.7886%	15	w	2.1090%
3	t	7.3863%	16	c	1.9304%
4	a	6.4299%	17	y	1.7116%
5	o	6.3072%	18	f	1.6836%
6	i	5.5735%	19	g	1.4194%
7	h	5.4605%	20	p	1.2323%
8	n	5.3495%	21	b	1.1637%
9	s	4.9988%	22	v	0.8499%
10	r	4.6938%	23	k	0.6302%
11	d	3.4122%	24	x	0.1224%
12	l	3.1676%	25	q	0.0624%
13	u	2.4411%	26	j	0.0513%
14	m	2.1655%	27	z	0.0395%

Tabuľka 3.1: Frekvenčná analýza znakov v anglickom jazyku.

3.3 Výsledky testov

Urobili sme 2 porovnania

poradie	bigram	frekvencia
1	e␣	3.9539%
2	␣t	2.8473%
3	th	2.4535%
4	he	2.2192%
5	t␣	2.1743%
6	d␣	2.1639%
7	s␣	2.0444%
8	␣a	1.9585%
9	␣h	1.6930%
10	␣i	1.6930%

Tabuľka 3.2: Frekvenčná analýza bigramov v anglickom jazyku.

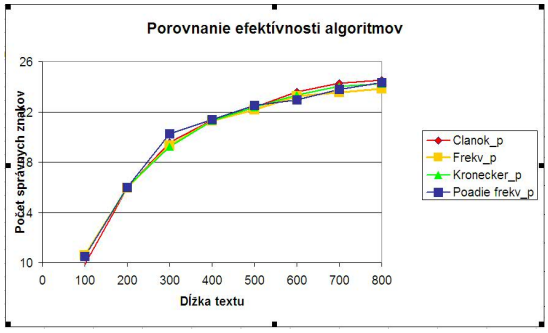
1. porovnali sme výsledky horeuvedených štyroch modifikácií algoritmu PSO, ktoré sme aplikovali na problém jednoduchej zámeny. Všetky 4 verzie boli testované s nulovou aj nenulovou hodnotou parametra w (viď tabuľku

   Pri prvom teste s porovnaním 4 verzií PSO („pôvodná" verzia a 3 modifikácie) sa výsledky signifikantne nelíšili. Jediný výraznejší rozdiel sa prejavil pri dĺžke textu 100 znakov, kde navrhnuté modifikácie vykazovali približne o 7% lepší výsledok ako „pôvodný" algoritmus. S nárastom dĺžky textu sa rozdiely znížili na maximálne $2.5\% - 3\%$. Všimnime si, že výsledky pre 4. variantu, kde rozdiel dvoch znakov sa rovná jednotke alebo nule v závislosti, či sa znaky rovnajú alebo nie, dáva rovnaké výsledky ako ďalšie modifikácie, kde sme sa snažili zaviesť rôzne metriky, ktoré by čo najpresnejšie vystihovali podstatu problému. Z tejto skutočnosti sme vyvodili, že v prípade jednoduchej zámeny dôležitá je len skutočnosť, či dva znaky sa rovnajú alebo nie, a algoritmus nie je závislý od konkrétnej metriky a kvantifikácie vzdialenosti.

   Aby sme overili toto tvrdenie urobili sme všetky testy ešte raz s nulovou hodnotou parametra w, teda na vektor rýchlosti vplýva len vzdialenosť aktuálnej polohy častice od svojej najlepšej polohy v kombinácii so vzdialenosťou polohy častice od najlepšej polohy svojich susedov. Tým sme vylúčili zotrvačnosť a jediný faktor, ktorý určuje nový vektor rýchhlosti je vzdialenosť 2 polohových vektorov. V tabuľke 3.5 sú uvedené výsledky pre dĺžku reťazca 200. Výsledky s nulovou a nenulovou hodnotou sa prakticky nelíšia, čo podporuje naše vysvetlenie.

Dĺžka textu	Pôvodný		Rozdiel frekvencií		Kronecker		Poradie frekvencií	
	x	s	x	s	x	s	x	s
100	9,84	3,81	10,59	3,59	10,48	3,60	10,47	3,56
200	15,94	3,26	15,94	3,63	16,06	3,43	16,00	3,06
300	19,58	3,12	19,31	2,83	19,23	2,78	20,25	2,78
400	21,36	2,49	21,31	2,56	21,28	2,73	21,37	2,76
500	22,37	2,60	22,16	2,45	22,39	2,35	22,51	2,47
600	23,61	2,15	23,25	2,03	23,33	2,20	22,94	2,13
700	24,30	2,03	23,57	1,98	24,05	1,85	23,80	1,94
800	24,52	1,47	23,85	1,85	24,21	1,90	24,34	1,80

Tabuľka 3.3: Porovnanie rôznych metód riešenia: stĺpec „x" označuje priemernú hodnotu počtu správnych znakov, stĺpec „s" ozncuje štandardnú odchýlku. Hodnoty v prvom riadku označujú dľžku textu.



Obr. 3.2: Priebeh funkcie znázorňujúcej počet správnych znakov v závislosti od dĺžky textu.

V ďalšom sme sa sústredili na porovnanie PSO algoritmu s tromi algoritmami: SA metóda simulovaného zažíhania, GA genetický algoritmus, TS zakázané prehľadávanie. Výsledky získané PSO algoritmom sú lepšie než pomocou genetického algoritmu alebo zakázaným prehľadávaním. Výsledky pomocou simulovaného zažíhania sú mierne lepšie než PSO. Je nutné však poznamenať, že pre PSO výsledky boli získané bez špeciálneho ladenia jednotlivých parametrov w, c_1, c_2.

Posledné testy, ktoré sme urobili súviseli s poradím transpozícií vo vzťahu (??). Ako vieme skladanie transpozícií nie je obecne komutatívne (okrem prípadu dizjunktných cyklov). Teda ak zmeníme poradie transpozícií, zmení sa aj výpočet. Vo vzťahu (??) skladáme transpozície v prirodzenom poradí (začíname prvou súradnicou a zisťujeme, či vektor rýchlosti

Dĺžka textu	SA		GA		TS	
	x	s	x	s	x	s
100	10,73	4,70	7,74	4,82	6,02	4,04
200	17,70	3,40	14,17	6,13	12.76	6,32
300	21,07	2,72	18,77	6,01	17,33	6,48
400	22,86	2,27	21,72	4,61	19,45	6,34
500	23,73	2,16	22,44	4,37	21,77	5,47
600	24,50	1,86	23,69	3,68	23,50	4,14
700	24,72	1,73	23,82	3,39	23,68	4,42
800	25,08	1,59	24,64	2,23	24,18	4,12

Tabuľka 3.4: Výsledky aplikácie rôznych algoritmov na riešenie jednoduchej zámeny (výsledky okrem PSO sú z práce [4]).

w	Pôvodný		Rozdiel frekvencií		Kronecker		Poradie frekvencií	
	x	s	x	s	x	s	x	s
$0,5 + rnd()/2$	15,96	3,23	15,44	3,29	16,28	3,01	15,97	3,25
0	15,94	3,26	15,94	3,63	16,06	3,43	16,00	3,06

Tabuľka 3.5: Porovnanie výsledkov pre nulovú a nenulovú hodnotu parametra w pre dĺžku textu 200.

indikuje zmenu, ak áno, tak ju prevediem a ideme na ďalšiu súradnicu. Tento proces by sme mohli robiť aj v opačnom poradí a začať od poslednej súradnice a postupovať k prvej súradnici, alebo ľubovoľným iným poradím. Každé poradie môže generovať iný výsledok. Preto sme urobili 3 rôzne poradia

(a) „rastúce" poradie od prvej súradnice po 27. súradnicu

(b) „klesajúce" poradie od 27. súradnice po prvú súradnicu

(c) náhodne generované poradie

Pre káždé poradie sme spustili algoritmus PSO pre jednoduchú zámenu a výsledky sme uviedli v tabuľke 3.6 pre dĺžku textu 200 a v tabuľke 3.7 pre dĺžku textu 400.

Ako vidíme, výsledky sú skoro totožné.

	Pôvodný		Rozdiel frekvencií		Kronecker		Poradie frekvencií	
	x	s	x	s	x	s	x	s
klesajúce	16,62	3,65	15,51	3,17	16,14	3,06	16,75	2,86
rastúce	15,94	3,26	15,94	3,63	16,06	3,43	16,00	3,06
náhodné	16,33	3,19	15,76	3,08	16,29	3,11	15,91	3,56

Tabuľka 3.6: Porovnanie výsledkov jednoduchej zámeny pre rôzne poradie transpozícií vo vzťahu (3.3) pre dĺžku textu 200.

	Pôvodný		Rozdiel frekvencií		Kronecker		Poradie frekvencií	
	x	s	x	s	x	s	x	s
klesajúce	21,24	2,78	20,95	2,57	21,64	2,94	21,55	2,92
rastúce	21,36	2,49	21,31	2,56	21,28	2,73	21,37	2,76
náhodné	21,51	2,55	20,85	2,83	20,92	2,80	21,01	2,59

Tabuľka 3.7: Porovnanie výsledkov jednoduchej zámeny pre rôzne poradie transpozícií vo vzťahu (3.3) pre dĺžku textu 400

Kapitola 4

Popis DES a priamy útok na DES pomocou PSO

V tejto kapitole sa zaoberáme možnosťou použitia PSO na útok na DES. Popíšeme ako priamym spôsobom aplikovať PSO na nájdenie kľúča. V prvej časti kapitoly popíšeme algoritmus šifrovania pomocou DES a uvedieme niektoré známe útoky na DES. V druhej časti ukážeme ako sa dá PSO použiť na nájdenie kľúča pre DES. Za týmto účelom sme testovali DES s rôznym počtom kôl. V prípade jedno a dvojkolovej verzie DES sa nám podarilo pomocou PSO nájsť kľúč. Pre 3 kolá existuje šanca ako popísaným spôsobom nájsť klúč, ale pre vyšší počet kôl daný prístup nevedie k úspechu. V závere sme analyzovali príčiny a uvádzame dôvody, prečo tomu tak je.

4.1 Popis DES

Skratka DES znamená Data Encryption Standard, teda štandard pre šifrovanie dát. V sedemdesiatých rokoch minulého storočia vznikla v USA potreba štandardu na šifrovanie údajov v civilných štátnych organizáciach. Firma IBM vyvinula DES zo skoršej šifry nazývanej Lucifer. DES sa stal oficiálnym štandardom v roku 1977 a odvtedy bol masívne používaný na celom svete až do konca devedesiatych rokov. Podrobný popis DES je možné nájsť v [20].

Pri navrhovaní šifry DES bol kladený veľký dôraz na to, aby sa dal algoritmus implementovať v hardvéri a docielila sa vyššia rýchlosť šifrovania. Samotný proces šifrovania a dešifrovania prebieha v 16 kolách.

V každom kole sa používa miešanie bitov (permutačné boxy, P-boxy), nelineárne funkcie (substitučné boxy, S-boxy) a lineárne sčítanie pomocou XOR-u (exkluzívny OR). Pri tvorbe podkľúčov pre jednotlivé kolá sa ešte používa triviálna operácia „posun". Všetky tieto operácie sú jednoduché na hardvérovú implementáciu (ide o XOR, rotácie a malé vyhľadávacie tabuľky). Zároveň sa ukázalo veľmi výhodným použiť Feistelovo schéma. Vďaka nemu má DES skoro identický proces šifrovania a dešifrovania.

DES je bloková šifra. Pracuje s blokom dlhým 64 bitov. Používa kľúč dlhý 64 bitov, z čoho je 8 bitov kontrolných a teda 56 signifikatných. Výstupom sú opäť bloky dlhé 64 bitov. DES je symetrická šifra, to znamená, žč rovnaký kľúč sa používa na šifrovanie aj na odšifrovanie.

DES by sme mohli rozdeliť na dva dôležité logické bloky:

(a) jednotlivé kolá kde sa aplikuje rovnaký postup na vstupné údaje

(b) expanzia kľúča na kolové kľúče.

Uveďme 3 základné kroky šifrovania DES: počiatočná permutácia, 16 kôl a záverečná permutácia. Nasleduje popis týchto krokov. Obrázok 4.1 graficky znázorňuje 16-kolový algoritmus DES.

(a) Nech x značí otvorený text. Majme počiatočnú permutáciu IP(initial permutacion). Označme x_0 výsledok aplikácie permutácie IP na vstupný reťazec x. Teda $x_0 = IP(x)$. Následne rozdeľme x_0 na dva 32-bitové reťazce, prvých 32 bitov označme L_0 a druhých 32 bitov R_0.

