Uniwersytet Śląski
Wydział Nauk Ścisłych i Technicznych
Instytut Informatyki

Rozprawa doktorska

Kamil Dworak

Algorytmy memetyczne w kryptoanalizie różnicowej symetrycznych szyfrów blokowych

Promotor: prof. dr hab. Urszula Boryczka

Sosnowiec, 2022 r.

Rodzicom, za wsparcie przez wszystkie lata nauki, dedykuję i dziękuję.

Szczególnie dziękuję promotorowi,

Pani prof. dr hab. Urszuli Boryczce, bez której niniejsza rozprawa by nie powstała. Bardzo dziękuję za możliwość współpracy, niezwykłe zaangażowanie, owocne dyskusje i wszystkie uwagi. Jestem wdzięczny za wsparcie podczas wzlotów i upadków związanych nie tylko z pracą naukową. Dziękuję również wszystkim współpracownikom z Zakładu Algorytmiki i Inteligencji Obliczeniowej, na których pomoc zawsze mogłem liczyć.

Spis treści

1 Wstęp **1**

2 Podstawy kryptologii i bezpieczeństwa informacji **6**

2.1 Terminologia bezpieczeństwa informacji 7

2.2 Rodzaje algorytmów kryptograficznych 8

2.3 Zasady projektowania szyfrów blokowych 12

2.4 Wybrane algorytmy kryptografii symetrycznej 15

2.4.1 Algorytm szybkiego przetwarzania danych *FEAL* 16

2.4.2 Standard szyfrowania danych *DES* 18

2.5 Podstawowe metody ataków kryptoanalitycznych 20

2.5.1 Kryptoanaliza różnicowa 21

2.5.2 Kryptoanaliza liniowa 23

3 Algorytmy ewolucyjne **25**

3.1 Uogólniony algorytm ewolucyjny 27

3.1.1 Selekcja najlepszych osobników 30

3.1.2 Charakterystyka procesu sukcesji 32

3.1.3 Rekombinacja osobników rodzicielskich 34

3.1.4 Rola operatora mutacji 38

4 Algorytmy memetyczne **42**

4.1 Uogólniony algorytm memetyczny 42

4.2 Metody przeszukiwania lokalnego 45

4.2.1 Algorytm wspinaczki 45

4.2.2 Symulowane wyżarzanie 47

4.2.3 Przeszukiwanie z listą Tabu 49

5 Algorytmy memetyczne w kryptoanalizie symetrycznych szyfrów blokowych 52
5.1 Atak memetyczny skierowany na szyfr *FEAL* 52
5.2 Atak memetyczny skierowany na szyfr *DES* 55
5.3 Inne algorytmy ewolucyjne w problemie kryptoanalizie szyfrów blokowych 60
5.3.1 Atak na szyfr *FEAL* z wykorzystaniem algorytmów ewolucyjnych 60
5.3.2 Atak na szyfr *DES* z wykorzystaniem algorytmu *DPSO* 62

6 Badania eksperymentalne zaproponowanych algorytmów 66
6.1 Przygotowanie zbiorów testowych 66
6.2 Dobór wartości parametrów 68
6.3 Badanie jakości rozwiązań zaproponowanych algorytmów 70
6.3.1 Kryptoanaliza z wykorzystaniem ataków *NEA* oraz *MASA* . . . 70
6.3.2 Kryptoanaliza z wykorzystaniem ataku *DPSO* 87

7 Analiza skuteczności opracowanych algorytmów oraz wnioski z przeprowadzonych eksperymentów 94
7.1 Porównanie algorytmów pod kątem liczby sprawdzonych rozwiązań . . 94
7.2 Porównanie algorytmów pod kątem czasu odszyfrowania kryptogramu . 97
7.3 Statystyczna weryfikacja poprawności opracowanych algorytmów 100
7.4 Wnioski 101

8 Podsumowanie 103

Bibliografia 106

A Dodatek 116

B Dodatek 132

Spis symboli i skrótów 140

Słownik pojęć 144

Spis algorytmów 150

Spis rysunków 151

Spis tabel 153

1

Wstęp

Rosnąca popularność komputeryzacji, a przy tym samego Internetu, pociąga za sobą wzrost zapotrzebowania na coraz bardziej zaawansowane metody zabezpieczeń. Wiąże się to z przede wszystkim z usprawnieniem obecnie wykorzystywanych technik bezpieczeństwa. Współczesne standardy zabezpieczeń systemów informatycznych powinny charakteryzować się wysokim stopniem [13]:

- **nienaruszalności** - zapewniającym, że przesyłana informacja nie może zostać zmodyfikowana w czasie transmisji,
- **poufności** - gwarantującym, że wiadomość, nie może być odczytana przez nikogo niepowołanego,
- **niezaprzeczalności** - nadawca, po wysłaniu wiadomości, nie może się jej wyprzeć, ani stwierdzić, że nigdy jej nie wysłał.

Termin bezpieczeństwa informacji nie jest związany tylko i wyłącznie z przechowywaniem samych danych, ale również z ich przetwarzaniem oraz wymianą. Ograniczenia polegające na standardowej kontroli dostępu poszczególnych użytkowników lub podstawowym uwierzytelnieniu stały się w dzisiejszych czasach niewystarczające. Przez ostatnie kilkadziesiąt lat inżynierowie z tej dziedziny zaprojektowali specjalne protokoły oraz algorytmy kryptograficzne spełniające wyszczególnione wcześniej aspekty bezpieczeństwa.

Głównym założeniem kryptografii jest utrzymanie w tajemnicy rzeczywistego wizerunku informacji. Algorytm szyfrujący nie odpowiada za ukrycie samego faktu istnienia wiadomości, a przekształcenie jej w taki sposób, aby była ona czytelna tylko i wyłącznie dla nadawcy oraz przeznaczonemu jej odbiorcy.

Kryptografia pociąga za sobą pojęcie kryptoanalizy. Przy pomocy odpowiednich metod matematycznych, pozwala ona udowodnić, że dany szyfr lub system kryptograficzny nie zapewnia wystarczająco dobrej ochrony. Najczęściej sprowadza się to do odgadnięcia właściwego klucza deszyfrującego, przy pomocy którego możliwe staje się odszyfrowanie przechwyconego kryptogramu.

Współczesne symetryczne szyfry blokowe realizują proces transformacji tekstu jawnego wykorzystując sieć Feistala oraz uogólnioną sieć podstawień i permutacji. Wprowadzenie tego typu przekształceń oraz odpowiednio długich kluczy spowodowało, że weryfikacja współczesnych algorytmów szyfrujących stała się znacznie trudniejsza. W 1990 roku upubliczniono całkowicie nową metodę kryptoanalityczną, mianowicie **kryptoanalizę różnicową**. Do dzisiaj, obok kryptoanalizy liniowej, jest ona jedną z najpopularniejszych technik ataku, skierowaną przeciwko współczesnym symetrycznym szyfrom blokowym.

W przypadku najnowocześniejszych i najbardziej zaawansowanych algorytmów szyfrujących sama kryptoanaliza różnicowa okazuje się być mało skuteczna. Mimo specjalnie „spreparowanego", ataku proces ten jest zbyt czasochłonny. W celu usprawnienia wydajności ataku zaproponowano połączenie algorytmów metaheurystycznych wraz z algorytmem kryptoanalizy różnicowej.

Na ogół algorytmy metaheurystyczne wykorzystywane są do uzyskiwania rozwiązań przybliżonych. W przypadku kryptologii koniecznym staje się odgadnięcie idealnego klucza deszyfrującego. Dzięki obecności tzw. **efektu lawinowego**, dostępnego w każdym współczesnym algorytmie szyfrującym, zmiana dowolnego bitu wejściowego pociąga za sobą proces wygenerowania całkowicie nowego szyfrogramu.

Zaproponowane podejście pozwala na automatyczne odrzucenie rozwiązań (kluczy) reprezentujących najgorszą funkcję przystosowania, dzięki czemu możliwe staje się zawężenie przestrzeni poszukiwań. Dodatkowe właściwości analityczne algorytmów memetycznych usprawniają proces przeszukiwania lokalnego w taki sposób, aby osiągnąć jak najlepsze rozwiązanie w jak najkrótszym czasie. Tego typu rozwiązanie pozwala na odgadnięcie od 90% do 100% bitów ostatniego podklucza dla każdej z charakterystyk Ω, w ostatniej rundzie algorytmu szyfrującego, co umożliwia wygenerowanie poprawnego klucza deszyfrującego. W przypadku, gdy znajdujemy się blisko optimum - mowa tutaj o brakujących 10% bitów podklucza, zaleca się wykorzystanie klasycznego przeszukiwania wyczerpującego - np. najprostszego ataku siłowego. Rozwiązanie to skutkuje znacznie szybszym przeszukiwaniem zbioru potencjalnych rozwiązań w przeciwieństwie do standardowego algorytmu kryptoanalizy różnicowej.

Dobre rezultaty algorytmów memetycznych sugerują ich efektywne wykorzystanie podczas badania podatności na wszelkiego rodzaju ataki kryptoanalityczne innych symetrycznych szyfrów blokowych. Mogą one również znaleźć zastosowanie w aktualnie wdrożonych systemach kryptograficznych wykorzystywanych w przemyśle.

Teza rozprawy

Zastosowanie algorytmów memetycznych, w procesach kryptoanalizy różnicowej, dla wybranych symetrycznych szyfrów blokowych, poprawia efektywność samego ataku, a tym samym jego skuteczność.

Cel rozprawy

Głównym celem pracy jest opracowanie i przeanalizowanie algorytmu memetycznego usprawniającego proces kryptoanalizy różnicowej. Zadaniem zaproponowanego ataku jest odgadnięcie właściwego klucza deszyfrującego, niezbędnego do odszyfrowania kryptogramu wygenerowanego za pomocą wybranego symetrycznego szyfru blokowego, w czasie znacznie krótszym, niż robi to obecnie klasyczny algorytm kryptoanalizy różnicowej. Tak powstały algorytm memetyczny należy porównać z innymi technikami metaheurystycznymi zaproponowanymi w literaturze. Kolejnymi etapami realizacji celu nadrzędnego są m.in.:

- przedstawienie stanu wiedzy dotyczącej aktualnie istniejących metod kryptoanalizy na przykładzie szyfrów: *FEAL* oraz *DES*,
- przekształcenie zagadnień związanych z kryptoanalizą symetrycznych szyfrów blokowych w klasyczny problem optymalizacji funkcji, ze szczególnym uwzględnieniem opracowania funkcji przystosowania osobnika. Pozwoli to na wykorzystanie wszelkich algorytmów metaheurystycznych dla omawianego problemu kryptoanalizy różnicowej,
- przygotowanie podstawowej wersji algorytmu memetycznego, na przykładzie szyfrów: *FEAL* oraz *DES* wraz z opracowaniem modyfikacji algorytmu memetycznego związanych głównie z usprawnieniem procesu przeszukiwania lokalnego,
- zaprojektowanie i zaimplementowanie oprogramowania pozwalającego na automatyczną ewolucyjną kryptoanalizę różnicową na przykładzie szyfrów: *FEAL* oraz *DES*,

- przebadanie zaproponowanego algorytmu pod kątem wartości parametrów, wyboru najkorzystniejszej modyfikacji oraz określenie ich wpływu na wyniki uzyskane przez algorytm. Rezultatem tego etapu będą ustalone optymalne wartości analizowanych parametrów.

Częściowe wyniki przedstawione w rozprawie zostały zaprezentowane na międzynarodowych konferencjach oraz artykułach naukowych [22–25].

Rozprawa składa się z 8 rozdziałów oraz dwóch dodatków. Po przedstawieniu tematyki oraz postawieniu celów rozprawy, w rozdziale 2, opisano najważniejsze aspekty teoretyczne, takie jak podstawowe zagadnienia związane z tematem kryptografii oraz kryptoanalizy. Przedstawiono również główne zasady projektowania symetrycznych szyfrów blokowych oraz uogólniony podział algorytmów szyfrujących wraz z szczegółowym opisem szyfrów *FEAL* oraz *DES*. Na samym końcu rozdziału „pochylono" się nad tematem dwóch najczęściej wykorzystywanych ataków w tego typu algorytmach szyfrujących, mianowicie nad kryptoanalizą różnicową i liniową.

Rozdział 3 zawiera opis podstawowego algorytmu ewolucyjnego oraz innych technik ewolucyjnych. W rozdziale przestawiono ideę samego algorytmu, a także jego poszczególne mechanizmy. W szczególności skupiono się na podstawowych operatorach genetycznych, takich jak krzyżowanie czy mutacja, z uwzględnieniem najczęściej wykorzystywanych wariantów owych operatorów. Zawarto również szczegółowy opis najpopularniejszych technik selekcji. Scharakteryzowano też, niezwykle ważny w tego typu algorytmach, proces sukcesji.

W rozdziale 4 skupiono się na idei algorytmów memetycznych. Ta część zawiera dokładny opis podstawowego algorytmu memetycznego oraz charakterystykę najczęściej wykorzystywanych algorytmów przeszukiwania lokalnego. Wśród nich znaleźć można opis algorytmu wspinaczki, symulowanego wyżarzania oraz przeszukiwania z listą Tabu. Każdy algorytm został opisany wraz z jego najpopularniejszymi modyfikacjami.

Rozdział 5 to szczegółowy opis dwóch oddzielnych wariantów autorskiego ataku memetycznego, dedykowanego każdemu z rozpatrywanych algorytmów szyfrujących tj. szyfrom *FEAL* oraz *DES*. Zagadnienia poruszane w tej części rozprawy dotyczyć będą ewolucyjnej kryptoanalizy różnicowej. W rozdziale opisane zostaną również dwa dodatkowe ataki ewolucyjne, jeden przeciwko szyfrowi *FEAL* [22] oraz standardowi szyfrowania danych *DES* [24], oraz drugi dedykowany szyfrowi *DES* [33, 34].

Rozdział 6 został poświęcony opisowi przygotowanych wcześniej zbiorów testowych dla każdego z ataków przedstawionych w rozdziale piątym. Następnie skupiono się na szczegółowym opisaniu środowiska uruchomieniowego dla wszystkich przetestowanych

algorytmów oraz przedstawieniu wszystkich dobranych parametrów dla każdego z przebadanych ataków kryptoanalitycznych. W rozdziale zostały również przedstawione wyniki z przeprowadzonych eksperymentów zarówno dla zaproponowanych algorytmów, jak i dla tych, które zostały zaczerpnięte z literatury.

Rozdział 7 zawiera dokładną analizę skuteczności opracowanych ataków, zarówno pod kątem liczby sprawdzonych podkluczy, jak i pod kątem czasu odszyfrowania kryptogramu bez oraz ze znajomością poprawnego klucza deszyfrującego. Na zakończenie przedstawione zostały wnioski, wynikające z przeprowadzonych eksperymentów.

Rozprawa zakończona jest krótkim podsumowaniem dotyczącym poszczególnych etapów badań oraz przedstawionych we wstępie zrealizowanych celach pracy. W ramach tego rozdziału zaproponowano również dalsze kierunki rozwoju badań. Do pracy dołączono dodatek A, opisujący szczegółowe wyniki dla charakterystyk Ω szyfru *DES* oraz pozostałe wartości funkcji przystosowania, nie zamieszczone w pracy, dla algorytmu *DPSO* i szyfru *DES*. Ponadto do rozprawy dołączono również dodatek B, zawierający spis wszystkich wykorzystanych w procesie kryptoanalizy różnicowej par tekstów jawnych oraz odpowiadających nim szyfrogramów dla charakterystyki Ω_1, pozwalających na zreprodukowanie zaproponowanych ataków. Na samym końcu rozprawy znajduje się dodatkowo spis symboli i skrótów oraz słownik pojęć.

2

Podstawy kryptologii i bezpieczeństwa informacji

Problem dotyczący przekazu poufnych informacji znany był już w starożytności. Goniec z tajnym listem, musiał dostarczyć zaszyfrowaną wiadomość do adresata w jak najkrótszym czasie. Odbiorca, posługując się prostym przedmiotem umożliwiającym odszyfrowanie listu, był w stanie zapoznać się z jego treścią [48]. Wraz z biegiem czasu, metody przekazu informacji ulegały stopniowemu unowocześnieniu. Opracowano m.in. znany do dzisiaj alfabet Morse'a [48], który już wtedy stanowił pewnego rodzaju szyfr.

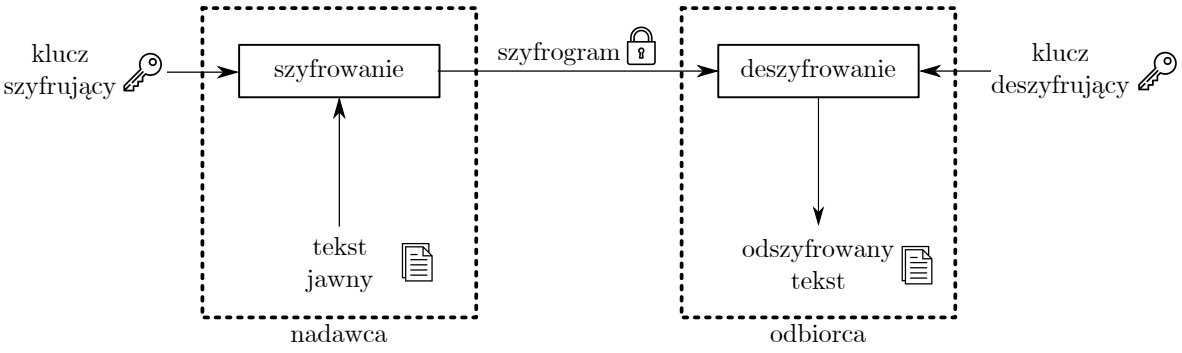
Ze względu na stopień bezpieczeństwa, informacje możemy podzielić na dwie grupy. Pierwsza z grup to informacje o bardzo wysokim priorytecie, wykorzystywane m.in. przez służby wojskowe lub wszelkiego rodzaju korporacje. Ujawnienie tego typu informacji wiązałoby się z poważnymi stratami finansowymi lub mogłoby zagrozić bezpieczeństwu np. całego kraju [86]. Druga grupa to informacje o znacznie mniejszym priorytecie, np. plan zajęć lub lista zakupów na najbliższy tydzień. Zważywszy na stopień bezpieczeństwa pierwszej z wymienionych powyżej grup konieczne stało się wprowadzenie bardziej restrykcyjnych środków bezpieczeństwa, jakim bez wątpienia jest kryptologia [86].

Kryptologia stanowi dyscyplinę nauki zajmującą się równocześnie kryptografią oraz kryptoanalizą. **Kryptografia** jest dziedziną skupiającą się na tworzeniu: algorytmów szyfrujących, protokołów, realizacji systemów wykorzystywanych do ochrony informacji oraz tematem szyfrowania. **Kryptoanaliza** natomiast zajmuje się odszyfrowaniem przechwyconego szyfrogramu bez znajomości klucza deszyfrującego. Znajduje ona zastosowanie w wyszukiwaniu luk i niedopatrzeń w aktualnie wykorzystywanych

szyfrach oraz systemach kryptograficznych [50, 76].

2.1 Terminologia bezpieczeństwa informacji

Na początku tego rozdziału koniecznym staje się wprowadzenie kilku niezbędnych terminów wykorzystywanych przez inżynierów i specjalistów w dziedzinie bezpieczeństwa informacji. Oryginalna wiadomość, oznaczana jako P, nazywana jest **tekstem jawnym** (ang. *plaintext*) [50]. Stanowi ona czytelny dla każdego tekst, widoczny obraz lub możliwe i zrozumiałe do przesłuchania nagranie audio. Wynikiem procesu szyfrowania jest **kryptogram** (ang. *ciphertext*) [50], zwany również **szyfrogramem**, oznaczany jako C. Jest to informacja przekształcona do takiej postaci, że staje się ona całkowicie nieczytelna - może przypominać ciąg pseudolosowo wygenerowanych znaków binarnych lub obraz imitujący jakieś zakłócenia. Proces transformacji tekstu jawnego na szyfrogram nazywany jest **szyfrowaniem** (ang. *encryption*), natomiast proces zamiany kryptogramu na tekst jawny określany jest jako **deszyfrowanie** (ang. *decryption*) [50]. Proces szyfrowania, zarówno jak i deszyfrowania, opiera się na zastosowaniu odpowiedniej funkcji matematycznej, pozwalającej na zrealizowane jednej z wymienionych operacji. Często można spotkać się z dodatkowym określeniem definiującym algorytm szyfrujący jako **schemat szyfrowania**. Uproszczony schemat działania algorytmu szyfrującego został przedstawiony na rys. 2.1:



Rysunek 2.1: Uproszczony schemat działania algorytmu szyfrującego

Należy również wspomnieć, że do realizacji procesu szyfrowania niezbędny jest **klucz szyfrujący** [50], oznaczany jako K_E. Samo przekształcenie wiadomości będzie mało pożyteczne, ponieważ odbiorca nie będzie w stanie jej odczytać. Dopiero za pomocą właściwego **klucza deszyfrującego** K_D, odbiorca jest w stanie przeprowadzić proces deszyfrowania [50]. Kryptoanalityk, który przechwycił zaszyfrowaną wiadomość, nie będzie mógł jej w jakikolwiek sposób odczytać bez znajomości wspomnianego klucza

deszyfrującego. Przykład procesu szyfrowania z udziałem tekstu jawnego oraz klucza szyfrującego został przedstawiony poniżej:

P = she was walking for the fierce sun was directly overhead her umbrella
K_E = XøR¡PÍ6
C = Á1aèÒ,©ÁÒ,©¿H‡ÜOÒ¿©Ú‡¿†ÒRQÒz1aèÒR‡èQÊèÒ©ßQ‡¿ÒÁ]Ü

Na podstawie rys. 2.1 oraz powyższego przykładu można zdefiniować **funkcję szyfrującą** E oraz **funkcję deszyfrującą** D na podstawie następujących formuł:

$$E(P, K_E) = C, \tag{2.1}$$

$$D(C, K_D) = P. \tag{2.2}$$

Warto wspomnieć, że każdy poprawnie zaprojektowany schemat szyfrowania danych powinien być odwracalny. Oznacza to, że dowolnie odszyfrowany tekst musi być taki sam, jak podany przed procesem szyfrowania oryginalny tekst jawny, co można zapisać jako:

$$D(E(P, K_E), K_D) = P. \tag{2.3}$$

Gdyby ta własność nie mogła być spełniona, nie bylibyśmy w stanie odszyfrować wygenerowanego kryptogramu. W takim przypadku zastosowanie kryptografii nie miałoby jakiegokolwiek sensu. Należy również zaznaczyć, że istnieje w pewna grupa algorytmów szyfrujących, w których własność ta nie jest zachowana. Mowa tutaj o jednokierunkowych funkcjach skrótu, zwanych również funkcjami haszującymi [92].

2.2 Rodzaje algorytmów kryptograficznych

Współczesne algorytmy kryptograficzne można podzielić na trzy podstawowe grupy:

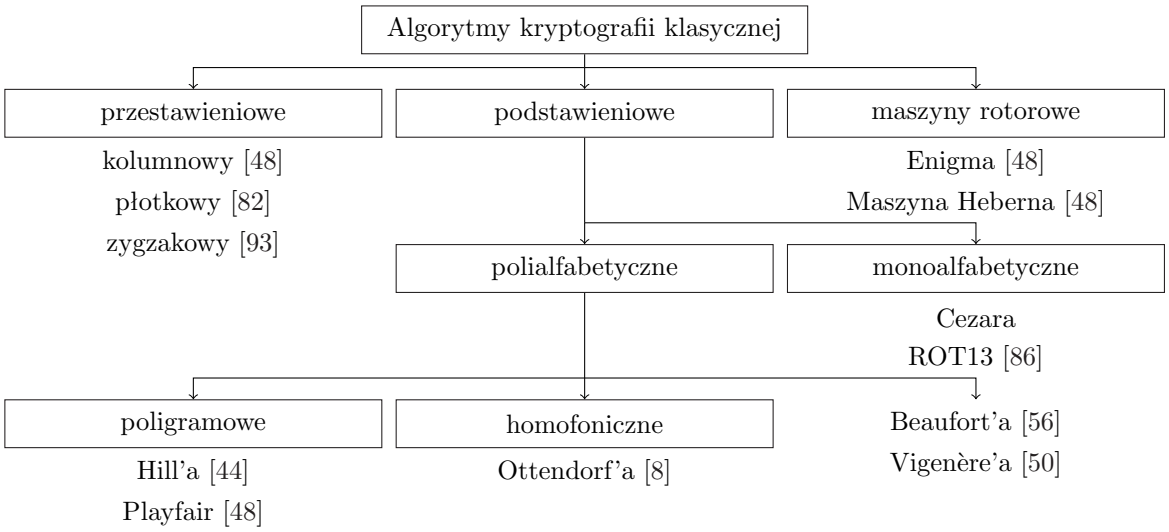
- algorytmy symetryczne - m. in. szyfry blokowe oraz strumieniowe,
- algorytmy asymetryczne,
- funkcje haszujące.

Jako czwartą grupę można zaliczyć nieużyteczną już dzisiaj kryptografię klasyczną.

W dzisiejszych czasach wszelkie szyfry klasyczne są zbyt proste do złamania przy pomocy zwykłego komputera domowego, a co za tym idzie nie gwarantują żadnego, nawet minimalnego, bezpieczeństwa. Wyróżnia się dwie podstawowe grupy tego typu algorytmów:

- **szyfry przestawieniowe** - ich podstawowym założeniem jest obecność tych samych znaków w szyfrogramie oraz tekście jawnym. Znaki występują nawet w tej samej liczbie, zmieniony jest jedynie ich układ i kolejność,
- **szyfry podstawieniowe** - monoalfabetyczne szyfry podstawieniowe zastępują każdy znak w tekście jawnym dokładnie jednym znakiem w szyfrogramie według ustalonego klucza szyfrującego. W szyfrach polialfabetycznych dopuszcza się szyfrowanie wielu znaków jednocześnie.

Szczególnym przypadkiem szyfru podstawieniowego jest **szyfr homofoniczny**. W szyfrogramie, wygenerowanym za pomocą tego typu schematu, może dojść do sytuacji, w której większa liczba znaków odpowiada pojedynczej literze. Oznacza to, że znak „a" może być reprezentowany przez litery „b", „z" oraz „g". W przypadku algorytmów poligramowych szyfruje są grupy znaków, tzw. n-gramy, np. dwójce „AN" może odpowiadać digram „ZD". Polialfabetyczne szyfry podstawieniowe są złożeniem wielu prostych szyfrów podstawieniowych [86]. Warto również wspomnieć o niezwykle istotnych w czasie II Wojny Światowej maszynach rotorowych [48]. Na rys. 2.2 przedstawiono rodzinę najbardziej znanych algorytmów kryptografii klasycznej.



Rysunek 2.2: Algorytmy kryptografii klasycznej

Do dwóch najpopularniejszych grup współczesnych algorytmów szyfrujących zalicza się algorytmy **symetryczne** oraz **asymetryczne**.

W przypadku kryptografii symetrycznej, ten sam klucz wykorzystywany jest jednocześnie do szyfrowania i deszyfrowania informacji, co można zapisać jako $K_E = K_D$. Ze względu na rozmiar przetwarzanych danych wejściowych, algorytmy symetryczne dzieli się na dwie podstawowe grupy:

- **szyfry blokowe** - w trakcie szyfrowania wiadomość zostaje podzielona na określoną liczbę części, o takiej samej długości, np. na 64-bitowe bloki danych. Następnie przekazywane są one do właściwej funkcji szyfrującej. Z jednego bloku tekstu jawnego generowany jest dokładnie jeden blok szyfrogramu. W przypadku, gdy wiadomość nie może zostać podzielona na równomierne części, zostaje stworzony dodatkowy blok przechowujący ostatni, niepełny, fragment danych, a następnie, w celu uzyskania spójności, uzupełniany on jest zerami,
- **szyfry strumieniowe** - algorytmy te są szczególnym przypadkiem symetrycznych szyfrów blokowych. Długość bloku jest ograniczona do minimum. Tego typu algorytmy operują na skończonych ciągach danych po jednym bicie lub bajcie. W przypadku tego typu schematu, jeden bit lub bajt tekstu jawnego zostanie zaszyfrowany na jeden bit lub bajt szyfrogramu [86]. Szyfry te są nieco szybsze od opisanych wcześniej szyfrów blokowych.

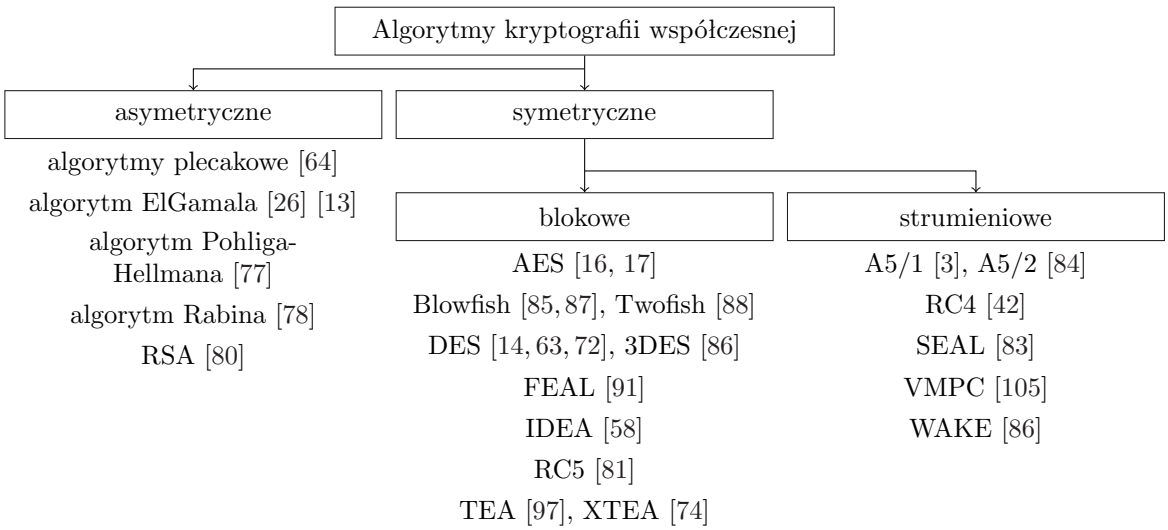
Całkowicie nową koncepcję wprowadza, zaproponowana nieco później, kryptografia **asymetryczna**. Opracowany kryptosystem pozwala na wygenerowanie i odczytanie szyfrogramu za pośrednictwem dwóch niezależnych od siebie kluczy: publicznego (klucz szyfrujący K_E) oraz prywatnego (klucz deszyfrujący K_D). Najważniejszym założeniem kryptografii asymetrycznej jest spełnienie warunku mówiącego o niemożliwości uzyskania jednego z tych kluczy przy pomocy drugiego [19]. Dzięki takiemu rozwiązaniu, stało się możliwe wykorzystanie publicznego kanału transmisji do przekazania klucza szyfrującego, zapewniając przy tym wysoki standard bezpieczeństwa zaszyfrowanych informacji [7].

Symetryczne szyfry blokowe najczęściej znajdują największe zastosowanie w bankowości. W szczególności wykorzystywane są one do szyfrowania wszelkiego rodzaju transakcji i płatności internetowych. Algorytmy te doskonale sprawują się również do szyfrowania większych wolumenów danych przechowywanych m.in. w magazynach, hurtowniach lub bazach danych. Do najpopularniejszych schematów szyfrowania blokowego można zaliczyć szyfry takie jak: *DES* oraz *AES*.

Szyfry strumieniowe wykorzystywane są w sytuacji, gdy mamy do czynienia z nieokreślonym zbiorem danych, przesyłającym kolejne informacje w czasie rzeczywistym. W tego typu sytuacjach, ważna jest prędkość szyfrowania i deszyfrowania. Przykładem zastosowania tego typu algorytmów może być standard telefonii komórkowej *GSM*. W trakcie nawiązywania połączenia inicjalizowany jest nowy strumień z danymi. Kolejne informacje przesyłane są z każdą sekundą trwania połączenia, przez co muszą być one szyfrowane po stronie nadawcy, oraz deszyfrowane u odbiorcy w czasie niemal rzeczywistym. Najbardziej znanym algorytmem szyfrowania strumieniowego jest szyfr *RC4*.

Algorytmy kryptografii asymetrycznej zazwyczaj wykorzystywane są do zaszyfrowania otwartej komunikacji wewnątrz kanałów transmisyjnych [7]. Przykładowym zastosowaniem tego typu szyfrów może być implementacja kanału *SSL* (ang. *Secure Socket Layer*) oraz standardu protokołów komunikacyjnych *SSH* (ang. *Secure Shell*) używanych w sieciach komputerowych *TCP/IP*. Schematy szyfrowania kryptografii asymetrycznej znajdują również szeroki wachlarz zastosowań w implementacji podpisów cyfrowych. Najpopularniejszym reprezentantem kryptografii asymetrycznej jest bez wątpienia algorytm *RSA* (ang. *Rivest–Shamir–Adleman*).

Na rys. 2.3 przedstawiono podział najbardziej znanych algorytmów kryptografii współczesnej:



Rysunek 2.3: Algorytmy symetryczne oraz asymetryczne kryptografii współczesnej

Ostatnią grupą algorytmów kryptograficznych są **jednokierunkowe funkcje skrótu**, znane również jako **funkcje haszujące**. Ich głównym zadaniem jest przekształcenie tekstu jawnego, o dowolnej długości, w nieczytelny, a co najistotniejsze, nieodwracalny ciąg znaków, o pewnej z góry ustalonej długości. Wygenerowany za pomocą jedno-

kierunkowej funkcji skrótu ciąg nazywany jest haszem. Właściwa funkcja haszująca powinna dawać w wyniku zbiór wartości o rozkładzie równomiernym dla odpowiednio dużego zbioru komunikatów wejściowych [93]. Narzuca to funkcjom skrótu dodatkowe wymaganie, mianowicie odporność na kolizje.

Do najpopularniejszych jednokierunkowych funkcji skrótu zalicza się: *MD6* [79], *SHA-3* [9] oraz *BLAKE2* [5]. Funkcje haszujące znajdują zastosowanie do przechowywania najwrażliwszych informacji, których odszyfrowanie nigdy nie powinno być możliwe. Najczęściej spotykanymi przykładami tego typu danych mogą być hasła systemowe. Hasło wprowadzone przez użytkownika, podczas np. operacji logowania zostaje zahaszowane, a dopiero później wygenerowany ciąg porównywany jest ze skrótem przechowywanym w bazie danych. Funkcje skrótu znajdują również zastosowanie przy weryfikacji spójności danych wszelkiego rodzaju plików z danymi oraz wiadomości podpisanej przy pomocy podpisu cyfrowego.

2.3 Zasady projektowania szyfrów blokowych

Algorytm szyfrujący powinien być zaprojektowany w taki sposób, aby kryptoanalityk, uzbrojony w cały zbiór szyfrogramów oraz odpowiadających nim tekstów jawnych, nie był w stanie odszyfrować przechwyconego szyfrogramu, ani odgadnąć klucza deszyfrującego [93].