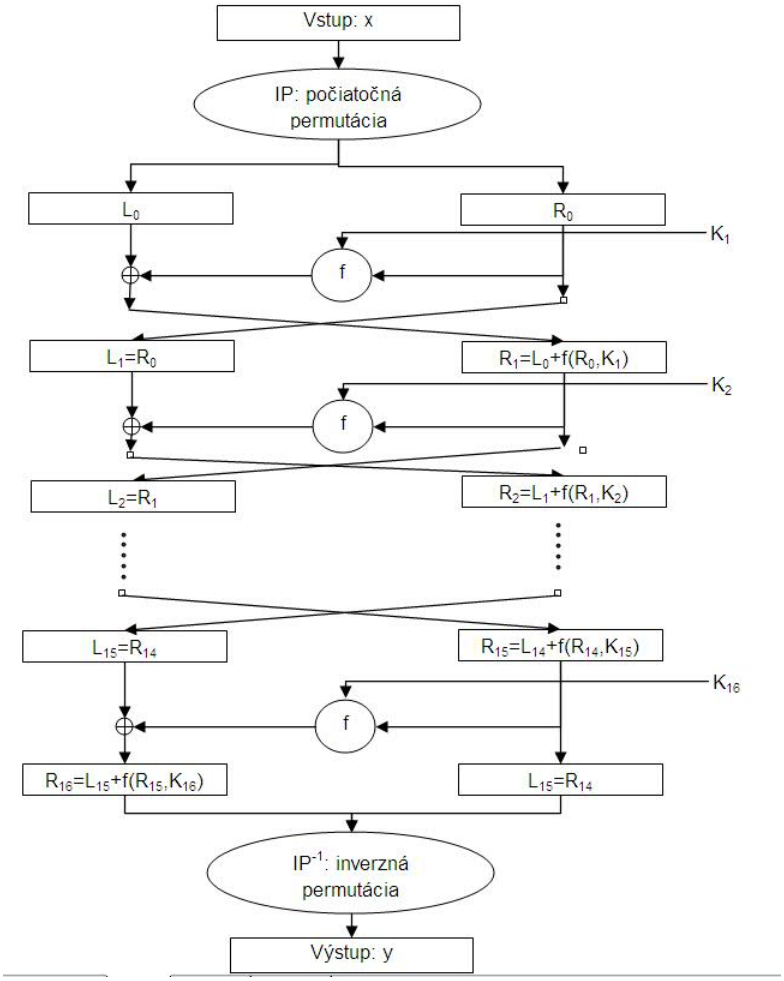
(b) Pre $1 \leq i \leq 16$ vykonaj

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

kde $\oplus$ značí XOR, f je funkcia, ktorú popíšeme neskôr a J_i je kľúč pre dané kolo. Tento kľúč je dlhý 48 bitov a je funkciou 56 bitového pôvodného kľúča K.

(c) Po aplikácii 16-tich kôl na výsledný reťazec bitov $R_{16}L_{16}$ aplikujeme inverznú permutáciu IP^{-1} a dostaneme šiforvý text y.

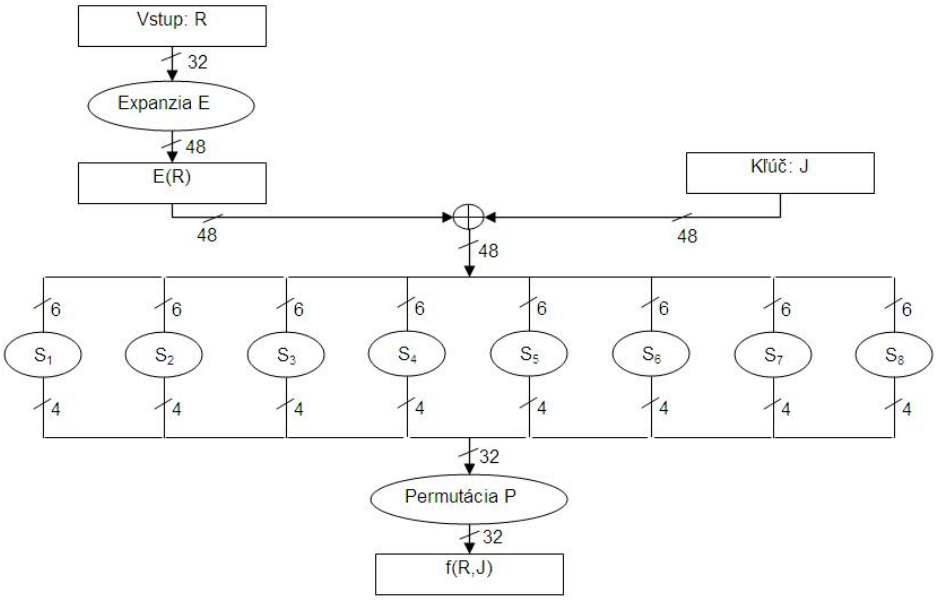
Vstupom pre funkciu f sú dva bitové reťazce. Prvý reťazec R dlhý 32 bitov a druhý reťazec J dlhý 48 bitov. Výstupom je bitový reťazec o dĺžke 32 bitov.



Obr. 4.1: Grafické znázornenie algoritmu DES

Obrázok (4.2) graficky znázorňuje funkciu f.

V prvom kroku je reťazec R rozšírený na reťazec dlhý 48 bitov pomocou fixnej expanzívnej funkcie E. Funkcia E je permutačná funkcia, ktorá polovicu bitov (16) zduplikuje. V druhom kroku spočítame $E(R) \oplus J$ a výsledný reťazec rozdelíme na osem 6-bitových reťazcov. Označíme ich $B = B_1B_2B_3B_4B_5B_6B_7B_8$. Reťazec B_j je vstupom do príslušného S-boxu S_j, $1 \leq j \leq 8$. Každý S-box S_j je pole rozmeru 4×16, kde na každom políčku je 4-bitový reťazec. Reťazec B_j slúži na výpočet adresy (čísla riadku a šisla stĺpca) c danom S-Boxe. Výstupom z S=Box je reťazec zapísane v S-Boxe na adesesa určenej vstupom B_j. Výstup označme C_j. Dostávame $C_j = S_j(B_j)$. Na záver sa na bitový reťazec



Obr. 4.2: Grafické znázornenie funkcie f v jednom kole DES-u

$C = C_1C_2C_3C_4C_5C_6C_7C_8$ aplikuje fixná permutácia P. Máme $f(R, J) = P(C)$.

Ostáva nám popísať spôsob generovania kľúčov pre jednotlivé kolá z 56 signifikantných bitov pôvodného kľúča K. Týchto 56 bitov zpermutujeme pomocou permutácie PC-1 (permuted choice), zapíšeme PC-1$(K) = C_0D_0$, kde C_0(resp. D_0) je prvých (resp. druhých) 28 bitov PC-1(K). Keďže je 16 kôl, potrebujeme 16 „podkľúčov", a preto pre $1 \leq i \leq 16$ vypočítame:

$$\begin{aligned} C_i &= LS_i(C_{i-1}), \\ D_i &= LS_i(D_{i-1}) \text{ a teda} \\ J_i &= \text{PC-2}(C_iD_i). \end{aligned}$$

LS_i reprezentuje cyklický posun doľava buď o jednu alebo dve pozície, v závislosti na hodnote i. Ak $i = 1, 2, 9, 16$ ide o posun o jednu pozíciu, inak o dve. PC-2 je ďalšia permutácia.

4.1.1 Úloha S-boxov

Najdôležitejšou časťou pre bezpečnosť DES-u sú S-boxy, keďže ide o jeho jedinú nelineárnu časť. Jedná sa o substitučné tabuľky, ktoré realizujú

zámenu vstupnej šestice bitov za štvoricu výstupných bitov. Nároky kladené na dizajn S-boxov boli dlhé roky utajované(na podnet organizácie NSA). Boli zverejnené až v roku 1994 [6], po prvých verejných článkoch o diferenčnej kryptoanalýze [19],[2]. Návrhové kritériá na S-boxy boli nasledovné [8]:

P0 Každý riadok v každom S-box je permutáciou čísel $0, \ldots, 15$.

P1 Žiadny S-box nie je lineárnou alebo afinnou funkciou svoho vstupu.

P2 Zmena jedného vstupného bitu do S-boxu má za následok zmenu minimálne 2 výstupných bitov.

P3 Pre ľubovoľný S-box a ľubovoľný vstup x, $S(x)$ a $S(x \oplus 001100)$ sa líšia aspoň v dvoch bitoch.

P4 Pre ľubovoľný S-box a ľubovoľný vstup x, a pre $e, f \in \{0, 1\}$, $S(x) \neq S(x \oplus 11ef00)$.

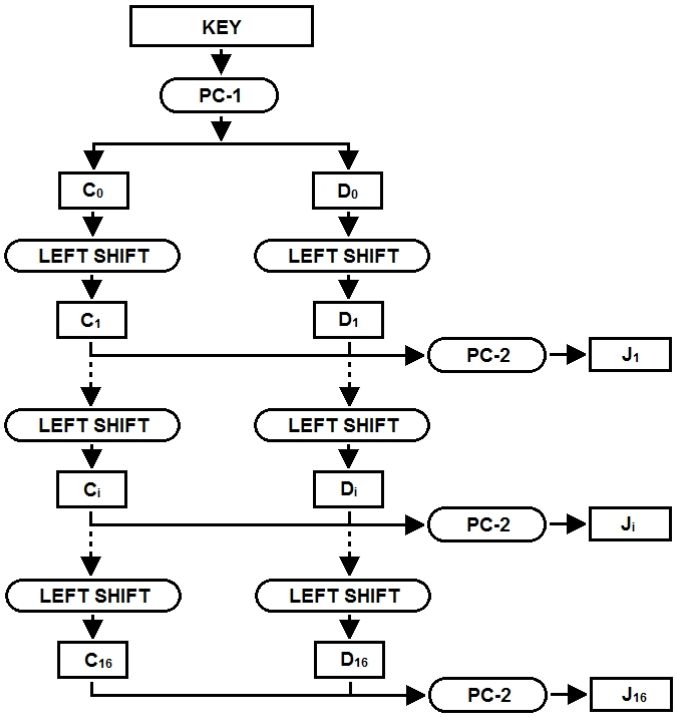
P5 Pre ľubovoľný nenulový 6-bitový rozdiel vstupov najviac 8(z možných 32) párov vstupov, ktoré majú tento rozdiel, môže viesť k rovnakému výstupnému rozdielu.

Obrázok (4.3) graficky znázorňuje generovanie podkľúčov pre jednotlivé kolá DES-u

Dlho otvorenou otázkou bolo aj usporiadanie S-Boxov. Existuje $8! = 40320$ rôznych možností usporiadania. Otázka bola, či jednoduchým preusporiadaním S-boxov je možné vylepšiť odolnosť DES-u voči kryptoanalytickým útokom. V [17] je ukázané, že existuje síce lepší spôsob preusporiadania S-Boxov aby sa sťažil útok pomocou diferenčnej kryptoanalýzy, ale rozdiel nie je relevantný. V tom istom článku je tiež ukázané, že žiadne preusporiadanie S-Boxov nedáva ochranu pred takýmto spôsobom útoku.

4.1.2 Striktné lavinové kritérium

Dôležitou vlastnosťou pre bezpečnosť blokovej šifry je striktné lavínové kritérium pre kľúč (strict key avalanche criterion SKAC). Nastáva vtedy, keď pre fixovaný otvorený text sa každý bit šifrového textu zmení s pravdepodobnosťou 50% vždy, keď sa zmení jeden bit kľúča. DES spĺňa SKAC [9].



Obr. 4.3: grafické znázornenie generovania podkľúčov pre jednotlivé kolá DES-u zo šifrovacieho kľúča

4.2 Známe útoky na DES

Za dobu životnosti DES-u sa investovalo veľké množstvo úsilia do jeho prelomenia, ale napriek tomu nie je známa žiadna ľahká alebo rýchla metóda.

Hrubá sila je najjednoduchším typom útoku, v ktorom sa postupne snažíme vyskúšať všetky možné kľúče. Potrebujeme jednu dvojicu otvoreného a jemu prislúchajúceho šifrového textu. Keďže kľúč pre DES má len 56 signifikantných bitov, stáva sa DES v dnešnej dobe týmto útokom ľahko zlomiteľný. Krátky kľúč bol jedným z dôvodov nahradenia DES novým štandardom, algoritmom AES. V roku 1996 bolo možné pomocou vytvorenia špecializovaného systému vybudovaným pre takýto účel odšifrovať DES v priemere za 35 minút [1].

Diferenčná kryptoanalýza je aplikovaná predovšetkým na blokové

šifry, aj keď sa dá použiť aj pri prúdových šifrách, či hešovacích funkciách. Ide o útok s možnosťou voľby otvoreného textu. Myšlienka za touto metódou je, že si zvolíme dva otvorené texty s nami vybraným bitovým rozdielom. Tieto zašifrujeme rovankým kľúčom a skúmame bitový rozdiel zašifrovaných textov. Inými slovami, zaujíma nás, ako rozdiel vstupu vplýva na rozdiel výstupu. Diferenčná kryptoanalýza je založená na nerovnomernej distribúcii výstupov S-boxov a využití tejto informácie na nájdenie kľúča.

Na úspešné prelomenie 16 kolového DES-u je však potrebných 2^{47} zvolených otvorených textov a k nim prislúchajúcich šifových textov [3]. Práve pre náročnosť získať takéto množstvo dvojíc, je tento útok skôr len teoretický. Diferenčnej kryptoanalýze sa budeme bližšie venovať v kapitole 5.

Lineárna kryptoanalýza je popísaná v [17]. Ide o útok so znalosťou otvoreného textu. Umožňuje nájdenie DES kľúča na základe analyzovania 2^{43} známych otvorených textov.