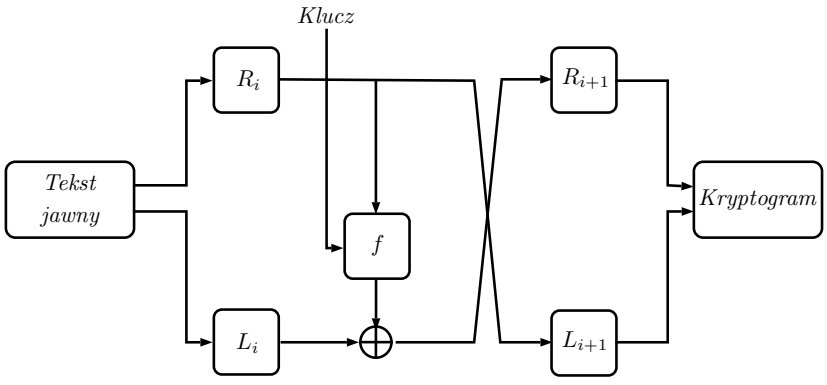
Proces szyfrowania powinien być zależny tylko i wyłącznie od algorytmu szyfrującego oraz klucza. Mogłoby się wydawać, że utrzymanie tych dwóch elementów w tajemnicy stanowi idealne rozwiązanie. W przypadku większości dzisiejszych szyfrów autorzy decydują się na ujawnienie swoich algorytmów. Jedno z podstawowych twierdzeń kryptografii zakłada, że intruz mimo tego, że ma pełny dostęp do algorytmu szyfrującego, nie jest w stanie odszyfrować przechwyconego szyfrogramu [52]. Jeżeli możliwym nie jest złamanie algorytmu kryptograficznego, mając wiedzę jak on działa, to na pewno nie jest możliwe jego złamanie, tej wiedzy nie posiadając [86, 93]. Sytuacja ta idealnie odwzorowuje jedną ze zdefiniowanych przez Kerckhoffsa rad i reguł, mianowicie: „Projekt systemu nie powinien wymagać jego tajności, a ewentualnie jego ujawnienie nie powinno przysparzać kłopotów korespondentom.” [53]. Reguła ta nazywana jest zasadą Kerckhoffsa, dzięki której ukształtowało się twierdzenie, że bezpieczeństwo szyfru powinno być oparte tylko i wyłącznie na bezpieczeństwie jego klucza [52]. Zasadę tę spełniają praktycznie wszystkie wykorzystywane algorytmy szyfrujące - specyfikacje każdego szyfru można znaleźć np. w Internecie. Większość niejawnych algorytmów,

która w jakikolwiek sposób stała się chwilowo upubliczniona, została bardzo szybko złamana.

Jednymi z najważniejszych zasad, które muszą zostać spełnione, aby szyfr był bezpieczny, jest obecność odpowiednio dużego stopnia przemieszania oraz rozproszenia informacji [54]. Zasady te zostały szczegółowo opisane przez Shannona w [90]. Przemieszanie poszczególnych bitów pozwala na ukrycie powiązań pomiędzy tekstem jawnym, kryptogramem, a kluczem szyfrującym. Rozproszenie wykorzystywane jest do ukrywania powiązań statystycznych pomiędzy tekstem jawnym a kluczem, co bez wątpienia utrudnia proces kryptoanalizy [86].

Niemal każdy współczesny algorytm opiera się na wykorzystaniu prostych operacji podstawiania oraz przestawiania [93]. Podstawienie polega na zastąpieniu każdego z elementów bloku danych innym elementem, natomiast przedstawienie sprowadza się do zamiany kolejności poszczególnych elementów. Szyfry, w dużej mierze wykorzystujące te dwie operacje, nazywane są sieciami podstawieniowo-permutacyjnymi (ang. *substitution-permutation network - SPN*) [51, 86].

Większość symetrycznych szyfrów blokowych opiera się również na zastosowaniu sieci Feistela [28]. Wejściowy blok danych zostaje podzielony na dwie równe części, lewą L oraz prawą P. Następnie definiowany jest specjalny cykl, który będzie powtarzany iteracyjne w każdej rundzie algorytmu szyfrującego. Pojedynczy cykl sieci Feistela został przedstawiony na poniższym schemacie (rys. 2.4):



Rysunek 2.4: Pojedynczy cykl sieci Feistela

Analizując rys. 2.4, można zauważyć, że każdy następny cykl sieci $i + 1$ zależy od poprzedniego i. Danymi wejściowymi dla każdej iteracji są bloki L_i, R_i oraz podklucz dla każdej rundy algorytmu szyfrującego K_i. Na podstawie przedstawionego na powyższym schemacie pojedynczego cyklu sieci Feistela półbloki L_{i+1} i R_{i+1} można zdefiniować jako:

$$\begin{aligned} L_{i+i} &= R_i \\ R_{i+i} &= L_i \oplus f(R_i, K_i), \end{aligned} \tag{2.4}$$

gdzie:

- L_i, R_i - aktualna lewa i prawa część bloku z danymi,
- L_{i+1}, R_{i+1} - lewa oraz prawa część bloku w kolejnej rundzie,
- K_i - podklucz dedykowany każdej z rund,
- f - funkcja rundy algorytmu szyfrującego.

Podczas każdej rundy prawa część R_i przekazywana jest do funkcji rundy f. Następnie w celu wygenerowania ciągu R_i wynik funkcji rundy poddawany jest operacji różnicy symetrycznej wraz z lewym podblokiem L_i. Lewa część L_{i+1} wyznaczana jest automatycznie poprzez prostą operację kopiowania podbloku R_i. Na ogół funkcja f jest odwracalna, aczkolwiek nie zawsze musi tak być [86]. Przykładowymi szyframi, korzystającymi z struktury sieci Feistela są: *DES*, *FEAL*, *Blowfish* oraz *TEA*.

Większość szyfrów blokowych została zaprojektowana w taki sposób, aby efekt lawinowy (ang. *Avalanche Effect*) był jak największy oraz występował już od samego początku algorytmu szyfrującego. Zjawisko efektu lawinowego gwarantuje, że każda minimalna modyfikacja danych wejściowych - w tekście jawnym lub kluczu szyfrującym, pociąga za sobą znaczące zmiany wewnątrz danych wyjściowych, czyli w szyfrogramie. Najczęściej zmiana jednego bitu wejściowego wymusza zmianę co najmniej połowy bitów wyjściowych. Ponadto stan każdego bitu wyjściowego zależny jest od każdego bitu wejściowego [86]. Występowanie efektu lawinowego, wraz z obecnością dwóch wspomnianych powyżej struktur, tj. sieci podstawieniowo-permutacyjnych oraz sieci Feistela, w jednym algorytmie szyfrującym, znacznie utrudnia proces kryptoanalizy szyfru.

Podczas projektowania szyfru blokowego, warto zwrócić uwagę na rozmiar bloku oraz klucza szyfrującego. Szyfr pracując na większych blokach danych staje się bezpieczniejszy, aczkolwiek traci na prędkości oraz wydajności podczas szyfrowania i deszyfrowania. Jako minimalny rozmiar bloku przyjmuje się 64-bity. Sytuacja wygląda podobnie w przypadku długości klucza. Dłuższy klucz zapewnia większe bezpieczeństwo, aczkolwiek wydłuża czas pracy algorytmu.

W przypadku szyfrów iteracyjnych, duże znaczenie ma zastosowana w algorytmie liczba rund. Jako minimum zakłada się przynajmniej 16 cykli. Większa złożoność funkcji rundy f, wraz z procesem generowania podkluczy, przekłada się na bardziej złożoną kryptoanalizę, ale również znacznie dłuższy czas działania algorytmu szyfrującego.

Prawdziwym problemem jest zaprojektowanie właściwego symetrycznego szyfru blokowego w taki sposób, aby był on nie tylko bezpieczny, ale również przejrzysty, prosty oraz łatwy w zaimplementowaniu. Schemat, który można łatwiej przeanalizować staje się bardziej odporny pod kątem potencjalnych luk i niedopatrzeń. Należy również pamiętać o tym, że wszystkie algorytmy szyfrujące stosowane są wewnątrz wszelkiego rodzaju systemów informatycznych oraz stanowią część aplikacji. Podczas projektowania szyfru czas jest ważnym parametrem algorytmu, dlatego proces szyfrowania nie może trwać zbyt długo.

Cytując Bruce'a Schneiera [86], jednego z najpopularniejszych kryptografów można powiedzieć, że:

> „*Prawdziwą sztuką i powodem, dla którego projektowanie szyfrów blokowych w rzeczywistości jest bardzo trudne, jest zaprojektowanie szyfru z możliwe najkrótszym kluczem, stosując możliwie małą pamięć i uzyskując możliwe najkrótszy czas szyfrowania*".

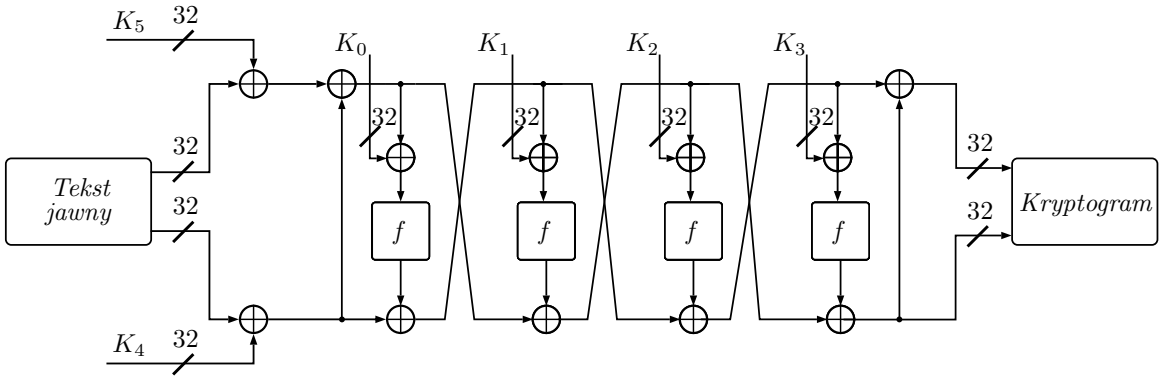
2.4 Wybrane algorytmy kryptografii symetrycznej

W rozprawie zdecydowano się na wybór dwóch symetrycznych szyfrów blokowych, pozwalających na udowodnienie postawionej tezy w akceptowalnym czasie. Jednym z wybranych szyfrów jest algorytm szybkiego przetwarzania danych (ang. *Fast Data Encipherment Algorithm*) *FEAL*. Szyfr ten w dużej mierze opiera się na wykorzystaniu sieci Feistela. Ze względu na swoją prostotę, wydajność oraz stosunkowo niską odporność na atak, algorytm ten stał się idealnym kandydatem do badań opracowanego ataku.

Drugim z wybranych algorytmów szyfrujących został jeden z najpopularniejszych szyfrów, mianowicie standard szyfrowania danych (ang. *Data Encryption Standard*) *DES*. Algorytm ten, podobnie jak wspomniany wcześniej szyfr *FEAL*, opiera się na wykorzystaniu sieci Feistela oraz dodatkowo sieci *SP*. Ze względu na swoją popularność, pierwsze komercyjne zastosowanie w przemyśle oraz potwierdzony w przeszłości pozytywnie przeprowadzony atak na uproszczoną wersję tego algorytmu stał się on atrakcyjnym szyfrem do przeprowadzenia badań zaproponowanego ataku memetycznego.

2.4.1 Algorytm szybkiego przetwarzania danych *FEAL*

Algorytm *FEAL* został zaprojektowany w 1987 roku przez Shimizu oraz Miyaguchi [91]. Stanowi on czterorundowy symetrycznym szyfr blokowy operujący na 64-bitowych blokach danych i posługujący się 64-bitowym kluczem szyfrującym. Zamiarem twórców było stworzenie szyfru znacznie efektywniejszego, a zarazem równie bezpiecznego, co popularny wówczas standard szyfrowania danych *DES*. Z każdą rundą algorytm miał stawać się silniejszy, a co za tym idzie, odporniejszy na wszelkiego rodzaju ataki. Szyfr ten nie wykorzystuje żadnych permutacji, ani tabel podstawień - ogranicza się tylko i wyłącznie do prostych operacji bitowych, typu alternatywy wykluczającej. Po jakimś czasie odkryto, że algorytm ten nie zapewnia wystarczającego bezpieczeństwa. Rozszerzenie algorytmu o dodatkową liczbę rund szyfrujących - nawet do 16 lub 32, przysporzyło kryptoanalitykom nieco trudności. Szyfr ten miał znaczący wpływ w rozwoju stosowanej wówczas kryptoanalizy. Z historycznego punktu widzenia *FEAL* jest bardzo ważnym schematem. Algorytm ten pozwolił na znaczny rozwój wszelkich metod kryptoanalitycznych. Szyfr *FEAL* jest pierwszym algorytmem, na którym zastosowano atak z wykorzystaniem kryptoanalizy różnicowej [94]. Schemat działania algorytmu szyfrującego został przedstawiony na rys. 2.5.



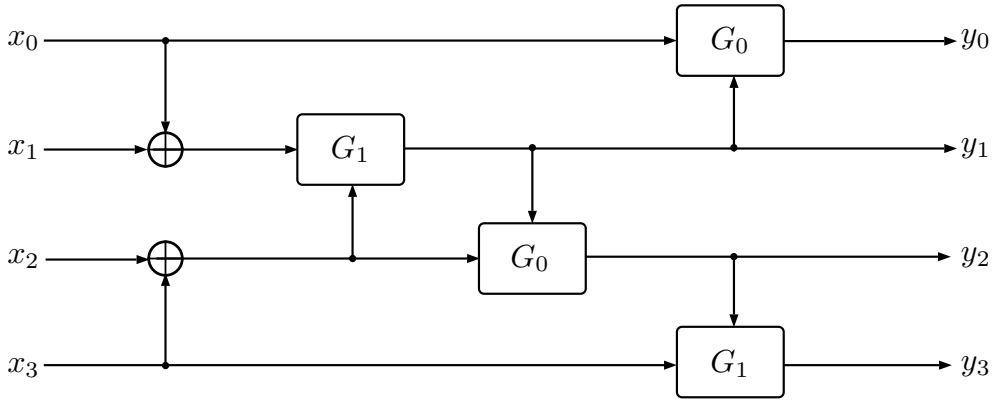
Rysunek 2.5: Algorytm szyfrujący *FEAL*

Na powyższym rysunku przedstawiono zmodyfikowaną wersję algorytmu *FEAL*. W celu zwiększenia stopnia złożoności problemu, klucz został podzielony na sześć 32-bitowych podkluczy [94], oryginalna wersja operuje na dwunastu 16-bitowych podkluczach. Dokładny opis rozkładu głównego klucza szyfrującego na podklucze został opisany w [86, 91].

Na samym początku 64-bitowy blok tekstu jawnego zostaje podzielony na dwie 32-bitowe części. Lewa część zostaje poddana operacji różnicy symetrycznej z podkluczem K_4, natomiast prawa z K_5. Następnie lewa i prawa część poddawane są działaniu róż-

nicy symetrycznej, wskutek czego powstaje nowa prawa część. Właśnie wygenerowany fragment danych, wraz z obliczoną wcześniej lewą częścią, zostają przekazane przez cztery, niemal identyczne, cykle szyfrujące zwane rundami. W każdym cyklu prawa część poddawana jest operacji różnicy symetrycznej z 32-bitowym podkluczem rundy. Otrzymany wynik zostaje przekazany do właściwej funkcji szyfrującej f. Obliczona wartość funkcji rundy poddawana jest po raz kolejny działaniu różnicy symetrycznej, tym razem z wygenerowanym wcześniej lewym blokiem. Na końcu cyklu lewa oraz prawa część zamieniane są miejscami. Po zakończeniu wszystkich czterech rund ostatnia prawa część dodatkowo poddawana jest operacji różnicy symetrycznej z lewą. Dwie 32-bitowe części zostają połączone tworząc 64-bitowy blok szyfrogramu [94].

Na poniższym rysunku przedstawiono funkcję rundy algorytmu *FEAL*. Funkcja f jako parametr wejściowy przyjmuje 32-bitowy blok danych, który następnie zostaje podzielony na cztery 8-bitowe części (x_0, x_1, x_2, x_3). Wygenerowane fragmenty danych poddawane są działaniu różnicy symetrycznej oraz funkcji G_0 i G_1, zgodnie z schematem przedstawionym na rys. 2.6.



Rysunek 2.6: Funkcja rundy f algorytm szyfrującego *FEAL*

Funkcje G_0 oraz G_1 określone są na podstawie poniższego wyrażenia:

$$G_x(a, b) = (a + b + x(mod256)) \lll 2. \tag{2.5}$$

Symbol $\lll$ oznacza operację cyklicznego przesunięcia w lewo. Na podstawie powyższego wyrażenia, oraz rys. 2.6, można wyznaczyć następujące wartości:

$$y_0 = G_0(x_0, y_1), \tag{2.6}$$

$$y_1 = G_1(x_0 \oplus x_1, x_2 \oplus x_3), \tag{2.7}$$

$$y_2 = G_0(y_1, x_2 \oplus x_3), \tag{2.8}$$

$$y_3 = G_1(y_2, x_3). \tag{2.9}$$

Konkatenacja wszystkich obliczonych powyżej powyższych wartości (y_0, y_1, y_2, y_3) umożliwia zwrócenie 32-bitowej wartości funkcji rundy f [94].

Proces deszyfrowania realizowany jest przez ten sam algorytm co szyfrowanie, z tym, że poszczególne podklucze każdej z rund stosowane są w odwrotnej kolejności.

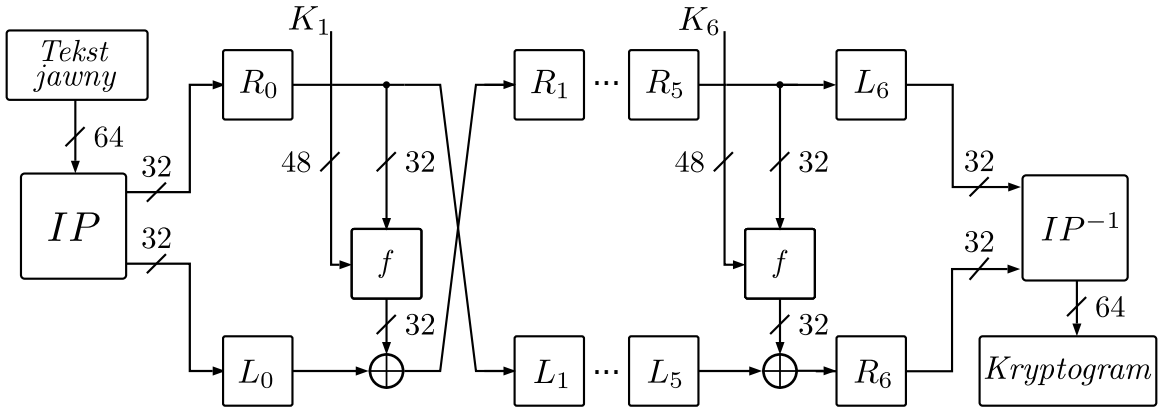
2.4.2 Standard szyfrowania danych *DES*

Standard szyfrowania danych *DES* został zaprojektowany w 1975 roku przez IBM, na bazie szyfru Lucifer [72,76]. W 1977 został przyjęty jako światowy standard ISO [86]. Sam algorytm określany jest jako Data Encryption Algorithm *DEA*, aczkolwiek w większości publikacji nazywa się go *DES*. Szyfr ten jako pierwszy został wykorzystany komercyjnie [63] w wielu aplikacjach biznesowych. Do dzisiaj wiele jego odmian, takich jak np. *3DES*, znajduje zastosowanie w sektorze usług bankowych. Algorytm *DES* składa się z kilkunastu rund odpowiadających za wymieszanie kryptogramów częściowych [76]. Przez kilka lat szyfr ten opierał się wszelkim atakom kryptoanalitycznym.

DES został zaprojektowany w taki sposób, aby efekt lawinowy występował od samego początku algorytmu [86]. Zmiana jednego bitu wejściowego wymusza zmianę co najmniej połowy bitów wyjściowych. Ponadto stan każdego bitu wyjściowego jest zależny od każdego bitu wejściowego [76].

Przedstawiony szyfr transformuje 64-bitowe bloki tekstu jawnego na 64-bitowe bloki szyfrogramu, posługując się przy tym 64-bitowym kluczem szyfrującym K [63, 93]. Na początku klucz redukowany jest do 56 bitów poprzez usunięcie co ósmego bitu - wykorzystywanego do weryfikacji poprawności wprowadzonego klucza szyfrującego [86]. Następnie K zostaje poddany rozkładowi na zbiór sześciu 48-bitowych podkluczy, dedykowanych każdej z rund algorytmu, $K_1, ..., K_6$ - opis szczegółowego rozkładu klucza można znaleźć w [63,76,86,93,95]. Na rys. 2.7 przedstawiono 6-rundowy algorytm *DES*.

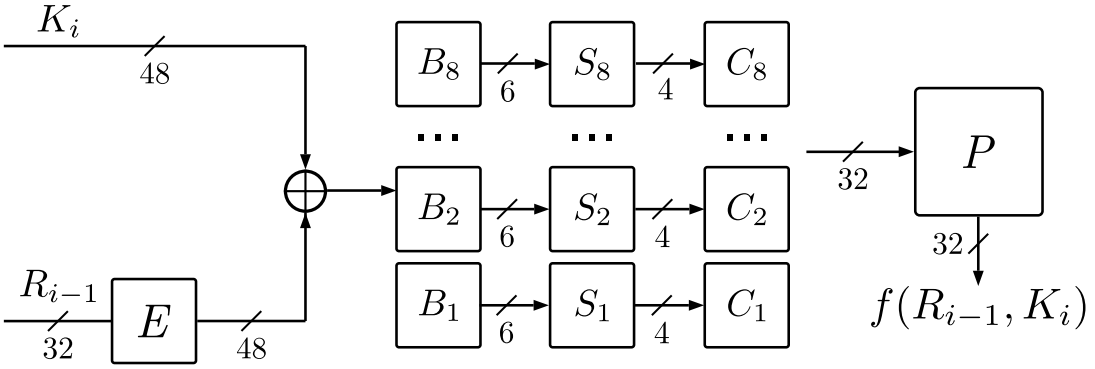
Po wygenerowaniu podkluczy, właściwy proces szyfrowania może zostać rozpoczęty. Fragment tekstu jawnego zostaje poddany permutacji początkowej IP. Wygenerowany blok zostaje podzielony na dwie 32-bitowe części, prawą R oraz lewą L. Następnie zostaje wykonanych sześć identycznych cykli, w których prawa część R_i zostaje przekazana do funkcji rundy f wraz z podkluczem K_i. Wygenerowany blok zostaje poddany operacji alternatywy wykluczającej z lewą częścią L_i, tym samym tworząc nową prawą część R_{i+1}. Nowa lewa część L_{i+1} odpowiada prawej części z poprzedniej rundy R_i.



Rysunek 2.7: Sześcio-rundowy algorytm szyfrujący *DES*

Po wykonaniu wszystkich sześciu rund, części lewa L_6 i prawa R_6 zostają złączone w jeden 64-bitowy blok, który następnie przekazany jest do permutacji odwrotnej IP^{-1}. Wynikiem transpozycji poszczególnych bitów jest 64-bitowy blok szyfrogramu.

Funkcja rundy f algorytmu *DES* została przedstawiona na rys. 2.8. Na wejściu podawana jest 32-bitowa porcja danych, która zostaje przekazana do permutacji rozszerzającej E. Zadaniem tego przekształcenia jest wyrównanie długości przekazanego bloku do rozmiaru podklucza poprzez duplikację wybranych bitów. Pozwalając jednemu bitowi wpływać na dwa podstawienia, zwiększany jest efekt lawinowy [86]. Otrzymany ciąg jest sumowany modulo 2 z bitami podklucza, a następnie podzielony na osiem 6-bitowych bloków $B_1, ..., B_8$.



Rysunek 2.8: Funkcja rundy f algorytmu *DES*

Każdy z bloków B_j przekazywany jest do specjalnych macierzy podstawień zwanym S-blokami S_j. Macierze te wykorzystywane są do kompresji danych wejściowych - przetwarzają 6-bitowe wejście na 4-bitowe wyjście. S_j składają się z liczb całkowitych z zakresu od 0 do 15, zapisanych w macierzach o długości szesnastu kolumn oraz czterech wierszy. Pierwszy i ostatni bit 6-bitowego ciągu B_j wyznacza numer wiersza.

Pozostałe cztery bity oznaczają numer kolumny, z której zostanie wybrana zwracana wartość [63, 86].

S_j są jedynym nieliniowym elementem szyfru *DES*. Zmiana jednego bitu w ciągu wejściowym może prowadzić do zmiany wszystkich bitów wyjściowych. Przeprowadzane w nich przekształcenia utrudniają kryptoanalizę całego szyfru.

Na końcu funkcji f wygenerowane ciągi konkatenowane są ze sobą w jeden 32-bitowy blok danych, który ostatecznie przekazywany jest do permutacji P. Jej zadaniem jest odwzorowanie każdego bitu wejściowego na dokładnie jeden bit wyjściowy nie dublując, ani nie pomijając żadnego z nich [86].

Podobnie, jak to miało miejsce w przypadku schematu szyfrującego *FEAL*, proces deszyfrowania realizowany jest przez ten sam algorytm, który został wykorzystany podczas szyfrowania, z tym wyjątkiem, że poszczególne podklucze wykorzystywane są w odwrotnej kolejności.

2.5 Podstawowe metody ataków kryptoanalitycznych

Jedną z najbardziej znanych technik kryptoanalitycznych jest metoda przeglądu zupełnego [93], zwaną potocznie jako atak siłowy (ang. *Brute Force Attack*) [47, 50]. Polega ona na sprawdzeniu wszystkich możliwych rozwiązań do momentu, aż uda się znaleźć właściwy klucz deszyfrujący. Opierając się na tej metodzie można przeprowadzić atak, który teoretycznie gwarantuje odgadnięcie poprawnego klucza, aczkolwiek sprawdzanie wszystkich możliwych kombinacji może stanowić długotrwały, a czasami nawet nieosiągalny proces. W przypadku niektórych algorytmów, np. w przypadku szyfru *AES*, rozmiar przestrzeni rozwiązań sięga nawet 2^{256} kluczy. Wykorzystanie ataku opierającego się o tą metodę przeglądu zupełnego nie będzie miało najmniejszego sensu.

Inną popularną techniką jest metoda łamania z szyfrogramami (ang. *Ciphertext Only Attack*). Opiera się ona na odgadnięciu klucza, będąc w posiadaniu skończonego zbioru kryptogramów. Na ich podstawie kryptoanalityk stara się odgadnąć sam klucz lub chociaż część wiadomości, która może okazać się pomocna przy odgadnięciu klucza deszyfrującego. Szczególną odmianą tej metody jest technika łamania ze znanym tekstem jawnym (ang. *Known Plaintext Attack*). Kryptoanalityk wybiera fragment tekstu jawnego na podstawie znajomości tekstów jawnych. Doskonałym przykładem może być przechwycona, zaszyfrowana, baza konwersacji email. Na podstawie podobnej, a w niektórych przypadkach nawet identycznej, części powitalnej lub pożegnalnej

w wiadomości, kryptoanalityk jest w stanie wywnioskować, jaki klucz szyfrujący został wykorzystany do wygenerowania przechwyconych szyfrogramów.

Jedną z najczęściej wykorzystywanych technik jest metoda ataku z wybranym tekstem jawnym (ang. *Chosen Plaintext Attack*). W metodzie tej zakłada się, że kryptoanalityk ma bezpośredni dostęp do algorytmu szyfrującego. Atakujący wybiera tekst jawny, który ma zostać zaszyfrowany, a następnie analizuje wygenerowany kryptogram. Tekst jawny może zostać dobrany w taki sposób, aby otrzymany szyfrogram ujawniał trochę więcej informacji na temat samego klucza. Pochodnymi technikami ataku jest metoda łamania z wybranym szyfrogramem (ang. *Chosen Ciphertext Attack*), wykorzystywana głównie w systemach kryptograficznych z kluczem publicznym, oraz metoda łamania z adaptacyjnie wybranym tekstem jawnym (ang. *Adaptive Chosen Plaintext Attack*), w której kryptoanalityk może kilkukrotnie wprowadzać wybrany przez siebie wybrany tekst jawny, w zależności od otrzymanych wcześniej wyników [50].

Do innych, najczęściej wykorzystywanych ataków zalicza się również atak słownikowy (ang. *Dictionary Attack*), polegający na zredukowaniu przestrzeni rozwiązań poprzez wykorzystanie zbioru kluczy - np. słów, wraz z ich modyfikacjami - np. dodanie najczęściej wykorzystywanych kombinacji cyfr, dużych i wielkich liter czy znaków specjalnych. Niestety, znaczna większość użytkowników wszelkiego rodzaju systemów informatycznych wybiera hasła stosunkowo proste do zapamiętania, dzięki czemu metoda ta staje się skuteczną metodą ataku.

Warto jeszcze wspomnieć o ataku algebraicznym (ang. *Algebraic Attack*), polegającym na przekształceniu algorytmu szyfrującego do zbioru równań algebraicznych, które następnie należy rozwiązać. Technika ta sprawdza się znacznie lepiej niż metoda przeglądu zupełnego, co zostało udowodnione na przykładzie szyfru *AES* [50].

2.5.1 Kryptoanaliza różnicowa

Pojęcie kryptoanalizy różnicowej (ang. *Differential Cryptanalysis*) zostało wprowadzone przez Bihama oraz Shamira w 1990 roku [11, 12]. Zaproponowany algorytm opiera się na ataku z wybranym tekstem jawnym. Zakłada się, że kryptoanalityk ma dostęp do algorytmu szyfrującego, dzięki czemu może dobierać pary tekstów jawnych, różniących się od siebie w z góry określony sposób, oraz analizować wygenerowane z nich szyfrogramy. W większości symetrycznych szyfrów blokowych, różnica pomiędzy tekstami jawnymi wyznaczana jest na podstawie prostej operacji różnicy symetrycznej, co można zapisać jako $P' = P \oplus P^*$, gdzie P oraz P^* są dwoma spreparowanymi tekstami jawnymi. Pary te mogą być wygenerowane losowo, ważne jest tylko, aby róż-

nica P' była zgodna z wcześniej ustaloną wartością. Następnie obserwuje się, w jaki sposób zmienia się różnica zadanej pary w kolejnych rundach szyfru, aż do wygenerowania szyfrogramów. Wykorzystując różnicę pomiędzy tekstami w kolejnych rundach, dla większej liczby par, można przypisać różne prawdopodobieństwa, sugerujące poprawność niektórych podkluczy [12]. Podczas analizy kolejnych par tekstów jawnych oraz odpowiadających nim szyfrogramów okazuje się, że jeden klucz może być bardziej prawdopodobny od pozostałych [50].

Każdy współczesny szyfr charakteryzuje się pewną nieliniowością. Nie da się wyznaczyć żadnego wzorca, za pomocą którego można przewidzieć wartość funkcji dla kolejnego argumentu [12]. W niemal każdym symetrycznym szyfrze blokowym, za nieliniowość odpowiada funkcja rundy f. Każda z wszystkich możliwych różnic charakteryzuje się pewnym prawdopodobieństwem, które określa jak często funkcja f zwróci oczekiwaną wartość [12]. Różnice te będą dalej nazywane w pracy charakterystykami Ω. Marcin Karbowski definiuje je w swojej książce [50] jako „rozbieżności między szyfrogramami wynikające z różnic między tekstami jawnymi". Wszystkie możliwe charakterystyki mogą zostać wyznaczone za pomocą specjalnej macierzy, gdzie wiersze odpowiadać będą wszystkim możliwym różnicom symetrycznym bloków wejściowych, natomiast kolumny wszystkim możliwym różnicom symetrycznym bloków wyjściowych. Każdy z elementów określać będzie, ile razy suma bitów wyjściowych pojawia się dla wybranej sumy bitów wejściowych [86].

Na przykładzie algorytmu *DES* oraz na podstawie rys. 2.8, można wyznaczyć wejściową różnicę symetryczną B' zakładając, że $E = E(R_{i-1})$, za pomocą następującego wyrażenia:

$$B' = \Big|\Big|_{j=1}^{8} B_j \oplus B_j^* = \Big|\Big|_{j=1}^{8} (E_j(R_i) \oplus K_i) \oplus (E_j(R_i^*) \oplus K_i) = \Big|\Big|_{j=1}^{8} E_j \oplus E_j^*, \tag{2.10}$$

gdzie symbol $||$ oznacza konkatenację kolejnych bloków danych. Na podstawie powyższego zapisu, można wywnioskować, że B' nie ma nic wspólnego z wykorzystywanym podkluczem. Kiedy znana jest wartość każdego B'_j można wyznaczyć zbiór wszystkich uporządkowanych par (B_j, B_j^*) dla wejściowej różnicy symetrycznej zgodnie z sugestią opisaną w [95]:

$$\Delta(B'_j) = \{(B_j, B_j \oplus B'_j) : B_j \in (\mathbb{Z}_2)^6\}. \tag{2.11}$$

Obliczając wyjściową różnicę $C'_j = S_j(B_j) \oplus S_j(B_j^*)$, dla każdej 4-bitowej pary,

uzyskuje się rozkład wszystkich możliwych wejściowych do wszystkich wyjściowych różnic zgodnie z twierdzeniem opisanym w [95]:

$$IN_j(B'_j, C'_j) = \{B_j \in (\mathbb{Z}_2)^6 : S_j(B_j) \oplus S_j(B_j \oplus B'_j) = C'_j\}. \tag{2.12}$$

W większości przypadków rozkład ten będzie jednostajny. Zadaniem kryptoanalityka jest znalezienie rozkładów o jak największej niejednostajności. Na podstawie wzoru 2.12, można wyznaczyć dodatkowy zbiór testowy za pomocą następującego wyrażenia [95]:

$$test_j(E_j, E^*_j, C'_j) = \{B_j \oplus E_j : B_j \in IN_j(E'_j, C'_j)\}. \tag{2.13}$$

Jeśli w zbiorze $test_j$ liczba elementów równa jest mocy zbioru IN_j, to zbiór ten musi zawierać bity podklucza K_{ij} [95].

Metoda ta pozwala na odgadnięcie poprawnego klucza deszyfrującego, w przypadku pełnego algorytmu *DES*, przy pomocy 2^{47} wybranych tekstów jawnych i odpowiadających nim szyfrogramów.