4.3 Definovanie účelovej funkcie pre útok prostredníctvom PSO

Úspech PSO do značnej miery závisí od správnej definície účelovej funkcie. Potrebujeme funkciu, ktorá by kľúče blízke k správnemu kľúču (napr. blízke v zmysle Hammingovej vzdialenosti) ohodnotila lepšie ako kľúče, ktoré sú vzdialenejšie. Ako uvidíme, nájsť takúto funkciu je veľmi ťažké.

Chceme previesť útok so znalosťou otvoreného textu (known plaintext attack), teda predpokladáme, že máme k dispozícii n otvorených textov, každý dlhý 64 bitov a k nim prislúchajúce šifrové texty zašifrované kľúčom K. Množinu týchto dvojíc značíme $\{(M_i, C^K(M_i), i = 1, \ldots n\}$, pričom $C^K(M_i)$ je šifrový text zašifrovaný kľúčom K odpovedajúci otvorenému textu M_i. Nech T je ľubovoľný kľúč. Hodnotu účelovej funkcie vypočítame nasledovne.

$$h(T) = \frac{1}{n} \sum_{i=1,..,n} d_H(C^K(M_i), C^T(M_i)), \tag{4.1}$$

kde d_H je Hammingova vzdialenosť medzi dvoma bitovými reťazcami a K je vopred definovaný DES kľúč.

- Každý otvorený text M_i zašifrujeme kľúčom T, čím dostaneme šifrový text $C^T(M_i)$.
- Pre každú dvojicu šifrových textov $(C^K(M_i), C^T(M_i))$ vypočítame Hammingovu vzdialenosť $d_H(C^K(M_i), C^T(M_i))$.
- Účelová funkcia je aritmetický priemer Hammingových vzdialeností šifrového textu vytvoreného DES kúčom a šifrového textu vytvoreného navrhovaným kľúčom pre všetky otvorené texty $M_i, i = 1, 2, \ldots, n$.

Aby sme preskúmali vhodnosť tejto účelovej funkcie, definujme novú funkciu h^*.

Definícia 4. Nech $j = 1, 2, \ldots, 16$ označuje počet kôl DES-u, ktoré sa vykonajú počas šifrovania. Nech K_1, K_2 sú 2 kľúče pre DES, d_H nech označuje Hammingovu vzdialenosť medzi dvoma bitovými reťazcami rovnakej dĺžky, $M_i, i = 1, 2, \ldots, 500$ sú náhodne vygenerované 64-bitové reťazce. Nech $C_j^K(M)$ označuje šifrový text pre vstupný reťazec M pomocou kľúča K po aplikácii j kôl DES-u. Potom funkciu $h_j^*(K_1, K_2)$ definujeme nasledovne

$$h_j^*(K_1, K_2) = \frac{1}{500} \sum_{i=1,..,500} d_H(C_j^{K_1}(M_i), C_j^{K_2}(M_i)), \tag{4.2}$$

Chceli sme skúmať ako sa bude správať účelová funkcia (4.1) v závislosti od nasledovných faktorov

- počtu kôl DES-u,
- Hammingovej vzdialenosti dvoch kľúčov.

Najprv sme vygenerovali 500 náhodných textov $M_1, \ldots, M_{500}$, každý dlhý 64 bitov. Následne pre každé $z = 1, 2, \ldots, 8$ sme vykonali nasledovné:

- vygenerovali sme 500 náhodných dvojíc kľúčov K_1, K_2 tak, že $d_H(K_1, K_2) = z$ (pre $z = 1$ sa kľúče opakovali),
- pre každé $j = 1, 2, \ldots, 16$ sme vypočítali hodnutu $h_j^*(K_1, K_2)$.

V tabuľke 4.1 sú uvedené hodnoty funkcie h_j^*, ktorá nie je identická s účelovou funkciou. Ak však fixujeme jeden z kľúčov, tak sa funkcie rovnajú. Tabuľka poukazuje, že nami definovaná účelová funkcia nedokáže kopírovať šifrovací proces DESu. Pre kľúče, ktoré majú Hammingovú

	Počet rozdielných bitov v kľúči							
Počet kôl	1	2	3	4	5	6	7	8
1	2,184	4,112	5,839	7,320	8,635	9,800	10,807	11,667
2	11,501	17,068	20,343	22,526	24,178	25,525	26,622	27,583
3	24,215	28,775	30,421	31,188	31,537	31,732	31,808	31,880
4	30,826	31,810	31,968	31,982	31,996	31,995	32,011	32,005
5	31,960	31,972	31,992	32,011	32,007	31,985	31,991	32,003
6	32,053	31,992	31,991	31,993	32,005	32,016	32,002	31,983
7	32,027	31,990	32,003	32,005	32,000	31,993	32,006	31,998
8	32,000	31,997	32,009	31,991	32,003	32,000	31,993	31,995
9	31,967	32,009	32,000	32,012	31,996	31,999	31,988	32,003
10	32,007	31,996	32,011	31,997	31,995	31,993	31,988	31,999
11	32,024	32,010	32,009	31,993	31,989	32,004	31,998	32,010
12	32,013	32,000	32,003	32,000	32,000	32,015	31,999	32,006
13	32,031	32,000	31,995	31,997	32,010	31,995	32,020	32,012
14	32,007	31,998	32,002	31,995	31,997	32,009	32,003	31,996
15	32,014	32,003	31,983	31,992	32,001	32,004	32,012	32,000
16	31,970	32,003	31,999	32,002	32,006	31,996	31,991	31,999

Tabuľka 4.1: Priemerný počet bitov, v ktorých sa šifrové texty líšia po šifrovaní dvojicou kľúčov s Hammingovou vzdialenosťou $z = 1, 2, \ldots, 8$ a po aplikácii $j = 1, 2, \ldots, 16$ kôl DES-u.

vzdialenosť 1 (teda sa líšia len v jednom bite) sa šifrové texty už po 4 kolách DES-u líšia v priemere o 32 bitov, čo predstavuje 50% všetkých bitov. Tento výsledok nie je prekvapivý, lebo ako sme uviedli v časti 4.1.2, DES spĺňa striktné lavínové kritérium. Ak sa kľúče líšia o viac bitov, Hammingova vzdialenosť textov, ktoré vzniknú po šifrovaní otvoreného textu týmito dvoma kľúčmi je skoro konštantná a v priemere sa pohybuje okolo hodnoty 32.

Počas útoku nemáme vedomosť o skutočnom kľúči. To, čo vieme pozorovať pri nami popísanom útoku je len porovnanie šifrových textov vytvorených skutočným kľúčom a nami navrhovaným kľúčom. Ako však vyplýva z tabuľky 4.1 tento rozdiel pri viac ako 4 kolách DES-u je skoro konštantný a teda nedá sa z tohto porovnania získať žiadna informácia. Práve táto neschopnosť navrhnutej účelovej funkcie ohodnotiť kľúč je dôvodom zlyhania popísaného útoku pomocou PSO na DES.

V súvislosti s uvedeným faktom nám vyvstáva niekoľko otázok:

(a) je možné s nami definovanou účelovou funkciou zlomiť DES, ak limitujeme počet kôl? Ak áno, koľko kolový DES sa dá takýmto spôsobom zlomiť?

(b) dá sa definovať účelová funkcia, ktorá by nemala spomínané nedostatky? Ak áno, ako?

Výsledky uvedené v tabuľke by nasvedčovali, že existuje šanca uvedeným spôsobom odšifrovať 4-kolový DES (pre viac kôl sú rozdiely v hodnote účelovej funkcie v závislosti od počtu rozdielnych bitov prakticky nemerateľné). Obdobné výsledy - t.j. zlomenie 4-kolového DES sú známe pomocou genetického algoritmu [25]. Nám sa podarilo odšifrovať dvojkolový DES s použitím PSO algoritmu a obdobným prístupom ako v [25]. Podrobnejší popis uvádzame v časti (4.4.1).

Druhá otázka je oveľa ťažšia. Podľa nás na nájdenie vhodnej účelovej funkcie je potrebné do nej zakomponovať oveľa hlbšie znalosti o fungovaní samotného algoritmu, resp. využiť kryptoanalytické metódy pri vyhodnotení konkrétneho kľúča účelovou funkciou.

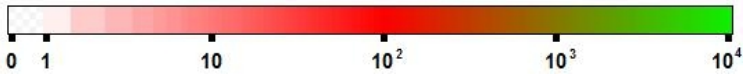
Na ilustráciu, prečo sa nami popísaným spôsobom nedá zlomiť DES, uvádzme 4 obrázky pre 1-kolový, 2-kolový, 3-kolový a 4-kolový DES. Popis obrázkov je nasledovný:

- náhodne sme vybrali jeden 56-bitový kľúč pre DES, označme ho K. Vygenerovali sme 200 náhodných reťazcov $M_i,\ i = 1, \ldots, 200$ dĺžky 64.
- nech $z = 0, 1, 2, \ldots, 55$ predstavuje Hammingovu vzdialenosť od definovaného kľúča K. Pre každé z sme vygenerovali $N = 10000$ náhodných kľúčov $T_k,\quad k = 1, \ldots, N$, ktoré mali Hammingovu vzdialenosť rovnú z. Teda $d_H(K, T_k) = z$.
- pre každé T_k sme vypočítali hodnotu účelovej funkcie $h(T_k)$. Účelová funkcia bola definovaná vzťahom (4.1), pričom n=200.
- hodnoty $h(T_k)$ pre $k = 1, N$ sme rozdelili do intervalov

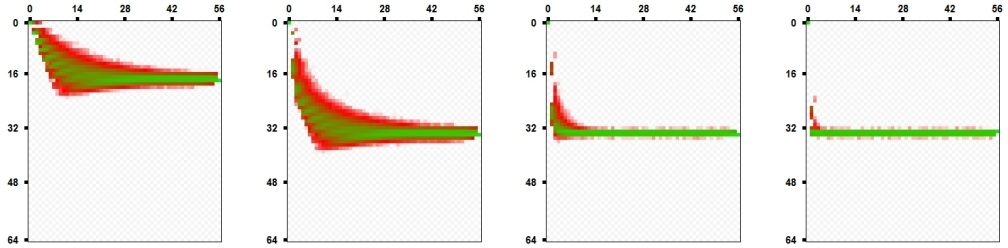
$$I_0 = [0, 1)\,, I_1 = [1, 2)\,, \ldots, I_{63} = [63, 64]$$

  a vytvorili sme tabuľku početností nad týmito intervalmi (pre každý interval sme spočítali koľko hodnôt účelovej funkcie bolo v rozpätí daného intervalu)
- každej početnosti sme priradili farbu (viď obrazok 4.4. Každý z obrázkov má 56×64 políčok, 56 stĺpcov (každý stĺpec predstavuje Hammingovu vzdialenosť od kľúča K) a 64 riadkov (každý riadok

predstavuje jeden interval I_s, pre hodnoty účelovej funkcie). Na jednotlivých obrázkoch nejaká farba v políčku $(riad, stlp)$ predstavuje početnosť účelovej funkcie pre interval I_{riad} pre kľúč s Hammingovou vzdialenosťou $stlp$.



Obr. 4.4: Farebná škála znázorúujúca priradnie farieb jednotlivým početnostiam.



Obr. 4.5: Obrázky znázorňujú početnosti kľúčov, v závislosti od Hammingovej vzdialenosti od stanoveného kľúča K a od hodnoty účelovej funkcie

Všimnime si, že počet rôznych kľúčov s Hammingovou vzdialenosťou z od daného kľúča K je $\binom{56}{z}$. Teda v stĺpcoch, ktoré sú pri pravom a ľavom okraji obrázkov je len veľmi málo kľúčov. Obrázky pre 1-kolový a 2-kolový DES naznačujú, že PSO algoritmus má dobrú šancu nachádzať kľúče so stále lepšou hodnotou účelovej funkcie a skonvergovať k hľadanému kľúču (V pravom hornom rohu obrázku). Pre 3-kolový DES sa však PSO musí trafiť do kľúčov s Hammingovou vzdialenosťou menšou než 8, aby algoritmus skonvergoval. Pre 4-kolový DES by PSO muselo nájsť kľúč s menšou Hammingovou vzdialenosťou než 4. Hodnoty účelovej funkcie pre kľúče s väčším rozdielom než 10 sú však skoro konštantné a tak nedávajú žiadnu indíciu pre konvergenciu algoritmu k lepším hodnotám.

4.4 Útok na 2-kolový DES priamou aplikáciou PSO

V tejto časti popíšeme algoritmus, ktorý sme použili pri útoku na DES s rôznym počtom kôl. Išlo o útok so znalosťou otvoreného textu. Pracovali sme s 20 náhodne vybratými reťazcami, ku ktorým sme poznali aj šifrový text. Použili sme diskrétnu variantu PSO popísanú v časti **??**. Štandardná implementácia PSO nie je schopná úspešne nachádzať celý kľúč po jednom behu. Preto sme zvolili obdobný postup ako v článku [**??**], v ktorom sa autorom podarilo úspešne zlomiť 4-kolový DES pomocou genetického algoritmu.

- algoritmus PSO sme aplikovali opakovane 40-krát za sebou,
- na konci každého behu sme si zapamätali nájdený kľúč a hodnotu účelovej funkcie pre nájdený kľúč,
- po 40 behoch sme vypočítali aritmetický priemer všetkých hodnôt účelovej funkcie a vybrali sme len tie kľúče, pre ktoré hodnota účelovej funkcie bola lepšia než priemerná hodnota,
- v takto vyselektovaných kľúčoch sme vybrali tie bity, ktorých hodnota bola rovnaká pre 80% kľúčov,
- celý beh sme opakovali dovtedy, kým sme nenašli všetky bity kľúča, alebo nedosiahli 50 iterácií.

```
Initializacia
do until (all_key_ bits_ found) {
    for i=1,40 {
        spusti PSO  (pocet_castic=80, iteracii=100)
        zapamätaj key_i, h(keyi)
    }
    h_average =  1/40 ∑ h(key_i))
    vyber key_i : h( key_i) ≤ h_average
    fixuj tie bity, ktoré sa vyskytujú vo viac než 80% prípadoch
end do
```

Obr. 4.6: Schéma algoritmu, ktorý sme použili na útok na DES

Použili sme účelovú funkciu definovanú vzťahom (4.1) s $n = 20$. Ako množinu M_i sme použili 20 náhodne vygenerovaných reťazcov a k nim prislúchajúcich šifrových textov. V algoritme PSO sme použili 80 častíc, a každý beh algoritmus sme zastavili po 100 iteráciách.

4.4.1 Priebeh algoritmu - príklad

Na ilustráciu správania algoritmu uveďme jeden príkladový priebeh pre dvojkolový DES. Nech

00100100 10110101 11010110 01100000 10100101 10100001 11101110 00100110

je hľadaný kľúč. Uvádzame len 56 bitov kľúča, zvyšných 8 bitov je paritných. Diskrétny PSO algoritmus sme spustili 40-krát a po každom behu sme si zapamätali kľúč s najlepšou hodnotou účelovej funkcie. Tak sme získali 40 adeptov na kľúč a vypočítali sme priemernú hodnotu z hodnôt účelových funkcií. Celkove 20 kľúčov malo lepšiu (to znamená nižšiu) hodnotu účelovej funkcie než bola priemerná hodnota, ktorá sa rovnala 22,701. Pomocou týchto 20 kľúčov sme zafixovali nasledujúcih 8 bitov (bity, ktoré mali rovnakú hodnotu aspoň pre 80% zostávajúcich kľúčov):

???0???? 1??????? ?1?????? ???????? ??1????? ???0???? ???????? 0?1??1??

V druhej iterácii sme opäť našli 40 adeptov, z ktorých 15 malo lepšiu hodnotu účelovej funkcie než priemer. Priemerná hodnota bola 20,205. Pomocou týchto 15 kľúčov sme zafixovali nasledujúcih 21 bitov:

00100??? 1??????? ?10?011? 011????? 1?10??0? 10100??? ??1????? 00100110

V tretej iterácii sme opäť našli 40 adeptov, z ktorých 23 malo lepšiu hodnotu účelovej funkcie než priemer. Priemerná hodnota bola 5,254. Pomocou týchto 23 kľúčov sme zafixovali nasledujúcih 24 bitov:

001001?? 101?010? 1101011? 0110000? 1010010? 1010000? 111?111? 0010011?

V priebehu štvrtej iterácie sa nám podarilo nájsť kľúč, ktorý mal nulovú hodnotu účelovej funkcie, a teda bol totožný s hľadaným kľúčom. Tu si treba uvedomiť, že osem bitov je paritných, a preto nevstupujú do šifrovania.

Kapitola 5

Diferenčná kryptoanalýza

Značenie. *V ďalsom texte budeme častokrát pracovať s dvojicami otvorených a šifových textov a s ich rozdielom. Pod rozdielom budeme vždy rozumieť XOR (exclusive or) hodnotu dvojice. Dvojice budeme vždy označovať ako* (T, T^*)*, ich rozdiel ako* T' *(miesto* T *môže byť ľubovoľné písmeno). Čiarkované premenné budú vždy označovať rozdiel, napríklad* T' *sa bude vždy vzťahovať k dvojici textov* T *a* T^**, takých, že* $T' = T \oplus T^*$

Diferenčná kryptoanalýza je spôsob útoku na šifrovací systém. Útočník si volí dvojice otvorených textov, ktoré majú konštantný rozdiel a musí poznať aj k nim prislúchajce šifrové texty aby vedel určiť rozdiel šifrových textov. Vyberá si tie otvorené dvojice, kde rozdiel šifrových textov s istou pravdepodobnosťou nadobúda definovanú konštantnú hodnotu. Ak pravdepodobnosť, že pri danom vstupnom rozdieli dostaneme určený výstupný rozdiel je oveľa vyššia, než by tomu bolo v prípade rovnomerného rozloženia výstupu, je možné pri dostatočnom počte dvojíc otvorených textov (a poznaní k nim prislúchajúcich šifrových textov) nájsť šifrovací kľúč. Základom k úspechu je nájsť také rozdiely, kde pravdepodobnosť, že vstupný rozdiel sa premietne po šifrovaní na výstupný rozdiel je čo najvyššia. Počet potrebných dvojíc je nepriamo úmerný pravdepodobnosti, že vstupný rozdiel otvorených textov sa premietne na určený výstupný rozdiel šifrových textov. Neskôr podrobnejšie popíšeme spôsob, ako takýto útok pracuje a ako hľadať vhodné vstupné a výstupné rozdiely a takzvané diferenčné charakteristiky, aby útok bol úspešný.

Vo verejne dostupnej literatúre túto techniku popísali Biham a Shamir v roku 1990 v práci [2], aj keď ju už v čase návrhu DES (1974) poznali vedci v IBM a NSA [6].

5.1 Ako sa správajú S-Boxy v prípade, že vstupy sú rozdiely

S-Box je kľúčovou časťou DES-u, a ako to vyplýva z popisu, je to jediná nelineárna časť algoritmu. V prípade, že vstupom do S-Boxu je náhodná premenná s rovnomerným rozdelením, aj výstup z S-Box bude bude mať rovnomerné rozdelenie. Vyplýva to z definície S-Boxu (každý riadok S-Boxu je permutáciou čísel $0-15$).

Pozrime, čo sa stane, ak ako vstup do S-Boxu budeme uvažovať rozdiel dvojice textov a nie jeden konkrétny text.

Definícia 5. Nech $1 \leq j \leq 8$. S_j budeme označovať $j-$ty S-Box a $S_j(B)$ výstup z S-Boxu pri vstupe B. Pre ľubovoľné $B_j^{'} \in (Z_2)^6$, $\Delta(B_j^{'}) = \{(B_j, B_j^*) \in (Z_2)^6 \times (Z_2)^6, B_j^{'} = B_j \oplus B_j^*\}$

Pre nejakú konkrétnu hodnotu $B_j^{'}$ množina $\Delta(B_j^{'})$ je tvorená takými dvojicami (B_j, B_j^*), ktorých rozdiel (XOR hodnota) je rovná $B_j^{'}$. Z definície okamžite vyplýva, že počet prvkov množiny $\Delta(B_j^{'})$ je $2^6 = 64$ a $B_j^* = B_j \oplus B_j^{'}$. Pre každý prvok (B_j, B_j^*) z množiny $\Delta(B_j^{'})$ vypočítajme $S_j(B_j) \oplus S_j(B_j^*)$, teda rozdiel výstupu dvojice z S-Boxu S_j. Poznamenajme, že $S_j(B) \in (Z_2)^4$. Čiže na vstupe do S-Boxu máme 64 rôznych rozdielov a na výstupe 16 možných rozdielov.

Definícia 6. Nech $1 \leq j \leq 8$, $B_j^{'}$ je bitový reťazec dĺžky 6 a $C_j^{'}$ je bitový reťazec dĺžky 4.

$$IN_j(B_j^{'}, C_j^{'}) = \{B_j \in (Z_2)^6 : S_j(B_j) \oplus S_j(B_j \oplus B_j^{'}) = C_j^{'}\}$$
$$N_j(B_j^{'}, C_j^{'}) = |IN_j(B_j^{'}, C_j^{'})|$$

Zoberme si jeden konkrétny vstupný rozdiel $B_j^{'}$ do S-Boxu S_j a jeden konkrétny výstupný rozdiel z toho istého S-Boxu $C_j^{'}$. Množina $IN_j(B_j^{'}, C_j^{'})$ je množina takých vstupov B_j, že ak zoberieme k nemu asociovanú hodnotu B_j^* takú, že ich rozdiel je $B_j^{'}$, tak rozdiel ich výstupov z S-Boxu bude $C_j^{'}$. $N_j(B_j^{'}, C_j^{'})$ označuje počet prvkov množiny $IN_j(B_j^{'}, C_j^{'})$, t.j. koľko existuje dvojíc so vstupným rozdielom $B_j^{'}$, takých, že výstupný rozdiel je $C_j^{'}$.

Teraz pre daný S-Box a pre daný vstupný rozdiel $B_j^{'}$ môžeme vytvoriť ditribučnú tabuľku, kde v prvom stĺpci budú hodnoty výstupných rozdielov $C_j^{'}$ (t.j. tabuľka bude mať 16 riadkov) a v druhom stĺpci budú

prvky množiny $IN_j(B_j', C_j')$ a v treťom stĺpci početnosť, koľko rôznych dvojíc s daným vstupným rozdielom sa zobrazí na daný výstupný rozdiel C_j'. Fixujeme teda S-Box a vstupný rozdiel. Máme 64 rôznych možností ako vygenerovať daný vstupný rozdiel, ktorý sa môže zobraziť na 16 rôznych výstupných rozdielov.

Príklad 7. *Vyberme si S-Box S_1, vstupný rozdiel $B_1' = 110100$ a vytvorme distribučnú tabuľku*

Výstupný rozdiel C_j'	$IN_1(110100, C_1')$	$N_1(110100, C_1')$
0000		0
0001	000011, 001111, 011110, 011111 101010, 101011, 110111, 111011	8
0010	000100, 000101, 001110, 010001 010010, 010100, 011010, 011011 100000, 100101, 010110, 101110 101111, 110000, 110001, 111010	16
0011	000001, 000010, 010101, 100001 110101, 100111	6
0100	010011, 100111	2
0101		0
0110		0
0111	000000, 001000, 001101, 010111 011000, 011101, 100011, 101001 101100, 110100, 111001, 111100	12
1000	001001, 001100, 011001, 101101 111000, 111101	6
1001		0
1010		0
1011		0
1100		0
1101	000110, 010000, 010110, 011100 100010, 100100, 101000, 110010	8
1110		0
1111	000111, 001010, 001011, 110011 111110, 111111	6

Tabuľka 5.1: tabuľka znázorňuje ako S-Box S_1 zobrazuje vstupný rozdiel 110100 na jednotlivé výstupné rozdiely.

Príklad ukazuje, že S-Box S_1 zobrazuje vstupný rozdiel 110100 veľmi nerovnomerne. Existuje 7 rôznych výstupných rozdielov, na ktoré sa nič nezobrazí, kým na výstupný rozdiel 0010 sa zobrazí až 16 vstupných dvojíc. Pre každý S-Box je jednoduché vytvoriť tabuľku početnosti so 64 riadkami a 16 stĺpcami, kde prvok $t_{i,j}$ reprezentuje početnosť, koľko dvojíc so vstupným rozdielom „i" (presne povedané so vstupným rozdielom, ktorého bitová reprezentácia zodpovedá číslu i), sa zobrazí na výstupný rozdiel „j" (na výstupný rozdiel, ktorého bitová reprezentácia zodpovedá číslu j). Všetky S-Boxy vykazujú takúto vlatnosť, že zobrazujú vstupné rozdiely veľmi nerovnomerne. Práve túto vlastnosť využíva diferenčná kryptoanalýza na nájdenia šifrového kľúča.

5.2 Ako nájsť kľúč, ak poznáme vstupný a výstupný rozdiel pre S-Box

Vráťme sa k popisu DES-u z časti (4.1). Samotný vstup B do S-Boxov v $i-$tom kole algoritmu sa vytvorí nasledovne

(a) 32−bitový vstupný reťazec R (R je výsledkom $(i-1)$-ého kola) sa rozšíri pomocou expanznej permutácie E na 48 bitov (16 bitov pôvodného reťazca sa zduplikuke a aplikuje sa definovaná permutácia). Pre jednoduchosť miesto $E(R)$ budeme písať len E, kde je z kontextu jasné, že sa jedná o výstup expanznej permutáce a nie o permutáciu samotnú.

(b) výsledný 48−bitový reťazec E z predchádzajúceho kroku sa XOR-uje s 48−bitovým podkľúčom J pre $i-$te kolo

Teda

$$B = E \oplus J \tag{5.1}$$

z čoho vieme vyjadriť podkľúč J v danem kole DES algoritmu

$$J = E \oplus B \tag{5.2}$$

Pozrime sa, čo sa stane, ak budeme ako vstup uvažovať rozdiel dvoch reťazcov

$$\begin{aligned} B^{'} &= B \oplus B^{*} \\ &= (E(R) \oplus J) \oplus (E(R^{*}) \oplus J) \\ &= E(R) \oplus E(R^{*}) \\ &= E(R \oplus R^{*}) \\ &= E(R^{'}) \\ &= E^{'} \end{aligned}$$

Všimnime si, že vstupný XOR nezávisí od bitov kľúča. Výstupný XOR však závisí - vyplýva to priamo z definície ako sa počíta výstupný XOR (necháme každú dvojicu prejsť algoritmom a následne urobíme rozdiel, čiže pri výpočte výsledného rozdielu sa kľúče nevyrušia).