2.5.2 Kryptoanaliza liniowa

Skupiając się na wszelkich metodach ataku, nie sposób wspomnieć o kryptoanalizie liniowej (ang. *Linear Cryptanalysis*). Metoda ta została opracowana przez Matsu w 1994 roku [61, 62]. Głównym założeniem tego ataku jest sprowadzenie wszelkich operacji i procesów szyfrujących do postaci funkcji liniowej. W przypadku symetrycznych szyfrów blokowych, gdzie większość operacji opiera się na wykorzystaniu operatorów różnicy symetrycznej oraz koniunkcji, aproksymacja liniowa pozwala na znalezienie układu równań liniowych o następującej postaci [93]:

$$P[\alpha_1, \alpha_2, ..., \alpha_a] \oplus C[\beta_1, \beta_2, ..., \beta_b] = K[\gamma_1, \gamma_2, ..., \gamma_c], \tag{2.14}$$

gdzie α, β i γ oznaczają z góry ustalone unikatowe pozycje bitowe. Wartości tych pozycji znane są z prawdopodobieństwem $p \neq 0,5$. Oznacza to, że im bardziej prawdopodobieństwo będzie różne od 50% tym lepszym przybliżeniem będzie rozwiązanie układu równań [93]. Zapis $P[\alpha_1, \alpha_2, ..., \alpha_a]$ stanowi uproszczenie następującego wyrażenia:

$$P[\alpha_1, \alpha_2, ..., \alpha_a] = P[\alpha_1] \oplus P[\alpha_2] \oplus ... \oplus P[\alpha_a]. \tag{2.15}$$

Podobnie, jak to miało miejsce w przypadku kryptoanalizy różnicowej, koniecznym staje się stworzenie odpowiednio dużego zbioru tekstów jawnych oraz odpowiadających nim kryptogramów, tak aby aproksymacja była jak najlepsza [86]. Dysponując dużym zbiorem danych można rozpocząć proces przeliczania lewej strony równania 2.14 dla każdej z par. Podczas obliczeń zakłada się najbardziej prawdopodobną wartość wyrażenia $K[\gamma_1, \gamma_2, ..., \gamma_c]$ na podstawie częstości otrzymywanych wyników. Wynikiem przetworzenia wszystkich wygenerowanych par będzie układ równań liniowych, za pomocą którego można obliczyć poszczególne wartości klucza.

Metoda ta pozwala na odgadnięcie poprawnego klucza deszyfrującego, aczkolwiek wymaga ona bardzo dużej liczby znanych tekstów jawnych. W przypadku pełnego, 16-rundowego algorytmu *DES*, mowa tutaj o przynajmniej 2^{43} tekstach jawnych wraz z odpowiadającymi nim kryptogramami.

W literaturze można spotkać się z terminem kryptoanalizy różnicowo-liniowej, łączącej podejście jednej i drugiej metody ataku [59]. Atak ten opiera się na wykorzystaniu znacznie mniejszych zbiorów tekstów jawnych. Wydajnościowo atak ten porównywalny jest zarówno z kryptoanalizą różnicową oraz liniową.

3

Algorytmy ewolucyjne

Dla pewnej klasy problemów kombinatorycznych nie udało się opracować odpowiednich algorytmów, pozwalających na uzyskanie precyzyjnych rozwiązań w akceptowalnym czasie. Na ich potrzebę stworzono specjalne i bardzo wydajne algorytmy optymalizacyjne, których działanie ograniczać się będzie do wyznaczenia jednego z najlepszych, aczkolwiek niekoniecznie najlepszego, rozwiązania, z całej puli dostępnych wartości [46]. W wielu implementacjach dopuszcza się wygenerowanie całego zbioru akceptowalnych rozwiązań. Tego typu algorytmy wykonywane są w czasie znacznie krótszym, niż w przypadku najprostszego przeglądu zupełnego. Jeżeli wygenerowane w danej iteracji rozwiązanie osiągnęło satysfakcjonujący próg akceptacji działanie algorytmu zostaje przerwane.

Jednym z najpopularniejszych metod optymalizacyjnych są **metaheurystyki**. Techniki te stanowią uniwersalne algorytmy do rozwiązywania problemów obliczeniowych, dla których klasyczne metody są zbyt wolne lub nie są w stanie osiągnąć zadowalającego rozwiązania. Metody metaheurystyczne wykorzystywane są głównie do optymalizacji problemu poprzez iteracyjne poprawianie danego rozwiązania przy pomocy pewnej miary jakości. Z najważniejszych cech metaheurystyk można wymienić:

- nie gwarantują odnalezienia rozwiązania idealnego,
- wykorzystywane są do znajdowania rozwiązań przybliżonych w rozsądnym czasie,
- na podstawie rozwiązań przybliżonych można osiągnąć dokładniejszy wynik za pomocą metod przeglądu zupełnego lub innych algorytmów klasycznych.

Duża liczba tychże algorytmów opiera się na zastosowaniu wszelkich technik sztucznej inteligencji. Wśród nich znajdują się m.in. algorytmy ewolucyjne (ang. *Evolutionary*

Algorithms, EA). Stanowią one rozwiązania o charakterze stochastycznym, w których proces przeszukiwania przestrzeni rozwiązań naśladuje wybrane procesy pochodzące ze świata przyrody [67].

Natura posiada ogromną moc w poszerzaniu nowych form życia, zwłaszcza w procesie ewolucji. Bez wątpienia stało się to punktem zainteresowań wielu badaczy. Głównym ich celem było przeniesienie wybranych zasad i procesów ewolucyjnych z natury wprost do świata informatyki. Wynikiem ich pracy były implementacje wszelkiego rodzaju algorytmów o analogicznych funkcjach, które można spotkać w przyrodzie.

Na przestrzeni czasu, najpopularniejszymi metodami, zaproponowanymi w ramach uogólnionej grupy metaheurystyk wywodzących się z natury są:

- **strategie ewolucyjne** (ang. *Evolutionary Strategies, ES*) [89] - polegają one na wykorzystaniu operatora mutacji oraz selekcji w celu przeszukiwania globalnej przestrzeni rozwiązań,
- **algorytm genetyczny** (ang. *Genetic Algorithm, GA*) [45] - zasada działania tego algorytmu opiera się na darwinowskiej teorii dotyczycącej procesu ewolucji. Wśród tych algorytmów można spotkać się z takimi pojęciami jak walka o przetrwanie, rekombinacja czy mutacja osobników,
- **programowanie ewolucyjne** (ang. *Evolutionary Programming, EP*) [31] oraz **programowanie genetyczne** (ang. *Genetic Programming, GP*) [57] - metody te pozwalają na automatyczne tworzenie programów komputerowych,
- **systemy mrowiskowe** (ang. *Ant Systems, AS*) oraz **algorytmy optymalizacji mrowiskowej** (ang. *Ant Colony Optimization, ACO*) [20] - metody te są w stanie eksplorować przestrzeń potencjalnych rozwiązań naśladując zachowanie mrówek. W algorytmach tych obecne staje się zjawisko stygmergii, polegające na komunikowaniu się osobników za pomocą śladu feromonowego. Dzięki niemu mrówki są w stanie znaleźć pożywienie,
- **optymalizacja stadna cząsteczek** (ang. *Particle Swarm Optimization, PSO*) [27] - metoda ta polega na odwzorowaniu poruszania się zwierząt stadnych, takich jak np.: ptaki czy ławice ryb,
- **algorytm ewolucji różnicowej** (ang. *Differential Evolution, DE*) [96] - technika charakteryzująca się nowym podejściem do mutacji, która stał się nadrzędnym operatorem w stosunku do operacji krzyżowania dla tego algorytmu.

Pojawienie się opisanych powyżej technik metaheurystycznych poprzedzone było opracowaniem dodatkowego algorytmu, niezwykle ważnego w kontekście niniejszej rozprawy, mianowicie symulowanego wyżarzania (ang. *Simulated Annealing, SA*) [32]. Metoda ta polega na odwzorowaniu zjawisk fizycznych, zachodzących podczas wyżarzania się metali w metalurgii, po raz pierwszy opracowanego przez Metropolisa [66]. Do dzisiaj algorytm symulowanego wyżarzania stanowi jedną z najpopularniejszych technik przeszukiwania lokalnego. Alternatywną metodą jest dobrze znany algorytm wspinaczki (ang. *Hill Climbing*) oraz przeszukiwanie z listą Tabu (ang. *Tabu Search*) [35].

Jak już wspomniano wcześniej, rozwój metod metahurystycznych trwa do dzisiaj. Jesteśmy świadkami coraz to bardziej złożonych algorytmów, odzwierciedlających zachowania wszelkiego rodzaju zwierząt. Przykładami tego typu technik mogą być algorytmy pszczele (ang. *Bees Algorithm*) [49], nietoperzowe (ang. *Bat Algorithm*) [101], kukułcze (ang. *Cuckoo Search Algorithms*) [103], a nawet algorytmy robaczków świętojańskich (ang. *Firefly Algorithms*) [102]. Implementacje tego typu metod sprowadzają się do obserwacji specyficznych zachowań i interakcji pomiędzy wybranymi gatunkami zwierząt a następnie na przeniesieniu ich do świata informatycznego.

Powstanie obszernego wachlarza metod przeszukiwania lokalnego, wraz z ich modyfikacjami, wpłynęło na rozwój hybrydyzacji aktualnie wykorzystywanych metod. W literaturze można spotkać się z podejściami łączącymi *GA* wraz z np. algorytmem *PSO* [70] lub *ACO* [4]. Najpopularniejsze z nich, są bez wątpienia, algorytmy memetyczne (ang. *Memetic Algorithms, MA*) [71]. Stanowią one połączenie np. *GA*, pełniącego rolę narzędzia do eksploracji globalnej przestrzeni rozwiązań, oraz wybranego algorytmu przeszukiwania lokalnego, np. *SA*, odpowiadającego za proces eksploatacji.

Przedstawione powyżej metody opierają się na uogólnionym schemacie algorytmu ewolucyjnego. W tego typu rozwiązaniach operuje się na całym zbiorze osobników zwanych populacją, który ulega transformacji z każdą nową iteracją algorytmu. W związku z wykorzystaniem w rozprawie techniki algorytmów memetycznych zostaną one opisane w dalszej części tego rozdziału.

3.1 Uogólniony algorytm ewolucyjny

Algorytmy ewolucyjne opierają swój proces przeszukiwania na mechanizmach doboru naturalnego oraz dziedziczności [38]. Operują one na skończonym zbiorze rozwiązań, zwanym **populacją**. Podczas pracy algorytmu rozróżnia się dwa rodzaje populacji, tj. **populacją bazową** $P(t)$ (ang. *Base Population*), oraz **populacją potomną**

$O(t)$ (ang. *Offspring Population*), gdzie t oznacza aktualną iterację EA. W literaturze wprowadza się również pojęcie **populacji tymczasowej** $T(t)$ (ang. *Temporary Population*) [2].

Podczas działania algorytmu wprowadza się pojęcie **chromosomu** (ang. *Chromosome*), odnoszące się do odpowiednio zakodowanej struktury danych odwzorowującej osobnika. Tego typu struktura składa się ze skończonej liczby komórek, przechowujących informacje na temat osobnika, zwanych dalej genami. Każdy chromosom, dalej naprzemiennie nazywany osobnikiem, odpowiada dokładnie jednemu rozwiązaniu z całej przestrzeni potencjalnych rozwiązań - w przypadku tej rozprawy mowa będzie o kluczu lub podkluczu algorytmu szyfrującego. Warunkiem koniecznym każdego EA staje się zakodowanie każdego z osobników do takiej formy, aby była ona jak najbardziej przychylna procesowi ewolucji, a zwłaszcza realizacji oceny jakości przydatności osobnika.

W początkowej fazie działania algorytmu generowana jest **populacja początkowa** $P(0)$ (ang. *Initial Population*). Najczęściej zostaje ona wypełniona losowo wygenerowanymi osobnikami. W niektórych implementacjach wykorzystuje się specjalne algorytmy inicjalizujące populację $P(0)$ w taki sposób, aby uzyskać jak największą różnorodność osobników. Następnie tworzona jest główna pętla EA, realizująca proces ewolucji. Pierwszą instrukcją realizowaną w ramach pętli jest ocena wszystkich chromosomów za pomocą **funkcji przystosowania** F_f (ang. *Fintess Function*), określającej stopień przydatności każdego osobnika wewnątrz populacji. Z punktu widzenia natury przystosowanie określa jego zdolność do np. uniknięcia drapieżników [38].

Następnie rozpoczyna się do proces **selekcji** (ang. *Selection*). W ramach tej operacji wybiera się minimum dwóch osobników rodzicielskich. Wyselekcjonowane chromosomy mogą przystąpić do realizacji procesu rekombinacji. Oczywiście proces selekcji rzadko kiedy charakteryzuje się jednakowym prawdopodobieństwem - osobniki o największej wartości funkcji przystosowania cechują się większą szansą na przeżycie. W niektórych implementacjach dopuszcza się losowanie ze zwracaniem. Dzięki temu w populacji tymczasowej $T(t)$ znajduje się większa liczba lepiej przystosowanych osobników [2].

Kolejnym etapem EA jest proces **krzyżowania** (ang. *Crossover*). Najbardziej przystosowane osobniki, pochodzący z populacji tymczasowej $T(t)$, zostają dobrane w pary. Następnie na podstawie prostego losowania podejmowana jest decyzja o przystąpieniu do krzyżowania. Efektem poprawnie wykonanej rekombinacji są dwa nowo powstałe chromosomy potomne, które zastępują swoich rodziców wewnątrz populacji potomnej $O(t)$. W przypadku, gdy nie udało się skrzyżować dwoje osobników rodzicielskich naj-

częściej zostają one automatycznie przeniesione do populacji $O(t)$. O tym, jak często dochodzi do realizacji tego operatora, decyduje parametr prawdopodobieństwa krzyżowania, oznaczany jako p_c.

W dalszej kolejności uruchamiany jest proces **mutacji** (ang. *Mutation*). Operator ten polega na perturbacji genotypu wybranego osobnika potomnego. Warto zaznaczyć, że niewielkie perturbacje są bardziej prawdopodobne niż te duże [2]. Podobnie, jak ma to miejsce przy operatorze krzyżowania, obecność operatora mutacji obarczona jest pewnym prawdopodobieństwem, oznaczanym w literaturze jako p_m. Operator ten wykorzystywany jest do opuszczenia przez algorytm lokalnego ekstremum w celu znalezienia jeszcze bardziej interesującego rozwiązania.

Tak powstała populacja potomna $O(t)$ zostaje po raz kolejny poddana ocenie przy pomocy F_f. Następnie wygenerowany zbiór rozwiązań zostaje przekazany do następnej iteracji algorytmu, jako nowa populacja bazowa $P(t+1)$. Działanie głównej pętli algorytmu powtarzane jest do momentu spełnienia warunku stopu, którym może być czas, osiągnięcie liczy zadanych iteracji lub zadowalającego wyniku jednego z osobników wewnątrz populacji. Uproszczony schemat działania *EA* przedstawiono na poniższym pseudokodzie [67]:

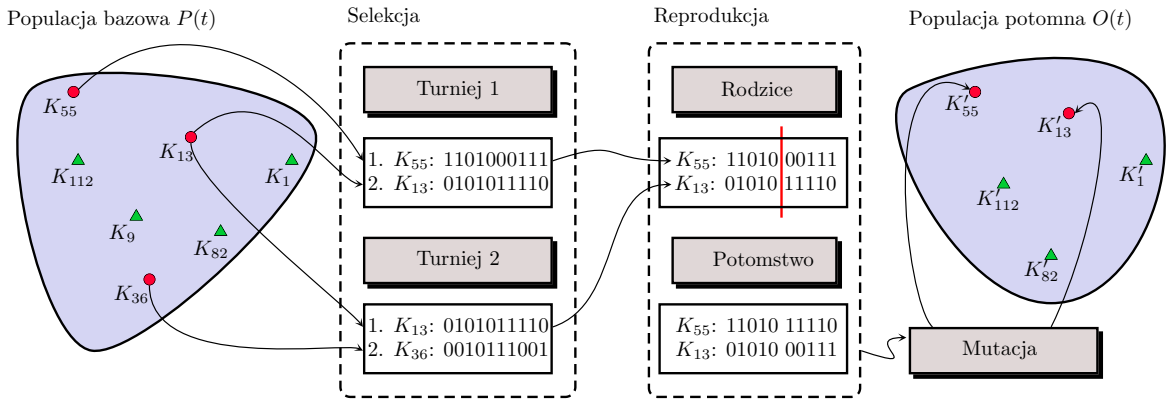
Algorytm 1: Podstawowy algorytm ewolucyjny

```
1  P(0) := zainicjuj_populację_początkową()
2  ocena_osobników(P(0))
3  while t < liczba_iteracji do
4  |   T(t) := selekcja(P(t))
5  |   O(t) := krzyżowanie(T(t))
6  |   O(t) := mutacja(O(t))
7  |   ocena_osobników(O(t))
8  |   P(t + 1) := O(t)
9  |   t := t + 1
10 end
```

Podstawowym założeniem tej techniki jest stała liczba osobników. Część z nich będzie poświęcona na stworzenie nowej populacji, reszta zostanie automatycznie przeniesiona jako nowe pokolenie. Przez kolejne iteracje, pod wpływem działania operatorów genetycznych, osobniki ewoluują, poprawiając jakość swojego przystosowania. Populacja algorytmu podlega ciągłej transformacji. Zdolność *EA* do poprawiania średniej

wartości przystosowania wewnątrz populacji nazywana jest **naciskiem selektywnym** (ang. *Selection Pressure*).

Na rys. 3.1 przedstawiono uproszczony *EA*, realizujący proces ewolucji biologicznej, opierając się na selekcji turniejowej oraz prostym krzyżowaniu jednopunktowym.