B, E, J sú 48-bitové reťazce, ktoré môžeme napísať ako reťazec ôsmich $6-$bitových reťazcov, aby sme zvýraznili a zviditeľnili vstupy do príslušných S-Boxov.

$$\begin{aligned} B &= B_1B_2B_3B_4B_5B_6B_7B_8 \\ E &= E_1E_2E_3E_4E_5E_6E_7E_8 \\ J &= J_1J_2J_3J_4J_5J_6J_7J_8 \end{aligned}$$

Ak si teraz zoberieme pre $1 \leq j \leq 8$ jeden konkrétny vstupný rozdiel $E_j^{'}$ a jeden konkétny výstupný rozdiel $C_j^{'}$, je jasné, že

$$E_j \oplus J_j \quad \in \quad IN_j(E_j^{'}, C_j^{'}).$$

To znamená, že hľadaný kľúč sa musi nachádzať v množine ktorú vytvoríme tak, že každý prvok množiny $IN_j(E_j^{'}, C_j^{'})$ XOR-ujeme s E_j.

Definícia 8. Nech $1 \leq j \leq 8$ a E_j, E_j^{*} sú bitové reťazce dĺžky 6 a $C_j^{'}$ reťazec dĺžky 4. Potom definujeme množinu $Key_j(E_j, E_j^{'}, C_j^{'}) = \{B_j \oplus E_j : B_j \in IN_j(E_j^{'}, C_j^{'})\}$

Veta 9. *Nech E_j a E_j^{*} sú vstupy do S-Boxu S_j, $E_j^{'} = E_j \oplus E_j^{*}$ a nech ich výstupný rozdiel z S-Boxu je $C_j^{'}$. Potom kľúč J_j sa musí nachádzať v množine $Key_j(E_j, E_j^{*}, C_j^{'})$.*

Všimnime si, že množina $Key_j(E_j, E_j^{'}, C_j^{'})$ bude mať rovnaký počet prvkov ako množina $IN_j(E_j^{'}, C_j^{'})$. Aby sme jednoznačne určili kľúč, potrebujeme vygenerovať množinu Key_j s rôznymi trojicami $E_j, E_j^{'}, C_j^{'}$.

Pre nájdenie správneho kľúča sa pre každy S-Box implementuje množina 64 počitadiel. Každý podkľúč má 6 bitov, čo predstavuje 64 rôznych možností. Pre S-Box S_j nech $count_j^i$ je počítadlo priradené podkľúču s bitovou reprezentáciou i, pričom hodnota počítadla sa zvýši o 1 vždy, keď pre nejakú trojicu $E_j, E_j^{'}, C_j^{'}$ podkľúč s bitovou reprezentáciou ï"sa nachádza v množine Key_j. Keď teda budeme mať k rôznych trojíc, a pre každý S-Box budeme mať práve 1 počítadlo $count_j^{i_j}$ s hodnotou k, t.j pre každé $1 \leq j \leq 8$ $\exists!$ i_j : $count_j^{i_j} = k$, tak sme našli všetky podľúče $J_1, J_2, J_3, J_4, J_5, J_6, J_7, J_8$. Bitová reprezentácia J_j sa rovná bitovej reprezentácii i_j (t.j. indexu počítadla pre S-Box S_j, ktorého hodnota sa rovná k). Ako ukážeme neskôr, tento prístup je veľmi jednoduchý, ale pri veľkom počte možných kľúčov potrebujeme veľkú pamäť pre počítadlá. Preto sa vyvinuli iné metódy, ktoré popíšeme neskôr.

Týmto spôsobom teda nájdeme 48 bitov hľadaného kľúča, zvyšných 8 bitov sa dá nájsť prehľadaním zvyšných 256 možností.

5.3 Diferenčná charakteristika

Útok na DES je založený na vhodnom výbere vstupného rozdielu, ktorý DES s istou pravdepodobnosťou premietne na výstupný rozdiel. Zavedieme dôležitý pojem diferenčnej charakteristiky.

Definícia 10. Nech $n \geq 1$. Pod $n-$kolovou diferenčnou charakteristikou budeme rozumieť zoznam nasledujúceho tvaru

$$L_0', R_0', L_1', R_1', p_1, \ldots, L_n', R_n', p_n,$$

spĺňajúci nasledovné podmienky:

(a) $L_i' = R_{i-1}'$, pre všetky $1 \leq i \leq n$.

(b) Nech $L_{i-1}R_{i-1}$, $L_{i-1}^*R_{i-1}^*$ sú zvolené tak, že $L_{i-1} \oplus L_{i-1}^* = L_{i-1}'$ a $R_{i-1} \oplus R_{i-1}^* = R_{i-1}'$. Predpokladajme, že L_iR_i a $L_i^*R_i^*$ sú výsledky aplikácie $i-$teho kola DES. Potom pravdepodobnosť, že $L_i \oplus L_i^* = L_i'$ a $R_i \oplus R_i^* = R_i'$ je práve p_i. (Pravdepodobnosť sa počíta so zreteľom na všetky možné kľúče v danom kole DES). Pravdepodobnosť celej charakteristiky definujeme ako súčin jednotlivých pravdepodobností $p = \prod_{i=1}^{n} p_i$.

Poznámka. *Skutočná pravdepodobnosť, že* $L_i \oplus L_i^* = L_i'$ *a* $R_i \oplus R_i^* = R_i'$ *pre* $1 \leq i \leq n$ *nie je presne* $\prod_{i=1}^{n} p_i$. *To by bola pravda v prípade, že podkľúče v jednotlivých kolách DES-u boli nezávislé a náhodne určené. V literatúre [??] sa súčin jednotlivých pravdepodobností považuje za dosť dobrý odhad ku skutočnej pravdepodobnosti. Pravdepodobnosť charakteristiky ako súčinu pravdepodobností je zavedená definitoricky.*

Pre útok je potrebné nájsť diferenčnú charakteristiku s čo najvyššou pravdepodobnosťou. Táto pravdepodobnosť pre DES je rádovo 10^{-48} a budeme potrebovať veľké množstvo vstupných dvojíc, aby sme našli takú, ktorá spĺňa diferenčnú charakteristiku.

Definícia 11. Predpokladajme $L_0 \oplus L_0^* = L_0'$ a $R_0 \oplus R_0^* = R_0'$. Hovoríme, že dvojica otvorených textov L_0R_0 a $L_0^*R_0^*$ tvoria správnu dvojicu vzľadom na danú $n-$kolovú charakteristiku, ak $L_i \oplus L_i^* = L_i'$ a $R_i \oplus R_i^* = R_i'$ pre $1 \leq i \leq n$. Inak dvojicu nazývame nesprávnou dvojicou.

Aby útok bol efektívny, musíme vedieť odfiltrovať nesprávne dvojice. Pôvodne sa pre každý potenciálny kľúč alokovalo jedno počítadlo, ktorého hodnota sa zvýšila o 1, ak počas útoku niektorá vstupná dvojica indikovala daný kľúč. Keďže správny kľúč sa počas útoku vyskytuje s oveľa vyššou pravdepodobnosťou než nesprávny kľúč, dá sa po dostatočnom počte vstupných dvojíc rozlíšiť správny kľúč. Táto metóda je veľmi jednoduchá, ale má nevýhodu v tom, že má veľké pamäťové nároky. Potrebujeme alokovať a udržiavať rádove až do 2^{48} počítadiel, čo je veľmi nepraktické.

Biham a Shamir v [3] vyvinuli metódu bez pamäťových nárokov. Počas útoku sa filtruje, či ide a správnu dvojicu alebo nie. Z každej správnej dvojice je možné odvodiť množinu možných kľúčov medzi ktorými sa nachádza aj hľadaný kľúč (kľúče odvodené z nesprávnej dvojice sú náhodne rozložené). Pre každý navrhovaný kľúč sa dá previesť okamžite test, že sa zašifruje otvorený text s daným kľúčom a výsledok sa porovná so šifrovým textom vytvoreným hľadaným kľúčom. Tento test sa dá previesť paralelne (napríklad na iných počítačoch).

Základom filtrovacieho procesu je nasledovné. Keď sa pozrieme na distribučné tabuľky pre jednotlivé S-Boxy, existujú kombinácie vstupných a výstupných rozdielov, ktoré nikdy nemôžu nastať (v tabuľke 5.3 sú uvedené pravdepodobnosti, že náhodne zvolený vstupný rozdiel a výstupný rozdiel je pre daný S-Box reálny. Pod tým rozumieme to, že existuje možnosť, že S-Box zobrazí daný vstupný rozdiel na daný výstupný rozdiel. Ak sa pozrieme na tabuľku 7, vidíme napríklad, že $(110100) \rightarrow (0001)$

je možné, ale (110100) $\rightarrow$ (1001) možné nie je. Približne 20% kombinácií nie je možných pre jednotlivé S-Boxy. To znamená, že ak všetky S-Box sú aktívne môžeme vylúčiť približne $1 - 0.8^8 = 83.2\%$ dvojíc. V útoku na 16-kolový DES sa z vedomosti o vstupoch a výstupoch pre 1., 15. a 16. kolo podarilo vylúčiť až 92.5% dvojíc. Pravdepodobnosť, že po filtrovacom procese dvojica bude správnou dvojicou bol 58%.

S-Box	Pravdepodobnosť možnej kombinácie vstupu a výstupu
S_1	79.4%
S_2	78.6%
S_3	79.6%
S_4	68.5%
S_5	76.5%
S_6	80.4%
S_7	77.2%
S_8	77.1%

Tabuľka 5.2: pravdepodobnosť, že daná kombinácia vstupného a výstupného rozdielu pre daný S-Box je reálna

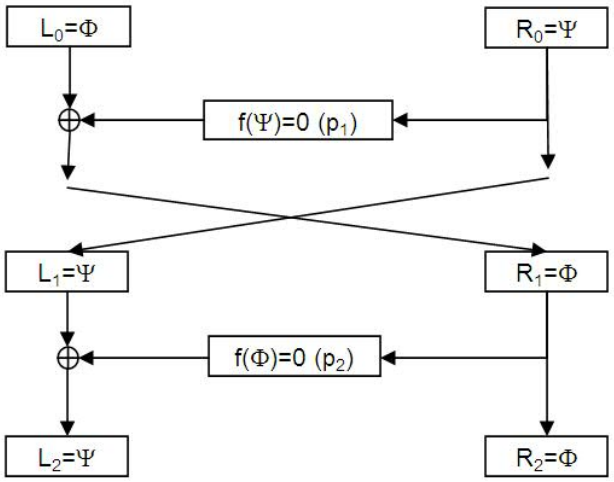
Často sa pre útoky používajú iteratívne charakteristiky, napríklad 2−kolová charakteristika sa iteratívne aplikuje niekoľkokrát za sebou, čím sa vytvorí $2n$-kolová charakteristika, kde n je počet iteráci. Obecná 2-kolová iteratívna charakteristika sa dá zapísať nasledovne

$$\begin{aligned} (L_0^{'}, R_0^{'}) &= (\Phi\ , \Psi\ ) \\ (L_1^{'}, R_1^{'}) &= (\Psi\ , \Phi\ ) \quad p_1 = p \\ (L_2^{'}, R_2^{'}) &= (\Psi\ , \Phi\ ) \quad p_2 = q \end{aligned} \tag{5.3}$$

Na obrázku 5.1 je grafické znázornenie, pre ľahšie pochopenie.

V literatúre [15] sa používa zjednodušené značenie, kde sa vyznačuje len ako Feistelovská funkcia f transformuje vstupný rozdiel v jednotlivých kolách. Vzťah $L_i^{'} = R_{i-1}^{'}$ je triviálny a nie je nutné ho uvádzať. V jednodušenom zápise by charakteristika (5.3) vyzerala nasledovne

$$\begin{aligned} Input &: (\Phi, \Psi) \\ 0 &\leftarrow \Psi \qquad p_1 = p \\ 0 &\leftarrow \Phi \qquad p_2 = q \\ Output &: (\Psi, \Phi) \end{aligned} \tag{5.4}$$



Obr. 5.1: Grafické znázornenie 2-kolovej diferenčnej charakteristiky

Do funkcie f je vstupom vždy pravá polovica reťazca, ktorú píšeme v pravej časti vzťahu. Aby sme to zachovali, šipky, ktoré naznačujú smer transformácie teda smerujú sprava doľava. Ako vidíme 2-kolová iteratívna charakteristika je jednoznačne určenná vstupom (Φ, Ψ) a celkovou pravdepodobnosťou. Preto budeme v prípadoch, keď je to z kontextu jednoznačné, odvolávať sa na 2-kolovú charakteristiku len vstupom (Φ, Ψ).

5.4 Hľadanie charakteristiky s najvyššou pravdepodobnosťou

V roku 1992 Biham a Shamir [3] publikovali úspešný diferenčný kryptoanalytický útok na 16-kolový DES menšou zložitosťou než prehľadávanie hrubou silou. Na útok potrebovali 2^{47} zvolených otvorených aj šifrových textov. Shamir a Biham použili 2 charakteristiky obe iteratívne dvojkolové, ktoré kombinovali medzi sebou v 2. až 14.-tom kole. Prvé kolo a posledné 2 kolá boli špeciálne ošetrené, aby nedošlo k horším výsledkom než by bolo priame vyhľadávanie hrubou silou. Pri sedemnásobnej iterácii 2-kolovej charakteristiky by tomu už tak bolo, preto sa nedá použiť

takýto priamočiary prístup. Nižšie sú charakteristiky použité v [3].

$$\begin{array}{rcll} Input & : & (19600000_x, 0) & \\ 0 & \leftarrow & 0 & p_1 = 1 \\ 0 & \leftarrow & 19600000_x & p_2 = \dfrac{1}{247} \\ Output & : & (0, 19600000_x) & \end{array} \tag{5.5}$$

a

$$\begin{array}{rcll} Input & : & (1B600000_x, 0) & \\ 0 & \leftarrow & 0 & p_1 = 1 \\ 0 & \leftarrow & 1B600000_x & p_2 = \dfrac{1}{247} \\ Output & : & (0, 1B600000_x) & \end{array} \tag{5.6}$$

Prirodzene vznikla otázka, či navrhnuté charakteristiky sú najlepšie možné. V roku 1993 Knudsen v článku [15] vyjadril silné presvedčenie, že najlepšia charakteristika je 2-kolová iteratívna charakteristika a iná než (19600000_x) a $(1b600000_x)$, ktoré použili Biham a Shamir v článku [3]. Knudsen uviedol 11 najlepších 2-kolových iteratívnych charakteristík s priemernými pravdepodobnosťami - my ich uvádzame v tabuľke 5.3. Označme

(a) $\Psi = (19600000_x$, jedna z charakteristík z [3] s priemernou pravdepodobnosťou 1/234

(b) $\Gamma = (1b600000_x)$ druhá z charakteristík z [3] s priemernou pravdepodobnosťou 1/234

(c) $\Theta = (00196000_x)$, Knudsenova charakteristika s pravdepodobnosťou 1/256

Pre Ψ a Γ existujú 2 rôzne pravdepodobnosti, v závislosti od kľúča. Jedna pravdepodobnosť je 1/146 a druhá 1/585 a ich priemerná hodnota je 1/234. V článku [7] Courtois urobil podrobnú analýzu a porovnanie charakteristiky Θ voči kombinovanému použitiu Ψ a Γ s nasledovnými výsledkami

- v prípade fixného kľúča najlepšou diferenčnou charakteristickou pre 16-kolový DES je Knudsenova charakteristika Θ. Pre Ψ a Γ existujú 2 rôzne pravdepodobnosti v závislosti od kľúča. Keď počítame celkovú pravdepodobnosť iteratívnej charakteristiky, pravdepodobnosti sa násobia a preto geometrický priemer je správnym

Charakteristika Φ	Pravdepodobnosť $1/n$	Priemerná prevdepodobnosť $1/n$
19600000_x	$146, 585$	234
$1b600000_x$	$585, 146$	234
00196000_x	256	256
$000003d4_x$	$210, 390$	273
$4000001d_x$	$205, 1024$	341
19400000_x	$0, 195$	390
$1b400000_x$	$195, 0$	390
40000019_x	$248, 390, 744, 1170$	455
$4000001f_x$	$248, 390, 744, 1170$	455
$1d600000_x$	$205, 512, 819, 2048$	468

Tabuľka 5.3: Pravdepodobnosti iteratívnych 2-kolových charakteristík pre DES. V prípade, že v druhom stĺpci je viac ako jedno číslo, znamená to, že v závislosti od hodnoty určitých bitov v kľúči sa pravdepodobnosť mení. Priemerná ppravdepodobnosť je uvedená v treťom stĺpci

vyjadrením priemernej pravdepodobnosti a nie aritmetixký priemer. Geometrický priemer pravdepodobností pre Ψ a Γ je $1/292$, čo je horšie než pravdepodobnosť pre Θ.

- Útok kombináciou 2 charakteristík Ψ a Γ popísaný v [3] je najlepším útokom v priemernom prípade nad všetkými kľúčmi, lebo môžeme využiť 2 charakteristiky súčasne s podobným šírením pravdepodobnosti. Útok vyžaduje $2^{48.34}$ otvorených textov a nie 2^{47}, ako bolo v pôvodnom článku.
- V prípade, že kľúč sa mení počas útoku, tak platí pôvodný odhad 2^{47}, čiže je ľahšie zlomiť DES s meniacim sa kľúčom než s fixným kľúčom, čo je dosť prekvapivý výsledok.

V [15] je ukázané, že žiadna 2-kolová alebo 3-kolová iteratívna charakteristika pre DES nemôže byť lepšia než 2-kolová iteratívna charakteristika. V článku (v časti 3.5) je hypotéze, že nie je možné nájsť n-kolovú iteratívnu charakteristiku s $n > 4$, ktorá by mala vyššiu pravdepodobnosť než najlepšia 2-kolová aplikovaná $\frac{n}{2}$ krát za sebou.

V roku 1994 Matsui v článku [17] publikoval praktický algoritmus pre odvodenie najlepšej diferenčnej charakteristiky pre DES. Stručný popis algoritmu uvádzame v prílohe. Pomocou uvedeného algoritmu sa poda-

rilo ukázať, že pre n-kolový DES, kde $7 \leq n \leq 16$, najlepšia charakteristika je 2-kolová iteratívna, ako to bolo predpokladané Knudsenom. Pre 5-kolový DES algoritmus našiel lepšiu charakteristiku, než Biham a Shamir v pôvodnom článku [3]. V tabuľke 5.4 uvádzame pravdepodobnosti najlepších charakteristík, ktoré našiel algoritmus pre n-kolový DES, $4 \leq n \leq 16$.

Počet kôl DES-u	Pravdepodobnosť najlepšej charakteristiky
4	1.31×2^{-10}
5	1.72×2^{-14}
6	1.03×2^{-20}
7	1.31×2^{-24}
8	1.43×2^{-31}
9	1.43×2^{-32}
10	1.57×2^{-39}
11	1.57×2^{-40}
12	1.71×2^{-47}
13	1.71×2^{-48}
14	1.87×2^{-55}
15	1.87×2^{-56}
16	1.02×2^{-62}

Tabuľka 5.4: V tabuľke sú znázornené pravdepodobnosti pre najlepšie charakteristiky, ktoré našiel program z [17] pre rôzny počet kôl DES-u

Kapitola 6

Použiitie PSO na nájdenie diferenčnej charakteristiky

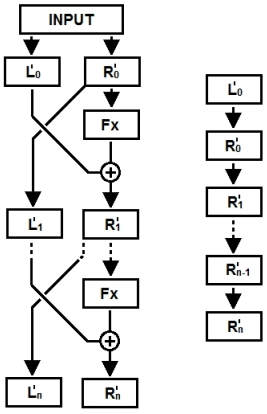
V tejto kapitole popíšeme akým spôsobom sa dá aplikovať PSO na nájdenie diferenčnej charakteristiky pre DES. Táto úloha je dosť netriviálna vzhľadom na komplexnosť pojmu diferenčnej charakteristiky a vyžadoval si zásahy aj do algoritmu PSO, najmä na výpočet zmeny polohy. Algoritmus PSO pracuje s niektorými základnými objektami, ktoré si musíme pre modifikáciu pre nájdenie diferenčnej charakteristiky ujasniť a presne definovať.

Priestor riešení

V prvom rade si musíme ujasniť, aké objekty predstavujú naše riešenia. Keďže hľadáme diferenčnú charakteristiku (viď definícia 10) pre n kôl, každé riešenie bude predstavovať jednu konkrétnu charakteristiku. Zaviedli sme jednoduchšie značenie pre charakteristiky, aby sme mohli s nimi lepšie pracovať. Keďže pre všetky $1 \leq i \leq n$ platí $L'_i = R'_{i-1}$, môžeme zo zápisu charakteristiky vynechať L'_i a písať ju v tvare $L'_0, R'_0, R'_1, p_1 \ldots, R'_n, p_n$. Charakteristiky budeme značiť veľkými gréckymi písmenami, napríklad $\Lambda = L'_0, R'_0, R'_1, \ldots, R'_n$. Každé riešenie bude teda reprezentované $32 \times (n+2)$ bitmi, kde n predstavuje počet kôl DES-u. Viď obrázok 6.1. V definícii diferenčnej charakteristiky vystupujú aj pravdepodobnosti prechodov z jedného rozdielu na druhý počas kola, tieto budeme v zápise vynechávať. Celkovú pravdepodobnosť charakteristiky budeme počítať v samotnom algoritme.

Všinminme si jednu dôležitú vec, že nie každý $32 \times (n+2)$ bitový reťazec

reprezentuje reálnu diferenčnú charakteristiku, nakoľko sa môže stať, že celková pravdepodobnosť takejto charakteristiky je rovné nule. Takéto reťazce nebudeme považovať za riešenia. Náš priestor budú síce všetky reťazce danej dĺžky, nakoľko až po preskúmaní a analýze konkrétneho reťazca vieme posúdiť, či reprezentuje reálne riešenie. Skutočnosť, že nie každý reťazec je riešenie, spôsobil, že počas inicializácie aj zmene polohy sme nemohli len priamočiaro vykonať zmenu, ale museli sme zaviesť kroky, ktoré zabezpečili alebo minimalizovali riziko, že nový reťazec nebude predstavovať reálne riešenie.

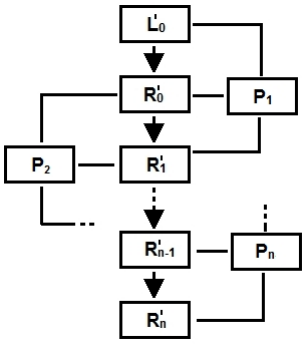


Obr. 6.1: Schéma nášho zápisu diferenčnej cesty

Inicializácia častíc

V ďalšom kroku musíme vybrať počet častíc pre PSO a definovať akým spôsobom ich budeme inicializovať. Ako sme poísali v predchádzajúcom odseku, nemôžeme len náhodne generovať jednotlivé bity, lebo by sme mohli dostať diferenčnú charakteristiku s nulovou pravdepodobnosťou. Preto už počas inicializácie sme museli definovaž proces, ktorý by vygeneroval charakteristiku s nenulovou pravdepodobnosťou.

Na polohu častice sa môžeme dívať ako na $n+2$ úsekov, každý dlhý 32 bitov. Každá trojica týchto úsekov R'_j, R'_{j+1}, R'_{j+2} sa dá prepísať ako $L'_{j+1}, R_{j+1}, L'_{j+2}, R'_{j+2}$ a teda k nej môžeme dopočítať pravdepodobnosť p_{j+2} (viď definíciu 10). Toto je znázornené na obrázku 6.2.



Obr. 6.2: Ku každej trojicii blokov vieme priradiť pravdepodobnosť

Polohu častíc sme inicializovali nasledovne. Zvolili sme pseudonáhodne celé číslo $-1 \leq k \leq n-1$. Úseky R'_k, R'_{k+1}(kde $R'_{-1} = L'_0$) sme bit po bite náhodne vygenerovali.

Následne sme „dopočítali" úseky R_j pre $j = k+2, \ldots, n$ (okrem prípadu $k = n-1$). Najprv sme aplikovali expanzívnu funkcu E na úsek R'_{k+1}, dostávame $E(R'_{k+1})$. Tento reťazec je po rozdelení na osem 6-bitových častí vstupom do jednotlivých S-boxov. Pripomeňme, že pre každý S-box a pre každý vstupný rozdiel máme distribučnú tabuľku výstupných rozdielov. Teda za použitia prislúchajúcej tabuľky vieme pre rozdiely vstupov určiť pravdepodobnosti rozdielov výstupov. Takže rozdiel výstupov zvolíme z tejto tabuľky proporčne podľa pravdepodobnosti jeho výskytu. Zreťazením výstupov jednotlivých S-boxov získame reťazec Z. Po aplikovaní permutácie P na tento reťazec dostávame $P(Z) = R'_k \oplus R'_{k+2}$. Na základe tejto rovnice vieme vyjadriť R'_{k+2}.

Ukázali sme, akým spôsobom inicializujeme úseky R_j pre $j = k+2, \ldots, n$. Úseky R_j pre $j = -1, \ldots, k-1$ dopočítavame rovnakým postupom, keďže ak vymeníme R'_k, R'_{k+1} za R'_{k+1}, R'_k vieme rovnakým postupom dopočítať R'_{k-1}.

Ako populačnú topológiu sme zvoli topológiu „global best". Pripomíname, že populačná topológia definuje pre každú časticu jej susedov, čo sa v algoritme PSO využíva pri stanoveni polohy lb. V topológii „global best" všetky častice susedia so všetkými.