Rysunek 3.1: Schemat przykładowego algorytmu ewolucyjnego (genetycznego)

Warto jeszcze wspomnieć, że wszystkie osobniki zostały, w ramach tej rozprawy, oznaczone jako K_i, co symbolizuje poszczególne podklucze dla wybranego algorytmu szyfrującego. Warto wspomnieć, że $i \in \{1, ..., \mu\}$, gdzie μ to liczba wszystkich osobników w populacji.

W kolejnych podrozdziałach zawarto bardziej szczegółowy opis przykładowych operatorów genetycznych wraz z algorytmami przeszukiwania lokalnego.

3.1.1 Selekcja najlepszych osobników

Z najpopularniejszych metod selekcji można wymienić **selekcję proporcjonalną** (ang. *Fitness Proportionate Selection*), zwaną również **metodą koła ruletki** (ang. *Roulette Wheel Selection*). Technika ta polega przyporządkowaniu każdemu osobnikowi K_i pewnego prawdopodobieństwa p_s, określającego szansę jego reprodukcji. Prawdopodobieństwo zostaje wyznaczone na podstawie wyrażenia [65]:

$$p_s(K_i) = \frac{F_f(K_i)}{\sum_{K_j \in P(t)} F_f(K_j)}, \tag{3.1}$$

gdzie:

- F_f - oznacza wartość funkcji przystosowania dla osobnika K_i lub K_j,
- $P(t)$ - aktualna populacja bazowa.

Na podstawie powyższego wzoru można zauważyć, że wysokość prawdopodobieństwa zależy od wartości F_f osobnika [38]. Metoda ta może powodować trudności w zbieżności *EA*, zwłaszcza w przypadku, gdy coraz więcej osobników potomnych uzyskuje zbliżone wartości F_f - intensywność nacisku selektywnego jest coraz mniejsza wraz z kolejnymi iteracjami algorytmu. Ponadto, w przypadku zbyt mocno zróżnicowanej populacji $P(0)$ siła nacisku selektywnego może doprowadzić do przedwczesnej zbieżności *EA* [2].

Inną popularną techniką jest **selekcja rankingowa** (ang. *Rank Based Selection*). W podejściu tym osobniki zostają uszeregowane według swojej wartości F_f. Następnie każdemu z nich zostaje przydzielona ranga, zgodnie z numerem porządkowym, jaki został mu przyporządkowany np. podczas sortowania. W przypadku, w którym kilka osobników charakteryzuje się takim samym przystosowaniem, wszystkie zostaną opisane taką samą rangą. Następnie należy zdefiniować zmienną losową, która wyznacza prawdopodobieństwo reprodukcji dla każdego osobnika na podstawie jego rangi. W literaturze funkcja odpowiadająca za wyznaczenie prawdopodobieństwa p_r wyrażana jest wzorem:

$$p_s(K_i) = a + k(1 - \frac{r(K_i)}{r_{MAX}}), \tag{3.2}$$

gdzie r oznacza rangę danego osobnika, natomiast r_{MAX} to ranga najlepszego osobnika przy czym:

$$r_{MAX} = \max_{K_i \in P(t)} r(K_i). \tag{3.3}$$

Symbole a oraz k opisują dodatkowe parametry sterujące. Ich dobór zrealizowany musi być zgodnie z założonymi poniżej wytycznymi:

$$\sum_{i=1}^{\mu} p_s(K_i) = 1, \tag{3.4}$$

$$0 \leqslant p_s(K_i) \leqslant 1, \tag{3.5}$$

$$\text{jeżeli } r(K_i) \geqslant r(K_j) \text{ to } p_s(K_i) \leqslant p_s(K_j). \tag{3.6}$$

Minusem selekcji rankingowej jest przypisanie osobnikom, o możliwie bardzo dużych wahaniach w wartości F_f, kolejnych rang. Spowoduje to, że rozwiązania te na

pozór będą zachowywać się bardzo podobnie, co nie do końca będzie zgodne z prawdą. Zaletą tego podejścia jest zabezpieczenie populacji przed powstaniem dużej liczby super osobników, charakteryzujących się największą wartością F_f.

Ostatnią omawianą metodą selekcji, istotną w ramach tej rozprawy, jest **selekcja turniejowa** (ang. *Tournament Selection*). W każdej iteracji zostaje wyznaczony dodatkowy podzbiór $Q \subset P(t)$, w ramach którego wybranych będzie q osobników pochodzących z populacji bazowej $P(t)$. Wszyscy kandydaci zostają wylosowani z jednakowym prawdopodobieństwem. Następnie w ramach podzbioru Q zostaje wyłoniony lider turnieju K_i^Q, charakteryzujący się największą wartością F_f [2]:

$$F_f(K_i^Q) \geqslant F_f(K_j) \text{ dla każdego } K_j \in Q. \tag{3.7}$$

Pozostali uczestnicy turnieju będą mogli wziąć udział w kolejnym, aż do wyczerpania populacji $P(t)$. Następnie proces reprodukcji zostaje powtórzony w celu selekcji kolejnego osobnika rodzicielskiego. Dopuszcza się dwie implementacje reprodukcji turniejowej. W pierwszej z nich, lider poprzedniego turnieju K_i^Q może, po raz kolejny, brać udział w procesie reprodukcji - losowanie ze zwracaniem, natomiast w drugim już nie - losowanie bez zwracania. Dla każdej z tych dwóch implementacji można wyznaczyć prawdopodobieństwo reprodukcji p_s, na podstawie wzorów [2]:

- losowanie ze zwracaniem

$$p_s(K_i) = \frac{q}{\mu}\left(\frac{\mu - r(K_i)}{\mu})\right)^{(q-1)}, \tag{3.8}$$

- losowanie bez zwracania

$$p_s(K_i) = \frac{q}{\mu}\prod_{k=1}^{q-1}\frac{\mu - r(K_i) - k}{\mu - k}, \tag{3.9}$$

gdzie r oznacza funkcję przyporządkowującą rangę.

Każdy z powyższych wariantów odpowiada w inny sposób za wartość nacisku selektywnego. Losowanie ze zwracaniem dopuszcza reprodukcję wcześniej wyłonionych osobników, przez to wariant ten wydaje się być mniej restrykcyjny.

3.1.2 Charakterystyka procesu sukcesji

Proces odpowiadający za kreowanie nowej populacji bazowej $P(t+1)$ na podstawie osobników znajdujących się w populacji potomnej $O(t)$ oraz poprzedniej populacji bazowej $P(t)$ nazywany jest **sukcesją** (ang. *Succession*) [2]. Z najpopularniejszych metod można wymienić sukcesję:

- z całkowitym zastępowaniem (ang. *Trivial Succession*),
- z częściowym zastępowaniem (ang. *Overlapping Populations*),
- elitarną (ang. *Elitist Succession*).

W pierwszej z wymienionych metod, nowa populacja $P(t+1)$ powstaje tylko i wyłącznie na podstawie osobników pochodzących z populacji potomnej $O(t)$. Jest to najczęściej wykorzystywany sposób kreowania nowego zbioru rozwiązań. Może to doprowadzić do sytuacji, w której najbardziej przystosowany osobnik populacji $P(t)$ zostanie odrzucony, nawet w przypadku, gdy nie udało się wygenerować ani jednego lepszego potomka wewnątrz populacji $O(t)$. Warto zaznaczyć, że metoda ta nie wprowadza nacisku selektywnego [2].

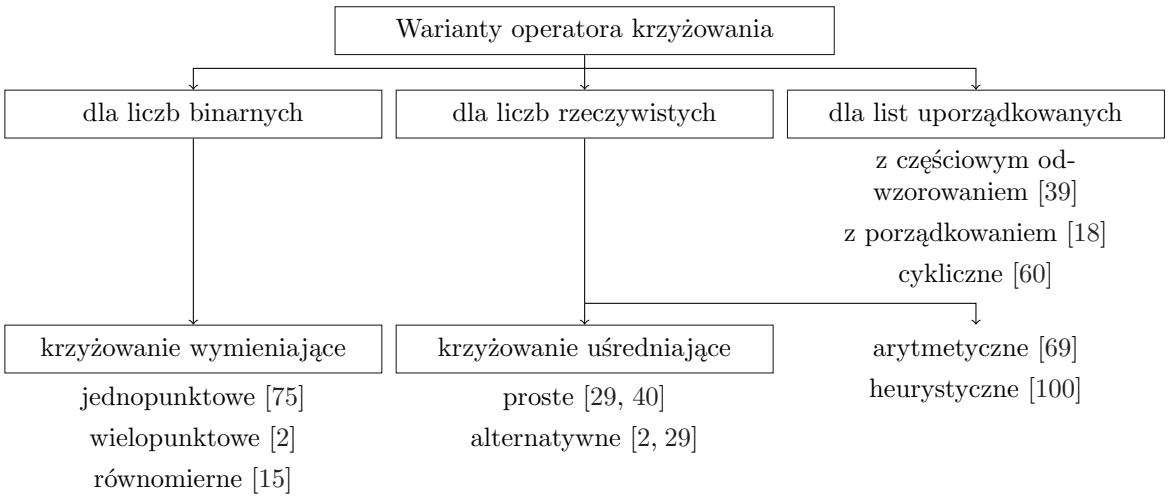
Sukcesja z częściowym zastępowaniem wprowadza pojęcie współczynnika wymiany η. Określa on jaki % osobników przetrwa ze starej populacji. Wartość tego parametru musi zawierać się w przedziale $0 < \eta \leqslant 1$. Wartość 0 w przypadku współczynnika η jest niedopuszczalna, ponieważ sprowadziłoby się to do zastosowania sukcesji z całkowitym zastępowaniem. Nowa populacja potomna zostanie stworzona na podstawie osobników z populacji $O(t)$ oraz ze starej populacji bazowej $P(t)$. Na ogół zakłada się, że populacja $P(t+1)$ będzie zawierać $\eta\mu$ osobników z populacji potomnej. Aby było to możliwe, koniecznym staje się usunięcie części starych osobników z aktualnej populacji bazowej $P(t)$. Liczba rozwiązań w nowej populacji musi zawsze być taka sama. Stosowane są tutaj różnego rodzaju warianty wyboru, od najprostszego losowania, poprzez usuwanie identycznych i zbliżonych do siebie osobników czy odrzucenia najsłabiej przystosowanych rozwiązań [2]. Sukcesja z częściowym zastępowaniem pociąga za sobą tendencję do wpadania algorytmu w maksima lokalne [2].

W przypadku **sukcesji elitarnej** populacja $P(t+1)$ stanowi połączenie osobników populacji potomnej $O(t)$ wraz z najlepszymi osobnikami poprzedniej populacji bazowej $P(t)$. Podejście to gwarantuje zachowanie najlepszego wygenerowanego osobnika z całej iteracji. Na początku wybieranych jest η najbardziej przystosowanych osobników z populacji $P(t)$, a następnie μ najlepszych rozwiązań z populacji $O(t)$. Populacja $P(t+1)$ powstaje wskutek złączenia wszystkich wyselekcjonowanych osobników. Metoda ta, podobnie jak sukcesja z częściowym zastępowaniem, może doprowadzić do szybkiej zbieżności algorytmu poprzez wpadnięcie w maksima lokalne. Im większa wartość parametru η, tym szybciej algorytm osiągnie zbieżność.

3.1.3 Rekombinacja osobników rodzicielskich

Głównym zadaniem operatora rekombinacji, zwanego również operatorem krzyżowania, jest wymiana materiału genetycznego pomiędzy osobnikami rodzicielskimi. W efekcie działania tego procesu otrzymuje się nowe osobniki potomne, stanowiące mieszankę genów, pochodzących od swoich rodziców [40]. Kombinacja wybranych cech pozwala na stworzenie zarówno gorszych, jak i lepszych osobników. Dzięki temu operatorowi kolejne pokolenia stają się coraz bardziej dostosowane do warunków środowiska, w którym się znajdują. Jak wspomniano wcześniej, potencjalni rodzice wybierani są drogą selekcji. W klasycznym *GA* najczęściej przyjmuje się zasadę, że w operatorze rekombinacji bierze udział dwoje osobników rodzicielskich (K_i, K_j). Wynikiem operatora krzyżowania, w większości implementacji, są dwa osobniki potomne (K'_i, K'_j). Nowo powstałe osobniki potomne zastępują w kolejnej iteracji swoich rodziców. Krzyżowanie zachodzi z pewnym przyjętym prawdopodobieństwem p_c, które przyjmuje się jako wysokie wartości. Na ogół jest to wartość z przedziału: $0,7 \leqslant p_c \leqslant 0,95$.

Ze względu na potężną liczbę wszelkiego rodzaju wariantów rekombinacji wprowadzono podział, mający na uwadze ich zastosowanie. Na rys. 3.2 przedstawiono przykładowe implementacje operatora krzyżowania:



Rysunek 3.2: Przykładowe warianty operatora krzyżowania

Z najpopularniejszych metod rekombinacji można bez wątpienia wymienić wszelkie warianty krzyżowania wymieniającego. Polegają one na stworzeniu potomków na podstawie wymiany wybranej części genotypu pomiędzy osobnikami rodzicielskimi. Najbardziej znanymi implementacjami, tej podgrupy rekombinacji, jest **krzyżowanie jednopunktowe** (ang. *1-Point Crossover*) oraz **dwupunktowe** (ang. *2-Point Crossover*).

W przypadku krzyżowania jednopunktowego zaczyna się od wyboru punktu przecięcia rodziców. Punkt ten musi zawierać się w zbiorze $p \in \{1, 2, ..., \mu - 1\}$, gdzie μ oznacza długość osobnika. Punkt przecięcia najczęściej wybierany jest losowo. Następnie osobniki rodzicielskie K_i oraz K_j zostają przedzielone na dwie części. Lewa część rodzica K_i sklejana jest z prawą częścią rodzica K_j, natomiast lewa K_j z prawą K_i, co można zapisać jako:

$$K'_{i_k} = \begin{cases} K_{i_k}, & \text{gdy } k \leqslant p, \\ K_{j_k}, & \text{gdy } k > p, \end{cases} \quad \text{oraz} \quad K'_{j_k} = \begin{cases} K_{i_k}, & \text{gdy } k > p, \\ K_{j_k}, & \text{gdy } k \leqslant p, \end{cases} \tag{3.10}$$

gdzie $k \in \{1, 2, ..., \mu\}$, natomiast K'_{i_k} oraz K'_{j_k} to poszczególne bity osobników potomnych. Bez względu na to, jaki punkt p zostanie wylosowany, przynajmniej jeden gen ulegnie wymianie [2].

W krzyżowaniu dwupunktowym rodziców dzieli się na trzy części, aczkolwiek procesowi wymiany podlega tylko środkowy fragment genotypu. Na początku algorytmu losuje się dwa punkty p_1 i p_2, gdzie, podobnie jak w przypadku krzyżowania jednopunktowego, $p_1 \in \{1, 2, ..., \mu - 1\}$ i $p_2 \in \{1, 2, ..., \mu - 1\}$ oraz $p_1 \leqslant p_2$. Wariant krzyżowania dwupunktowego można opisać jako:

$$K'_{i_k} = \begin{cases} K_{j_k}, \text{ gdy } p_1 \leqslant k \leqslant p_2, \\ K_{i_k} \text{ w przeciwnym wypadku,} \end{cases} \quad \text{oraz} \quad K'_{j_k} = \begin{cases} K_{i_k}, \text{ gdy } p_1 \leqslant k \leqslant p_2, \\ K_{j_k} \text{ w przeciwnym wypadku.} \end{cases} \tag{3.11}$$

Warto wspomnieć, że w przypadku tego typu rekombinacji, dopuszcza się sytuację w której $p_1 = p_2$. Stosowany jest wtedy wariant krzyżowania jednopunktowego.

Innym popularnym sposobem krzyżowania wymieniającego jest bez wątpienia operator **rekombinacji równomiernej** (ang. *Uniform Crossover*). W tym przypadku zakłada się dodatkowy parametr p_e, spełniający warunek $0 \leqslant p_e \leqslant 1$, gdzie p_e oznacza prawdopodobieństwo zamiany wybranego genu. Następnie, dla każdego fragmentu osobnika rodzicielskiego dokonuje się losowania liczby zmiennoprzecinkowej $\alpha \in [0, 1]$. Jeżeli wartość α będzie większa od założonego prawdopodobieństwa p_e, wówczas kopiuje się gen osobnika K_j, w przeciwnym wypadku pozostawia się oryginalny fragment rodzica K_i, co można zapisać jako:

$$K'_{i_k} = \begin{cases} K_{j_k}, \text{ gdy } \alpha > p_e, \\ K_{i_k} \text{ w przeciwnym wypadku,} \end{cases} \quad \text{oraz} \quad K'_{j_k} = \begin{cases} K_{i_k}, \text{ gdy } \alpha > p_e, \\ K_{j_k} \text{ w przeciwnym wypadku.} \end{cases} \tag{3.12}$$

Kolejnym z wymienionych na rys. 3.2 operatorów rekombinacji jest wariant **krzyżowania uśredniającego** (ang. *Average Crossover*). Wszelkie warianty tego operatora kierowane są dla kodowania rzeczywisto-liczbowego - operują one na wartościach samych genów każdego osobnika rodzicielskiego. W podstawowym wariancie krzyżowania uśredniającego dokonuje się losowania liczby zmiennoprzecinkowej α. Następnie postępuje się zgodnie z poniższym zapisem:

$$K'_i = K_i + \alpha(K_j - K_i) \quad \text{oraz} \qquad K'_j = K_j + K_i - K'_i. \tag{3.13}$$

W literaturze wspomina się dodatkowo o wersji alternatywnej [2], operującej na poszczególnych genach osobników rodzicielskich:

$$K'_{i_k} = K_{i_k} + \alpha(K_{j_k} - K_{i_k}) \quad \text{oraz} \qquad K'_{j_k} = K_{j_k} + K_{i_k} - K'_{i_k}. \tag{3.14}$$

Główną różnicą pomiędzy wariantami krzyżowania wymieniającego - a uśredniającego jest modyfikacja wartości genów. W przypadku metod wymieniających wszystkie oddziedziczone geny mają taką samą postać, jak u osobnika rodzicielskiego. Podczas krzyżowania uśredniającego każdy z genów zostaje w pewien sposób zmodyfikowany.