Definícia účelovej funkcie

Používali sme nasledujúcu účelovú funkciu.

$$f(\Lambda) = \begin{cases} -\log p & \text{ak } p \neq 0 \text{ (p je prevdepodobnosť charakteristiky } \Lambda) \\ 1000 & \text{inak} \end{cases}$$

Poznámka. *Pod* log *budeme rozumieť desiatkový logaritmus.* $p = \prod_{i=1}^{n} p_i$, *ako v definícii 10.*

Zmena polohy

Pri zmene polohy nepoužívame rýchlosť. Algoritmus identifikuje SBox (náhodne, ale s prihliadnutím na „váhu" každého SBox-u, presnejšie vysvetlíme nižšie), ktorý najviac znižuje celkovú pravdepodobnosť diferenčnej charakteristiky. Následne zmeníme tie bity v charakteristike, ktoré vstupujú alebo vystupujú z identifikovaného SBox-u (k takejto modifikácii PSO algoritmu nás inšpirovala tzv bare-bone verzia PSO, kde sa tiež nepoužíva rýchlosť na zmenu polohy, ale sa využíva štatistické rozdelenie vzdialeností častíc od personálnej najlepšej polohy alebo najlepšej polohy susedov [11]). Nižšie presnejšie popíšeme postup pre zmenu polohy častíc.

Nech n je dĺžka diferenčnej charakteristiky a $p_i,\ i = 1, \ldots, n$ predstavuje pravdepodobnosti z definície 10. Ku každej pravdepodobnosti p_i sme priradili novú pravdepodobnosť

$$p_i^* = \frac{1}{min_{j=1,\ldots,8}\Pr(SBox_j)}, \tag{6.1}$$

kde $Pr(SBox_j)$ je pravdepodobnosť, ktorou j-ty S-Box prispieva k pravdepodobnosti p_i. Vieme totiž, že $p_i = \prod_{j=1}^{8} \Pr(SBox_j)$.

- Vyberieme jedno z p_i^* podľa nasledovného kľúča. Interval $[0, \sum_{i=1}^{n} p_i^*]$ rozdelime na n časti, kde i-ta časť bude mať dĺžku p_i^*. Zvolíme si pseudonáhodné číslo z intervalu $[0, \sum_{i=1}^{n} p_i^*]$ a zistíme do ktorého inetrvalu padlo. Ak padlo do i-tého intervalu, vyberieme si p_i^*. Takýto výber bude náhodný a súčasne aj proporčný jednotlivým hodnotám p_i^*.
- Pre vybrané p_i^* obdobným spôsobom vyberieme jeden z SBox-ov, kde váha SBox-u pre výber je určená hodnotou $\frac{1}{\Pr(SBox_j)}$.

- Identifikujeme bity, ktoré majú vplyv na vybraný SBox v danom kole a priradíme im hodnotu 0 s pravdepodobnosťou $P = (0.2 \times rnd())^2$ a s pravdepodobnosťou $1 - Q$ im dáme hodnotu 1, kde $rnd()$ je pseudonáhodné číslo z intervalu $[0, 1)$ a je rovnaké pre všetky bity jednej častice (tento vzťah sme určili empiricky podľa správania sa algoritmu).

Mutácie

Aby sme jednak zabránili uviaznutiu algoritmu v lokálnom minime, ale na druhej strane posilnili prehľadávanie okolia lokálneho minima zaviedli sme nasledovné 2 zmeny do algoritmu PSO (nazvali sme ich mutácie).

(a) Na začiatku každej iterácie sme náhodne vybrali jednu časticu, ktorej sme zmenili aktuálnu polohu. Novú sme jej vygenerovali metódou, ako keď inicializujeme polohy častíc na začiatku behu algoritmu s tým rozdielom, že po náhodnom zvolení hodnôt $k, k + 1$, sme nechali pôvodné hodnoty reťazcov R'_k, R'_{k+1}(namiesto ich náhodného generovania).

(b) Hneď po prevedení predchádzajúcej mutácie sme znova náhodne vybrali jednu časticu (mohla to byť aj tá istá častica, ako v predchádzajúcom kroku) a tú sme „presunuli" na pozíciu globálneho maxima.

Voľba parametrov

Pracovali sme s 200 časticami a 500 iteráciami. Pre každú dĺžku diferenčnej charakteristiky od 1 do 5 sme spustili algoritmus 20-krát.

Prehľad výsledkov

V tabuľke 6.3 sú príklady najlepších diferenčných charakteristík pre jednotlivé kolá, ktoré sme našli. V tabuľke 6.1 sú uvedené pravdepodobnosti nasledujúcich charakteristík: pravdepodobnosť najlepšej existujúcej diferenčnej charakteristiky, pravdepodobnosť najlepšej charakteristiky, ktorú sme my našli a priemerná pravdepodobnosť našich nájdených charakteristík (v 20 behoch algoritmu).

Dĺžka cesty	Najlepšia pst	Naša najlepšia pst	Naša priemerná pst
1	1	1	1
2	1/4	1/4	1/4
3	1/16	1/16	1/16
4	$10^{-2.89}$	$10^{-3.64}$	$10^{-3.97}$
5	$10^{-3.98}$	$10^{-7.09}$	$10^{-8.31}$

Tabuľka 6.1: Psti ciest

Dĺžka cesty	Počet DCH s nenulovou pravdepodobnosťou v %
1	12,960
2	1,689
3	0,208
4	0,026

Tabuľka 6.2: Percentuálne vyjadrenie počtu diferenčných charakteristík (DCH) s nenulovou pravdepodobnosťou

Dĺžka cesty	Príklady
1	A004 0080 0000 0000 A004 0080
	0002 8000 0000 0000 0002 8000
	8002 1840 0000 0000 8002 1840
2	0000 0400 0000 0000 0000 0400 0020 0008
	0020 0008 0000 0400 0000 0000 0000 0400
	4008 0000 0400 0000 0000 0000 0400 0000
3	0020 0008 0000 0400 0000 0000 0000 0400 0020 0008
	4008 0000 0400 0000 0000 0000 0400 0000 4008 0000
4	0040 0040 0004 0000 0000 0000 0004 0000 0040 0040 0614 0504
	2501 0580 0020 2000 0000 0400 0000 0000 0000 0400 0020 0008
5	00C0 8040 2004 0000 0000 1200 0000 0000 0000 1200 2004 0000 00C0 8040
	1010 A40B 2000 0080 0000 2000 0000 0000 0000 2000 2000 0080 1210 A40B

Tabuľka 6.3: Príklady najlepších nájdených diferenčných charakteristík pre rôzne počty kôl

Kapitola 7

Záver

V práci sme úspešne aplikovali modifikovanú verziu algoritmu PSO na riešenie problému jednoduchej zámeny. Pre takýto typ problémov, PSO funguje efektívne a spoľahlivo. Navrhli sme postup ako aplikovať PSO na útok na DES. So znalosťou len 20 ľubovoľných otvorených textov a k nim prislúchajúcich šifrových textov sme dokázali zlomiť 2-kolový DES a podali sme analýzu a vysvetlenie, prečo takouto metódou nie je možné zlomiť viac než 4-kolový DES. Na dosiahnutie lepších výsledkov je nutné do algoritmu zabudovať hlbšie vedomosti o samotnom šifrovaní. V závere sme navrhli špecifickú variantu PSO bez použitia rýchlosti pri zmene polohy častíc na hľadanie optimálnej diferenčnej charakteristiky pre DES. Pre dvoj až päťkolové charakteristiky sme uviedli porovnanie nájdených výsledkov so známymi najlepšími charakteristikami.

Dodatok A

Popis algoritmu na nájdenie najlepšej charakteristiky

Program pracuje na princípe indukcie, na základe poznania najlepšej charakteristiky pre $n-1$ kôl, program vypočíta najlepšiu charakteristiku pre n-té kolo. Pretože pre prvé 3 kolá je jednoduché vypočítať najlepšiu charakteristiku, algoritmus je postavený pre $4 \leq n \leq 16$. Zavedieme označenia, ktoré budeme používať v popise algoritmu-

B_n	označuje najlepšiu diferenčnú charakteristiku pre n-kôl
$\bar{B_n}$	označuje počiatočnú hodnotu B_n
$X^{'}$	označuje vstupný rozdiel F
$Y^{'}$	označuje výstupný rozdiel z funkcie f v DES algoritme.
$(X^{'}, Y^{'})$	$\Pr\{f(X) \oplus f(X^{*}) = Y^{'}; X \oplus X^{*} = X^{'}\}$
$[p_1, p_2, \ldots, p_n]$	$\prod_{i=1}^{n} p_i$
B_n	$\max_{X_i^{'}=X_{i-2}^{'} \oplus Y_{i-1}^{'}, (3 \leq i \leq n)} \left[(X_1^{'}, Y_1^{'}), (X_2^{'}, Y_2^{'}), \ldots (X_n^{'}, Y_n^{'})\right]$

Procedure Round-1 :
Begin Program
For all $X_1^{'}$, do

- let $p_1 = \max_{Y^{'}}(X_1^{'}, Y^{'})$
- $if([p_1, B_{n-1}] \geq \bar{B_n})$ then call Procedure Round-2

Exit the Program .

Procedure Round-2 :

For all $X_2^{'}$,$Y_2^{'}$ do

- let $p_2 = (X_2^{'}, Y_2^{'})$
- $if([p_1, p_2, B_{n-2}] \geq \bar{B_n})$ then call Procedure Round-3

Return to upper Procedure .

Procedure Round-i $(3 \leq i \leq (n-1))$:

For all $Y_i^{'}$, do

- let $X_i^{'} = X_{i-2}^{'} \oplus Y_{i-1}^{'}, \quad p_i = (X_i^{'}, Y_i^{'})$
- $if([p_1, p_2, \ldots, p_i, B_{n-i}] \geq \bar{B_n})$ then call Procedure Round-$(i+1)$

Return to upper Procedure .

Procedure Round-n :

let $X_n^{'} = X_{n-2}^{'} \oplus Y_{n-1}^{'}, \quad p_n = \max_{Y'}(X_n^{'}, Y^{'})$

$if([p_1, p_2, \ldots, p_n] \geq \bar{B_n})$ then $\bar{B_n} = [p_1, p_2, \ldots, p_n]$

Return to upper Procedure .

Dodatok B

Pseudokód - Jednoduchá zámena

```
personal_koeficient = 1.49445  // koeficient kognitivneho zrychlenia
local_koeficient = 1.49445  // koeficient socialneho zrychlenia

FUNCTION PSO()
{
    FOR vsetky castice DO
    {
        pozicia = nahodna permutacia
        // castice mame v cyklickom poli
        sused1 = castica nalavo v poli od nej
        sused2 = castica napravo v poli od neh
        zotrvacnost = nahodna hodnova od 0.5 do 1
        rychlost = nulovy vektor // pole rovnako dlhe ako samotna permutacia
    }

    FOR FROM 1 TO pocet kol pso DO
    {
        zavolaj funkciu kolo_PSO()
    }

    return local best permutacia (pozicia) castice, s najlepsim ohodnotenim svojej local best pozicie
}

FUNCTION kolo_PSO()
{
    FOR vsetky castice DO
    {
        IF local_best > ohodnotenie aktualnej pozicie THEN
            local_best = ohodnotenie aktualnej pozicie

        IF personal_best > ohodnotenie aktualnej pozicie THEN
            personal_best = ohodnotenie aktualnej pozicie
    }

    FOR vsetky castice DO
    {
        IF local_best > sused1.personal_best THEN local_best = sused1.personal_best
        IF local_best > sused2.personal_best THEN local_best = sused2.personal_best
    }

    FOR vsetky castice DO
    {
        // tu sme pouzivali 4 rozne sposoby ako vypocitavat rychlost castice A, B, C, D
        zavolaj fukciu aktualizacia_rychlosti()  // _A, _B, _C alebo _D

        max_rychlost = maximalna hodnota zlozky rychlosti tejto castice zo vsetkych zloziek rychlosti

        IF permutacia pozicie = permutacia pozicie local best THEN
        {
            // nahodna mutacia v pripade ze sa castica ustalila
            vymen nahodne 2 zlozky permutacie aktualnej pozicie
        }
```

```
        ELSE
        {
            FOR vsetky zlozky rychlosti castice AS i DO
            {
                nahodne_cislo = nahodne cislo z [0, 1)

                IF nahodne_cislo * max_rychlost < rychlost[i] THEN
                {
                    // vymenou 2 hodnot v permutacii sa "priblizime" k permutacii local best
                    na i-te miesto v permutacii (aktualnej pozicii castice) nastav hodnostu,
                    ktore je na i-tom mieste v permutacii local best

                    na miesto kde bola povodne prave dosadena hodnota dosadime
                    povodnu hodnotu i-teho miesta permutacie (aktualnej pozicie)
                }
            }
        }
    }
}

FUNCTION aktualizacia_rychlosti_A()
{
    nahodny1 = nahodne cislo z [0, 1)
    nahodny2 = nahodne cislo z [0, 1)

    FOR vsetky zlozky rychlosti (permutacie) AS i DO
    {
        // vyjadrenie vzdialenosti pozicie castice od pozicie personal best a od pozicie local best
        pismeno1 = pismeno na ktore sa zobrazuje i-te pismeno podla permutacie aktualnej pozicie castice
        pismeno2 = pismeno na ktore sa zobrazuje i-te pismeno podla permutacie pozicie personal best
        pismeno3 = pismeno na ktore sa zobrazuje i-te pismeno podla permutacie pozicie local best

        rychlost[i] = rychlost[i] * zotrvacnost
        rychlost[i] += nahodny1 * personal_koeficient * (vzdialenost pismeno1 a pismeno2 v abecede, v absolutnej hodnote)
        rychlost[i] += nahodny2 * local_koeficient * (vzdialenost pismeno1 a pismeno3 v abecede, v absolutnej hodnote)
    }
}

FUNCTION aktualizacia_rychlosti_B()
{
    nahodny1 = nahodne cislo z [0, 1)
    nahodny2 = nahodne cislo z [0, 1)

    FOR vsetky zlozky rychlosti (permutacie) AS i DO
    {
        // vyjadrenie vzdialenosti pozicie castice od pozicie personal best a od pozicie local best
        frekvencia1 = frekvencia vyskytu pismena na ktore sa zobrazuje i-te pismeno podla aktualnej pozicie castice
        frekvencia2 = frekvencia vyskytu pismena na ktore sa zobrazuje i-te pismeno podla pozicie personal best
        frekvencia3 = frekvencia vyskytu pismena na ktore sa zobrazuje i-te pismeno podla pozicie local best

        rychlost[i] = rychlost[i] * zotrvacnost
        rychlost[i] += nahodny1 * personal_koeficient * absolutna_hodnota(frekvencia1 - frekvencia2)
        rychlost[i] += nahodny2 * local_koeficient * absolutna_hodnota(frekvencia1 - frekvencia3)
    }
}

FUNCTION aktualizacia_rychlosti_C()
{
    nahodny1 = nahodne cislo z [0, 1)
    nahodny2 = nahodne cislo z [0, 1)

    FOR vsetky zlozky rychlosti (permutacie) AS i DO
    {
        // vyjadrenie vzdialenosti pozicie castice od pozicie personal best v zlozke i
        hodnota1 = IF pozicia[i] = personal_best_pozicia[i] THEN 0 ELSE 1 ENDIF
        // vyjadrenie vzdialenosti pozicie castice od pozicie local best v zlozke i
        hodnota2 = IF pozicia[i] = local_best_pozicia[i] THEN 0 ELSE 1 ENDIF

        rychlost[i] = rychlost[i] * zotrvacnost
        rychlost[i] += nahodny1 * personal_koeficient * hodnota1
        rychlost[i] += nahodny2 * local_koeficient * hodnota2
    }
}

FUNCTION aktualizacia_rychlosti_D()
{
    fungovanie tejto funkcie je identicke s funkciou typu "A", s tym rozdielom ze
    pri vzdialenosti pismen neuvazujeme zoradenie standardnej abecedy ale zoradenie
    pismen podla frekvencie ich vyskutu v anglickom texte
}
```

Dodatok C

Pseudokód - DES

```
personal_koeficient = 1.49445  // koeficient kognitivneho zrychlenia
local_koeficient = 1.49445  // koeficient socialneho zrychlenia

FUNCTION DES_PSO()
{
    FOR vsetky castice DO
    {
        pozicia = nahodny bitovy retazec (vektor)
        // castice mame v cyklickom poli
        sused1 = castica nalavo v poli od nej
        sused2 = castica napravo v poli od neh
        zotrvacnost = nahodna hodnova z [0.5, 1)
        rychlost = nulovy vektor // pole rovnako dlhe ako samotna permutacia
    }

    FOR FROM 1 TO pocet kol pso DO
    {
        zavolaj funkciu kolo_DES_PSO()
    }

    return local best permutacia (pozicia) castice, s najlepsim ohodnotenim svojej local best pozicie
}

FUNCTION kolo_DES_PSO()
{
    FOR vsetky castice DO
    {
        IF local_best > ohodnotenie aktualnej pozicie THEN
            local_best = ohodnotenie aktualnej pozicie

        IF personal_best > ohodnotenie aktualnej pozicie THEN
            personal_best = ohodnotenie aktualnej pozicie
    }

    FOR vsetky castice DO
    {
        IF local_best > sused1.personal_best THEN local_best = sused1.personal_best
        IF local_best > sused2.personal_best THEN local_best = sused2.personal_best
    }

    FOR vsetky castice AS castica DO
    {
        nahodny1 = nahodne cislo z [0, 1)
        nahodny2 = nahodne cislo z [0, 1)

        FOR vsetky zlozky rychlosti castice AS i DO
        {
            rychlost[i] = zotrvacnost * rychlost[i]
            rychlost[i] += personal_koeficient * nahodny1 * (personal best pozicia[i] - pozicia[i])
            rychlost[i] += local_koeficient * nahodny2 * (local best pozicia[i] - pozicia[i]);
        }

        FOR vsetky zlozky rychlosti castice AS i DO
        {
            nahodne_cislo = nahodne cislo z [0, 1)
```

```
            // sigmoida je sigmoidna funkcia y = 1 /(1 + e^(-x))
            IF nahodne_cislo < sigmoida(rychlost[i]) THEN pozicia[i] = 1 ELSE pozicia[i] = 0
        }
    }
}
```

Dodatok D

Pseudokód - Diferenčná charakteristika

```
dlzka_cesty = ??  // dlzka cesty

FUNCTION inicializacia_castice(znahodni)
{
    IF (znahodni) THEN
    {
        pozicia = nahodny binarny vektor[(dlzka_cesty + 2) * 32]   // dlzka vektora je (dlzka_cesty + 2) * 32

        // vypocet pravdepodobnosti ktora sa pouziva pri zmene hodnoty bitov pri pohybe castice
        rand = nahodne cislo z [0, 1)
        pravdepodobnost_1 = (rand * 0.2) * (rand * 0.2)
    }

    x = nahodne cele cislo z [0, dlzka_cesty]

    FOR i FROM x + 2 UPTO dlzka_cesty - 1 DO
    {
        L = podretazec z pozicia od indexu (i - 2) * 32 s dlzkou 32
        R = podretazec z pozicia od indexu (i - 1) * 32 s dlzkou 32

        ER = E(R)  // kde E je expanzivna funkcia DESu

        FOR s FROM 1 UPTO 8 DO
        {
            input = podretazec z ER od indexu (s - 1) * 6 s dlzkou 6

            out[s] = 4 bitovy retazec, tento vyberieme nahodne podla pravdepodobnosti
            distribucnej tabulky pre dany s-ty SBox a vstup input
        }

        output = spojenie vsetkych out[i] pre i z 1 az 8
        output = P(output) // kde P standardna permutacia pouzivana v DESe
        oldL = L;
        L = R;
        R = output XOR oldL

        nasledne vlozime retazec R do retazca pozicia pocnuc indexom i * 32
    }

    FOR i FROM x - 1 DOWNTO 0 DO
    {
        L = podretazec z pozicia od indexu (i + 2) * 32 s dlzkou 32
        R = podretazec z pozicia od indexu (i + 1) * 32 s dlzkou 32

        zvysok kodu FOR cyklu je identicky ako v predoslom FOR cykle
    }
}

FUNCTION CESTY_PSO()
{
    FOR vsetky castice DO
```

```
    {
        zavolaj funkciu inicilizacia_castice(TRUE)
    }

    FOR FROM 1 UPTO pocet kol pso DO
    {
        zavolaj funkciu kolo_CESTY_PSO()
    }

    return global_best_pozicia, ktora bola najdena pocas behu algoritmu
}

FUNCTION kolo_CESTY_PSO()
{
    FOR vsetky castice DO
    {
        IF global_best_pozicia > ohodnotenie aktualnej pozicie THEN
            global_best_pozicia = ohodnotenie aktualnej pozicie
    }

    nahodna = vybereme nahodnu casticu spomedzi vsetkych
    nahodna.inicializacia_castice(FALSE)

    nahodna = vybereme nahodnu casticu spomedzi vsetkych
    nahodna.pozicia = global_best_pozicia

    FOR vsetky castice DO
    {
        FOR vsetky kola cesty (pozicie) castive AS k DO
        {
            FOR vsetky SBoxy v kole k AS s DO
            {
                pravdepobnost_SBox_k_s = vycislime pravdepodobnost ktorou prispieva dany S-Box
                do celkovej pravdepodobnosti cesty

                vaha_SBox_k_s = 1 / pravdepobnost_SBox_k_s
            }

            vaha_kola_k = MAX(vaha_SBox_k_s) pre s z 1 az 8
        }

        K = cislo kola, vybraneho podla vah vaha_kola_k (nahodne)
        S = cislo SBoxu, vybraneho z kola K podla vah vaha_SBox_k_s (nahodne)

        FOR vsetky indexy pola pozicia, ktorych bity vstupuju do vybraneho Sboxu S z kola K AS i DO
        {
            h = nahodne cislo z [0, 1)
            IF (h < pravdepodobnost_1) THEN pozicia[i] = 1 ELSE pozicia[i] = 0

        }

        IF (pozicia je obsahuje same nuly) THEN
        {
            zavolaj funkciu inicializacia_castice(TRUE)
        }
    }
}
```

Literatúra

[1] http://www.rsa.com/rsalabs/node.asp?id=2227.

[2] E Biham and A. Shamir. Differential cryptanalysis of des-like cryptosystems. In *Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '90, pages 2–21, 1991.

[3] E. Biham and A. Shamir. Differential cryptanalysis of the full 16-round des. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 487–496, London, UK, 1993. Springer-Verlag.

[4] A.J. Clark. *Optimization Heuristics for Cryptology.* PhD thesis, Queensland University of Technology, 1998.

[5] M. Clerc and J. Kennedy. The particle swarm - explosion, stability and convergence in multidimensional complex space. *IEEE Transaction on Evolutionary Computation*, 6(1):58–73, 2002.

[6] D. Coppersmith. The data encryption standard (des) and its strength against attacks. *IBM J. Res. Dev.*, 38(3):243–250, May 1994.

[7] N. Courtois. The best differential characteristics and subtleties of the biham-shamir attacks on des. In *on DES, On eprint.iacr.org/2005/202. Nicolas T. Courtois and Gregory V.* Bard, 2005.

[8] N. Courtois, G. Castagnos, and Goubin L. What do DES S-boxes say to each other? *Cryptology ePrint Archive*, Report 2003/184, 2003.

[9] E. Dawson, H. Gustafson, and A.N. Pettitt. Strict key avalanche criterion. *The Australasian Journal of Combinatorics*, 6:147–153, 1992.

[10] X. Hu, R.C. Eberhart, and Y. Shi. Swarm intelligence for permutation optimization: a case study of n-queens problem. In *Proceedings*

of the 2003 IEEE on Swarm Intelligence Symposium, pages 243–246, 2003.

[11] J. Kennedy. Bare bones particle swarms. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 80–87, 2003.

[12] J. Kennedy and R. C. Eberhart. A discrete binary version of the particle swarm algorithm. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 5, pages 4104–4108, 1997.

[13] J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *Proc. IEEE Int'l. Conf. on Neural Networks*, volume IV, pages 1942–31848, 1995.

[14] J. Kennedy and R. Mendes. Population structure and particle swarm performance. *Computational Intelligence, Proceedings of the World on Congress on*, 2:1671–1676, 2002.

[15] L. R. Knudsen. Iterative characteristics of des and s²-des. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 497–511, London, UK, 1993. Springer-Verlag.

[16] B. Koh, A.D. George, R.T. Haftka, and B.J. Fregly. B.: Parallel asynchronous particle swarm optimization. In *International Journal for Numerical Methods in Engineering 67(4*, pages 578–595, 2006.

[17] M. Matsui. The first experimental cryptanalysis of the data encryption standard. In *Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '94, pages 1–11, London, UK, 1994. Springer-Verlag.

[18] R. Mendes. *Population Topologies and their influence in Particle Swarm Optimization*. PhD thesis, Universidade do Minho, 2004.

[19] S. Murphy. The cryptanalysis of feal-4 with 20 chosen plaintexts. *J. Cryptol.*, 2(3):145–154, September 1990.

[20] National Institute of Standards and Technology(NIST). Announcing the data encryption standard, January 1977. Originally issued by National Bureau of Standards.

[21] E. Ozcan and C.K. Mohan. Analysis of a simple particle swarm optimization system. *Intelligent Engeneering Systems Through artificial Neural Networks*, pages 253–258, 1998.

[22] M. Pedersen. Good parameters for particle swarm optimization. *Hvass Laboratories Technical Report no. HL1001*, 2010.

[23] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1:33–57, 2007.

[24] C. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pages 25–34, 1987.

[25] W. Shahzad, A. B. Siddiqui, and F. A. Khan. Cryptanalysis of four-rounded des using binary particle swarm optimization. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 1757–1758. ACM, 2009.

[26] Y. H. Shi and R. C. Eberhart. A modified particle swarm optimizer. In *IEEE International Conference on Evolutionary Computation*, Anchorage, Alaska, 1998.

[27] D. R. Stinson. *Cryptography: Theory and Practice*. Chapman & Hall/CRC, 2002.