Z pozostałych wariantów rekombinacji dla liczb rzeczywistych można wymienić **krzyżowanie arytmetyczne** (ang. *Arithmetical Crossover*) oraz **heurystyczne** (ang. *Heuristic Crossover*). W przypadku krzyżowania arytmetycznego, po raz kolejny wprowadza się liczbę $\alpha \in [0, 1]$, a następnie, dla każdego genu, postępuje się zgodnie z poniższym wyrażeniem [69]:

$$K'_{i_k} = \alpha K_{i_k} + (1 - \alpha) K j_k \quad \text{oraz} \qquad K'_{j_k} = \alpha K_{j_k} + (1 - \alpha) K i_k. \tag{3.15}$$

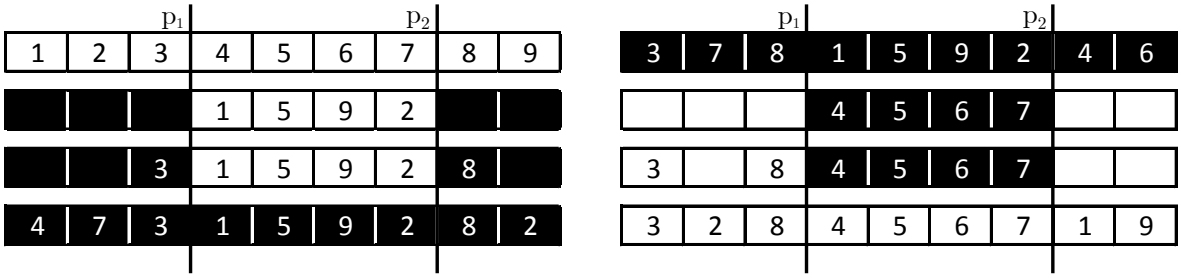
W przeciwieństwie do wcześniej zaprezentowanych operatorów rekombinacji w krzyżowaniu heurystycznym tworzony jest tylko jeden potomek na podstawie dwóch osobników rodzicielskich. Proces rekombinacji rozpoczyna się od wylosowania liczby zmiennoprzecinkowej $\alpha \in [0, 1]$. Następnie pod uwagę brana jest aktualna wartość F_f każdego z rodziców. Osobnik potomny wyznaczany jest na podstawie następującego wyrażenia:

$$K_i' = \begin{cases} \alpha(K_j - K_i) + K_j, & \text{gdy } F_f(K_i) \geqslant F_f(K_j), \\ \alpha(K_i - K_j) + K_i, & \text{gdy } F_f(K_i) < F_f(K_j). \end{cases} \tag{3.16}$$

W ramach ostatniej grupy wybranych operatorów rekombinacji, należy wymienić operatory krzyżowania operujące na listach uporządkowanych. Przykładem ich zastosowania może być problem komiwojażera. Z najbardziej znanych operatorów rekombinacji można wymienić krzyżowanie:

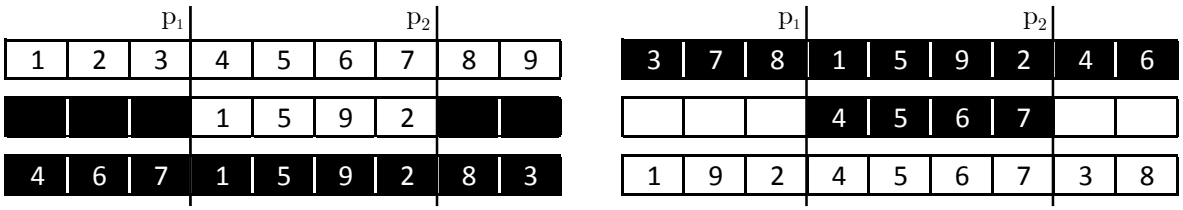
- z częściowym odwzorowaniem (ang. *Partially Matched Crossover, PMX*),
- z porządkowaniem (ang. *Ordered Crossover, OX*),
- cykliczne (ang. *Cycle Crossover, CX*).

Na rys. 3.3 przedstawiono proces **krzyżowania z częściowym odwzorowaniem**. Na początku, w sposób losowy, wybierane są dwa punkty przecięć, tak aby $p_1 < p_2$. Następnie wykonywana jest transpozycja genów znajdujących się pomiędzy punktami p_1 i p_2 z rodziców do osobników potomnych. W ten sposób powstaje tabela odwzorowań, którą na podstawie przedstawionego rysunku wygląda następująco: $[4 \leftrightarrow 1, 5 \leftrightarrow 5, 6 \leftrightarrow 9, 7 \leftrightarrow 2]$. Kolejnym krokiem jest uzupełnienie wybranych genów z przeciwległego rodzica w taki sposób, aby nie powstały jakiekolwiek duplikaty. Na samym końcu, na podstawie tabeli odwzorowań, uzupełniane są pozostałe fragmenty osobników potomnych.



Rysunek 3.3: Przykład krzyżowania z częściowym odwzorowaniem

Przykład **rekombinacji z porządkowaniem** został przedstawiony na rys. 3.4. Podobnie jak miało to miejsce w krzyżowaniu *PMX*, proces rekombinacji rozpoczyna się od wylosowania dwóch punktów przecięć p_1 oraz p_2. Następnie, wyznaczony materiał genetyczny, zostaje przeniesiony z osobników rodzicielskich do potomków. Od tego czasu uruchamiany jest proces cyklicznego kopiowania genów, z przeciwległego rodzica, rozpoczynając od indeksu $p_2 + 1$ aż do pozycji p_2, z wykluczeniem tych genów, które już aktualnie w osobniku potomnym występują.

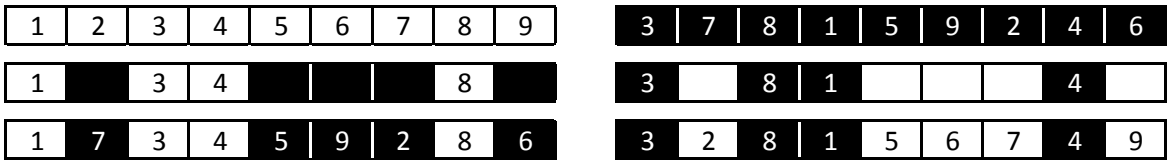


Rysunek 3.4: Przykład krzyżowania z porządkowaniem

W przypadku **krzyżowania cyklicznego** (rys. 3.5), każdy element permutacji potomnej przyjmuje taką samą pozycję, jak w jednej z dwóch permutacji rodzicielskich. W ramach tego typu rekombinacji wybiera się dodatkowy podzbiór pozycji I, składający się z takich samym elementów jak w każdym z rodziców:

$$\{K_{i_k} : k \in I\} = \{K_{j_k} : k \in I\}. \tag{3.17}$$

Elementy na pozycjach ze zbioru I pozostawiane są bez zmian, natomiast pozostałe elementy zamieniane są miejscami pomiędzy osobnikami rodzicielskimi.



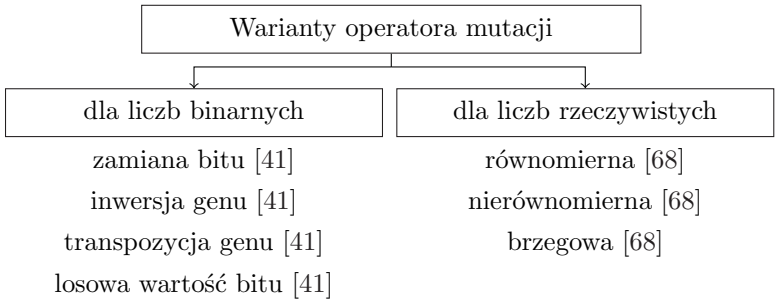
Rysunek 3.5: Przykład krzyżowania cyklicznego

3.1.4 Rola operatora mutacji

Operator mutacji odgrywa drugoplanową rolę w porównaniu do opisanego operatora rekombinacji. W przeciwieństwie, do często występującego krzyżowania, w analogii do natury, zjawisko mutacji zachodzi niezwykle rzadko. Zadaniem tego operatora jest wprowadzenie pewnego elementu losowości wewnątrz osobnika potomnego. Najczęściej jest to jakaś niewielka zmiana, perturbacja, wewnątrz genotypu, mająca raczej mniejsze znaczenie dla osobnika. Wprowadzenie tak niespodziewanego zachowania ma pomóc algorytmowi na wyskoczenie z aktualnie eksploatowanej przestrzeni rozwiązań w celu znalezienia bardziej satysfakcjonującego rozwiązania. Prawdopodobieństwo zaistnienia zjawiska mutacji przyjmuje się zwykle rzędu $0 < p_m \leqslant 0,1$.

Podobnie jak to miało miejsce w przypadku krzyżowania, ze względu na pokaźną liczbę różnego rodzaju operatorów mutacji, zdecydowano się na wybór kilku przykładowych implementacji z uwzględnieniem typu przestrzeni rozwiązań, w której mogą

zostać one zastosowane. Pominięto zastosowanie dla list uporządkowanych, w większości przypadków opisane operatory mutacji mogą zostać dostosowane do tego typu problemów. Na rys. 3.6 przedstawiono przykładowe implementacje operatora mutacji:



Rysunek 3.6: Przykładowe warianty operatora mutacji

Najczęściej spotykanym wariantem mutacji, dla binarnej przestrzeni rozwiązań, jest **zamiana pojedynczego bitu na przeciwny** (ang. *Bit Flip*). Dla każdego elementu wyznaczana jest liczba zmiennoprzecinkowa $\alpha \in [0, 1]$. Następnie, jeżeli spełniony zostanie warunek $\alpha \leqslant p_m$ liczba 0 zostaje zamieniona na 1 lub adekwatnie 1 na 0. Działanie omawianego operatora mutacji można sprowadzić do następującej formuły:

$$K'_{i_k} = \begin{cases} \sim K_{i_k}, & \text{gdy } \alpha \leqslant p_m, \\ K_{i_k}, & \text{gdy } \alpha > p_m. \end{cases} \tag{3.18}$$

Mutacja poprzez **inwersję** (ang. *Gene Inversion*) polega na wyznaczeniu wewnątrz osobnika podciągu genotypu na podstawie jednego lub dwóch punktów inwersji p_1 oraz p_2. W kolejnym etapie wszystkie elementy podciągu poddawane są prostemu odwróceniu:

$$K'_{i_k} = \begin{cases} \sim K_{i_k}, & \text{gdy } p_1 \leqslant k \leqslant p_2, \\ K_{i_k} & \text{w przeciwnym wypadku.} \end{cases} \tag{3.19}$$

W przypadku **mutacji z transpozycją genu** (ang. *Gene Transposition*) losowo wyznaczany jest wskaźnik genu $\alpha \in \{1, ..., \mu\}$ oraz wskaźnik transpozycji $\beta \in \{1, ..., \mu\}$, gdzie μ oznacza liczbę elementów wewnątrz chromosomu. Następnie wylosowany fragment, o pozycji α, zostaje przeniesiony na pozycje wyznaczoną przez wskaźnik transpozycji β:

$$K'_{i_k} = \begin{cases} K_{i_k}, & \text{gdy } k \leqslant \beta \text{ lub } k > (\beta + 1), \\ K_{i_\alpha}, & \text{gdy } k = \beta + 1. \end{cases} \tag{3.20}$$

Ostatnią wspomnianą mutacją, przeznaczoną dla binarnej przestrzeni rozwiązań, jest **mutacja z losową wartością bitu** (ang. *Random Bit Value*). Polega ona na podmianie bieżącej postaci na nową wylosowaną wartość. Zakłada się pewną stałą, decydującą o tym, którą wartość należy wybrać, a następnie losowana jest liczba zmiennoprzecinkowa $\alpha \in [0, 1]$. Jeżeli wylosowana liczba jest większa od założonej stałej przyjmuje się wartość 1, w przeciwnym wypadku 0.

Jeżeli chodzi o zastosowanie mutacji dla liczb rzeczywistych do najpopularniejszych operatorów można zaliczyć mutację:

- równomierną (ang. *Uniform Mutation*) [67],
- nierównomierną (ang. *Non-uniform Mutation*) [67],
- brzegową (ang. *Boundary Mutation*) [67].

W operatorze **mutacji równomiernej** każdy element osobnika K_i charakteryzuje się równomiernym prawdopodobieństwem zajścia procesu mutacji [67]. Rezultatem pojedynczego zastosowania tego typu mutacji jest zamiana losowo wybranego fragmentu osobnika o indeksie $k \in [1, ..., \mu]$ [67]. Wylosowany element zostaje zastąpiony nowo wylosowaną wartością K'_{i_k} z przedziału $< left_k, right_k >$, gdzie $left_k$ oraz $right_k$ oznaczają kolejno dolny i górny kraniec dopuszczalnej dziedziny K'_{i_k}. Mutacja równomierna może wprowadzać bardzo duże modyfikacje wewnątrz osobników. Dlatego operator ten znajduje największe zastosowanie w początkowej fazie działania *EA*, gdzie szerokie przeszukiwanie przestrzeni rozwiązań jest najbardziej pożądane [10].

Mutacja nierównomierna jest tzw. operatorem ze strojeniem [10], co oznacza, że jego działanie modyfikowane jest w kolejnych iteracjach *MA*. Przed przystąpieniem do mutacji zostaje wylosowana nieujemna liczba całkowita $\alpha \in \{0, 1\}$. Wybrany fragment osobnika K_i zostaje poddany mutacji na podstawie poniższego wyrażenia:

$$K'_{i_k} = \begin{cases} K_{i_k} + \Delta(t, right_k - K_{i_k}), & \text{gdy } \alpha = 0, \\ K_{i_k} - \Delta(t, K_{i_k} - left_k), & \text{gdy } \alpha = 1, \end{cases} \tag{3.21}$$

gdzie:

- t - odpowiada numerowi aktualnej iteracji algorytmu,
- $\Delta(t, x)$ - oznacza pewną funkcję przyjmującą wartości z przedziału $[0, x]$.

Prawdopodobieństwo tego, że wartość funkcji $\Delta(t, x)$ będzie bliska zeru jest większe wraz ze wzrostem t [67]. Dzięki temu *EA* eksploruje początkowo przestrzeń rozwiązań

w sposób równomierny, a następnie, w kolejnych iteracjach algorytmu, gdy wartość t staje się większa, skupia się na przeszukiwaniu lokalnym [67]. Funkcja $\Delta(t, x)$ może być zdefiniowana za pomocą poniższego wyrażenia [67]:

$$\Delta(t, x) = x(1 - r^{(1-\frac{1}{T})b}), \tag{3.22}$$

gdzie:

- r stanowi liczbę losową, taką że $r \in [0, 1]$, T oznacza maksymalną liczbę iteracji,
- T oznacza maksymalną liczbę iteracji,
- b jest parametrem określającym stopień niejednorodności - stopień zależności od numeru iteracji [67].

Zastosowanie **mutacji brzegowej** jest najbardziej efektywne w przypadku, gdy rozwiązanie optymalne znajduje się na brzegu, lub w jego pobliżu, obszaru przestrzeni rozwiązań [10]. Operator ten stanowi odmianę opisanej wcześniej mutacji równomiernej z tą różnicą, że element K'_{i_k} zawsze przyjmie, z jednakowym prawdopodobieństwem, jedną z wartości zbioru $\{left_k, right_k\}$, co można zapisać jako:

$$K'_{i_k} = \begin{cases} left_k, & \text{gdy } \alpha = 0, \\ right_k, & \text{gdy } \alpha = 1. \end{cases} \tag{3.23}$$

Mutacja brzegowa prowadzi do zbyt wczesnej zbieżności populacji, co w niektórych problemach może być korzystne.

4

Algorytmy memetyczne

W literaturze można spotkać się z wieloma przykładami hybrydyzacji *EA*, w których stwierdzono, że włączenie do algorytmu, w dowolnej formie, dodatkowej wiedzy na temat rozpatrywanego problemu, pozwala na uzyskanie znacznie efektywniejszych rozwiązań [65]. Tego typu kombinacje można uzyskać na wiele różnych sposobów, np. poprzez połączenie *EA* z innymi technikami heurystycznymi. Najbardziej obiecująca okazała się być hybrydyzacja z wszelkiego rodzaju metodami **optymalizacji lokalnej** [65].

Połączenie tych dwóch podejść pozwala m.in. na ewolucyjne, a nie losowe, wygenerowanie punktów startowych dla procesu przeszukiwania lokalnego [2]. Warto również wspomnieć, że proces eksploatacji wcale nie musi być dokładny. Można wykonać tylko kilka pierwszych ruchów w celu przyśpieszenia przybliżenia algorytmu do poszukiwanego ekstremum [2].

Wynikiem hybrydyzacji *EA* są dobrze znane dzisiaj algorytmy memetyczne. Jak wspomniano wcześniej stanowią one rozwiązania hybrydowe, łączące w sobie wszelkiego rodzaju *EA*, wykorzystywane do eksploracji aktualnej przestrzeni rozwiązań, wraz z algorytmami przeszukiwania lokalnego - odpowiadających za proces eksploatacji wybranych podprzestrzeni rozwiązań.

4.1 Uogólniony algorytm memetyczny

Algorytmy memetyczne zostały opracowane przez Pablo Moscato w 1989 roku [71]. Podobnie, jak to miało miejsce w przypadku standardowego algorytmu ewolucyjnego, *MA* operują na całym zbiorze rozwiązań zwanych populacją. Ponadto można się tu-

taj również spotkać z takimi samymi procesami, jak w przypadku najprostszego *EA*. Mowa o ocenie przydatności każdego rozwiązania przy pomocy funkcji przystosowania, selekcja najlepiej przystosowanych osobników, krzyżowaniu osobników rodzicielskich czy potencjalnej perturbacji (mutacja) nowo powstałych osobników potomnych [71].

Warto zaznaczyć, że *MA* posiadają dodatkowe informacje, nabyte podczas procesu ewolucji, co pozwala na uzyskanie znacznie lepszych wyników niż te, które zostały otrzymane przy użyciu klasycznych *EA* [73]. Właśnie ta dodatkowa wiedza pozwala na eksploatacje wybranych podgrup przestrzeni, dzięki czemu możliwe jest odnalezienie jeszcze bardziej satysfakcjonującego rozwiązania.

Dodatkowym krokiem, uruchamianym w ramach *MA*, jest obecność lokalnego przeszukiwania. W większości podejść proces ten zostaje umiejscowiony jednorazowo po operatorze krzyżowania lub mutacji. Zdarzają się również implementacje, w których proces lokalnego przeszukiwania uruchamiany jest kilkukrotnie w różnych miejscach *MA*. Eksploatacja realizowana może być za pomocą różnych algorytmów przeszukiwania lokalnego. Na poniższym pseudokodzie przedstawiono uproszczony schemat *MA* [73]:

Algorytm 2: Podstawowy algorytm memetyczny

```
1  P(0) := zainicjuj_populację_początkową()
2  ocena_osobników(P(0))
3  while t < liczba_iteracji do
4  |   T(t) := selekcja(P(t))
5  |   O(t) := krzyżowanie(T(t))
6  |   O(t) := mutacja(O(t))
7  |   ocena_osobników(O(t))
8  |   O(t) := przeszukiwanie_lokalne(O(t))
9  |   ocena_osobników(O(t))
10 |   P(t + 1) := O(t)
11 |   t := t + 1
12 end
```

W przeciwieństwie do standardowych *EA*, w których to populacja $P(0)$ zazwyczaj generowana jest w sposób losowy, *MA*, już na samym początku, starają się wygenerować populację składającą się z rozwiązań o jak najlepszej jakości. Najczęściej realizowane to jest poprzez wstrzyknięcie najbardziej przystosowanych osobników przy pomocy dodatkowego kroku odpowiadającego za realizację procesu przeszukiwania lokalnego

[73]. Optymalizacja lokalna znajduje również zastosowanie po zastosowaniu operatora krzyżowania lub/i mutacji.

Podczas procesu optymalizacji lokalnej osobnik otrzymuje nowe cechy, które nie wynikają ani z procesu mutacji, ani z operatora krzyżowania. Zostały one natomiast wyuczone podczas życia osobnika podczas trwania procesu ewolucji [65]. Wskutek takiego działania otrzymuje się nowe rozwiązanie K_i' z bardziej korzystną wartością F_f niż poprzednio.

W literaturze można spotkać się z różnymi podejściami zapamiętywania tak wygenerowanych osobników. Najprostszym wariantem jest zapamiętanie w pamięci nowego osobnika, wraz z wartością jego przystosowania, bez wprowadzania go do kolejnej populacji bazowej [2]. Oczywiście, zapamiętane rozwiązania cały czas są brane pod uwagę w rezultacie zwracanym przez algorytm.

Kolejnym sposobem jest zastąpienie oryginalnej wartości $F_f(K_i)$ wartością $F_f(K_i')$. Mowa tutaj o tzw. **efekcie Baldwina** (ang. *Baldwin Effect*) [30]. Technika ta sprowadza się na przypisaniu osobnikowi samej wartości funkcji przystosowania, wyznaczonej w skutek działania procesu przeszukiwania lokalnego [2,30]. Wszystkie wyuczone cechy zostają zapomniane - nie mają one żadnego wpływu na postać genotypu. Warto wspomnieć, że wszystkie osobniki, znajdujące się w pobliżu jednego maksimum lokalnego, prawdopodobnie odznaczać się będą taką samą wartością F_f [2].

Ostatnim spotykanym wariantem jest zastąpienie oryginalnego osobnika K_i nowym, wygenerowanym podczas procesu optymalizacji lokalnej, rozwiązaniem K_i'. W tym przypadku można mówić o **ewolucji Lamarkowskiej** (ang. *Lamarckian Evolution*) [98]. Nie tylko zapamiętywany jest nowy osobnik, ale również wartość jego przystosowania. Można powiedzieć, w przeciwieństwie do efektu Baldwina, że wszystkie wyuczone w procesie eksploatacji cechy mają wpływ na kształt genotypu nowego rozwiązania [98]. Największą wadą tego podejścia jest podatność algorytmu na maksima lokalne [2].

Bardzo ważnym czynnikiem jest dobór odpowiedniego algorytmu przeszukiwania lokalnego. Mogą to być gotowe techniki, takie jak: *HC*, *SA* czy *TS*, lub całkowicie nowe metody, dedykowane aktualnie rozwiązywalnemu zagadnieniu. Gdy już odpowiedni algorytm przeszukiwania lokalnego zostanie wybrany należy zastanowić się jak często i gdzie powinien zostać wykorzystany, jak wybierać rozwiązania, które będą ulegać lokalnej poprawie oraz jak długo powinny trwać tego typu poprawy. Niewłaściwy dobór tego typu parametrów może negatywnie wpłynąć na jakość wygenerowanych rozwiązań.

Należy również zaznaczyć, że wprowadzenie dodatkowego operatora przeszukiwania

lokalnego posiada również swoje minusy. Wiąże się to z powstaniem dodatkowego narzutu związany z preferowaniem obszarów przyciągania ekstremów lokalnych, co stanowi dodatkowe źródło nacisku selektywnego [2]. Koniecznym staje się również wprowadzenie dodatkowych operatorów odpowiadających za zwiększoną różnorodność populacji.

4.2 Metody przeszukiwania lokalnego

Większość metod przeszukiwania lokalnego opiera swoje działanie na iteratywnym przeszukiwaniu zbioru rozwiązań poprzez analizę swojego sąsiedztwa $\mathcal{N}$. Dowolne rozwiązanie nazywane jest sąsiadem, tylko i wyłącznie w przypadku, gdy może zostać ono osiągnięte przy pomocy dowolnego pojedynczego kroku. Przez krok zdefiniować można procedurę przejścia z jednego rozwiązania K_i do innego K_j. Takim przykładowym krokiem może być negacja jednego bitu w chromosomie [65] lub zamiana genów miejscami. Zbiór sąsiadów dla rozwiązania K_i można zdefiniować jako:

$$\mathcal{N}(K_i) \subseteq \mathcal{X}, \tag{4.1}$$

gdzie $\mathcal{X}$ oznacza przestrzeń wszystkich możliwych rozwiązań.

Sąsiedztwo $\mathcal{N}$ będzie można nazwać dobrym, jeżeli przynajmniej jedno rozwiązanie będzie różne od aktualnego [55]. Ponadto $\mathcal{N}$ nie może zawierać całego $\mathcal{X}$, ponieważ byłoby ono zbyt dokładne [55]. Innym istotnym warunkiem sąsiedztwa staje się również niezależność wyboru punktu inicjalnego algorytmu, tak aby możliwe było osiągnięcie każdego rozwiązania przestrzeni $\mathcal{X}$ [55]. Do tego wszyscy sąsiedzi, znajdujący się w zbiorze $\mathcal{N}$, powinni być podobni do aktualnie przechowywanego rozwiązania, dzięki czemu zapamiętanie nowego lidera nie będzie wymagało stworzenia kolejnego rozwiązania [55].

Z najbardziej znanych metody przeszukiwania lokalnego można wymienić algorytmy:

- wspinaczki (ang. *Hill Climbing*),
- symulowanego wyżarzania (ang. *Simulated Annealing*),
- przeszukiwania z listą Tabu (ang. *Tabu Search*), które opisano w kolejnej części rozprawy.

4.2.1 Algorytm wspinaczki

Metoda wspinaczki stanowi jedną z najprostszych algorytmów przeszukiwania lokalnego. Proces rozpoczyna się od wyboru losowego punktu z przestrzeni rozwiązań.

Następnie wylosowane rozwiązanie zostaje poddane ocenie za pomocą F_f. Kolejnym etapem algorytmu jest przegląd rozwiązań sąsiednich. W większości implementacji dokonuje się przeglądu zupełnego wszystkich sąsiadów. Może jednak zdarzyć się sytuacja w której liczba sąsiadów jest zbyt duża, aby sprawdzić ich wszystkich. Wykonuje się wtedy skończoną liczbę losowań t_{MAX} rozwiązań sąsiednich. Jeżeli okaże się, że algorytm napotka na lepsze od aktualnie zapamiętanego rozwiązania, proces przeszukiwania zostaje wstrzymany, a następnie poprzednie rozwiązanie zostanie nim zastąpione. Następnie proces analizy zostaje wznowiony, tym razem dla rozwiązań sąsiednich nowego lidera [43]. Algorytm powtarzany jest do momentu, dopóki udaje się znaleźć bardziej satysfakcjonujące rozwiązanie. Pseudokod algorytmu wspinaczki z losowaniem został przedstawiony poniżej:

Algorytm 3: Algorytm wspinaczki

```
1  K_0 := generuj_losowo(X)
2  lider := K_0
3  for t := 0 to t_MAX do
4  |   k := generuj_losowo(N(lider))
5  |   if F_f(k) ⩾ F_f(lider)  oraz  k ≠ poprzedni_lider then
6  |   |   poprzedni_lider := lider
7  |   |   lider := k
8  |   |   t := 0
9  |   end
10 end
```

Skuteczność tego podejścia w dużej mierze zależy od wyboru sąsiada oraz punktu startowego. Zaletą algorytmu jest dosyć prosta implementacja oraz wydajność.

Podążanie w kierunku najlepszego rozwiązania może wydawać się najlepszą drogą. Jest to jednak błędne myślenie. Otrzymane rozwiązanie może stanowić lokalne maksimum, które będzie znacznie słabsze niż rozwiązanie globalne - poza tym, w tej metodzie, nie ma możliwości opuszczenia tego rodzaju maksimum. Wyszukanie lepszego rozwiązania możliwe jest tylko po kolejnym uruchomieniu algorytmu - punkt startowy zostanie wybrany na nowo [104]. Ponadto algorytm może zmuszony być do sprawdzenia wszystkich rozwiązań znajdujących się w zbiorze $\mathcal{N}$, co w niektórych przypadkach może być zbyt kosztowne - stąd pomysł losowego wyboru kilku rozwiązań sąsiednich. Warto również zauważyć, że tego typu algorytm nie gromadzi żadnych informacji na temat wcześniej sprawdzonych rozwiązań, które mogłyby okazać się użyteczne [104].

Jedną z najbardziej znanych modyfikacji algorytmu wspinaczki jest **metoda najszybszego wzrostu** (ang. *Steepest Ascent*) [1]. W przeciwieństwie do poprzedniego podejścia, w którym proces przeszukiwania zostaje wstrzymany tylko, gdy napotkano na pierwsze lepsze rozwiązanie, tutaj sprawdzana jest cały zbiór $\mathcal{N}(K_i)$. W przypadku wariantu z losowaniem proces przeszukiwania trwa do momentu osiągnięcia maksymalnej liczby losowań t_{MAX} [1]. Następnie z wygenerowanego podzbioru rozwiązań lepszych od aktualnego wybierany jest najbardziej przystosowany osobnik. Podobnie, jak to miało miejsce w poprzednim podejściu, procedura przeszukiwania powtarzana jest dla nowo wyłonionego lidera aż do osiągnięcia najlepszego stanu końcowego.

4.2.2 Symulowane wyżarzanie

Algorytm **symulowanego wyżarzania** (ang. *Simulated Annealing, SA*) zapoczątkowany został przez Kirkpatricka, Gelatta oraz Vecchiego w 1983 roku [32]. Technika ta inspirowana jest zjawiskami fizycznymi zachodzącymi w procesie obróbki cieplnej i ochładzania metali. Algorytm rezygnuje z metody przeglądu zupełnego $\mathcal{N}(K_i)$, każde kolejne rozwiązanie wybierane jest losowo - proces ten powtarzany jest L razy. Ponadto algorytm dopuszcza przejścia z rozwiązania lepszego do gorszego. Po wylosowaniu punktu o mniejszej wartości F_f, na podstawie funkcji prawdopodobieństwa decyduje się, czy gorsze rozwiązanie zostanie zaakceptowane. Wartość funkcji prawdopodobieństwa jest większa im wyższa będzie aktualna temperatura T. Oznacza to, że na początku algorytm będzie skłony do akceptacji większej liczby gorszych rozwiązań. Dzięki temu podejściu możliwe staje się uniezależnienie pracy *SA* od punktu startowego. Wraz z upływem czasu prawdopodobieństwo przyjęcia gorszego rozwiązania będzie mniejsze - temperatura będzie stopniowo wygaszana.

Na początku algorytmu ustalana jest temperatura początkowa T_0 i końcowa T_{MIN}. W kolejnych iteracjach T obniżana jest o stałą $\alpha \in [0, 1]$ aż do osiągnięcia T_{MIN}. Na każdym etapie analizowane jest L losowo wybranych rozwiązań sąsiednich. Gdy nowy punkt okaże się być lepszy od poprzedniego, zostaje on automatycznie nim zastąpiony. Pseudokod *SA*, z zapamiętywaniem najlepszego rozwiązania, przedstawiono poniżej:

Algorytm 4: Algorytm symulowanego wyżarzania

```
1  T := T_0
2  K_0 := generuj_losowo(X)
3  lider := K_0
4  repeat
5      for i := 0 to L do
6          k := generuj_losowo(N(K_0))
7          if generuj_losowo([0,1]) < Pr(k, K_0) then
8              K_0 := k
9              if F_f(K_0) > F_f(lider) then
10                 lider := K_0
11             end
12         end
13     end
14     T = T · α
15 until T > T_MIN
```

Funkcja prawdopodobieństwa, zezwalającą na akceptację gorszej jakości rozwiązań, została zdefiniowana za pomocą następującego wyrażenia:

$$Pr(x, y) = \exp\left(\frac{-\Delta F}{k_B \cdot T}\right), \tag{4.2}$$

gdzie:

- k_B oznacza stałą Boltzmanna, w przybliżeniu równą $k_B \approx 1,38$,
- ΔF wyznaczana jest na podstawie następującego wyrażenia: $F_f(x) - F_f(y)$.

W niektórych implementacjach funkcja prawdopodobieństwa wyznaczana jest z pominięciem stałej k_B - mówi się wtedy o sztucznym symulowanym wyżarzaniu.

Od samego początku *SA* nastawione jest na eksplorację całej przestrzeni rozwiązań. W późniejszej fazie algorytm coraz bardziej przesuwa się w stronę procesu eksploatacji. *SA* przestaje akceptować gorszych osobników, a skupia się na ulepszeniu bieżącego rozwiązania. Na samym końcu temperatura T powinna zostać obniżona do wartości minimalnej T_{MIN}, która na ogół przyjmuje wartość bliską zeru.

Większość modyfikacji algorytmu *SA* dotyczy sposobu zmiany temperatury oraz liczby losowań L.

Proces schładzania może być realizowany za pomocą funkcji [99]:

- stałej: $T(t) = C$,
- arytmetycznej: $T(t) = T(t-1) - C$,
- geometrycznej: $T(t) = \alpha(t)T(t-1)$,
- odwrotnej: $T(t+1) = \frac{C}{t+1}$,
- logarytmicznej: $T(t+1) = \frac{C}{ln(t+1)}$.

Liczba losowań L osobników sąsiednich może przyjąć postać [99]:

- pojedynczą: $L(t) = 1$,
- stałą: $L(t) = C$,
- arytmetyczną: $L(t) = L(t-1) + C$,
- geometryczną: $L(t) = \frac{L(t-1)}{\alpha(t)}$,
- logarytmiczną: $L(t) = \frac{C}{lnT(t)}$,
- potęgową: $L(t) = L(t-1)^{\frac{1}{\alpha(t)}}$.

W wielu implementacjach stosuje się wariant z zapamiętywaniem najlepszego, dotychczas odgadniętego rozwiązania. W pierwotnej wersji *SA* zwracane jest rozwiązanie otrzymane w momencie osiągnięcia warunku stopu. Pociąga to za sobą możliwość porzucenia najlepszego rozwiązania. W rezultacie wynikiem pracy algorytmu będzie rozwiązanie gorsze, mimo, że to najlepsze zostało wcześniej odgadnięte.

4.2.3 Przeszukiwanie z listą Tabu

Ostatnią z wymienionych metod optymalizacji lokalnej jest algorytm **przeszukiwania z listą Tabu** (ang. *Tabu Search, TS*). Metoda ta została zaproponowana przez Glovera w 1986 roku [37]. Ideą tego algorytmu jest próba uniknięcia wielokrotnych nawrotów do sprawdzonych wcześniej rozwiązań, co bez wątpienia mogło być obecne w przedstawionych wcześniej technikach. Algorytm *TS* pozwala na przechowywanie listy aktualnie sprawdzonych rozwiązań - pod postacią listy Tabu, do których nie może już powrócić w najbliższej przyszłości [2]. Zbiór zabronionych punktów przechowywany jest w pamięci pod postacią haszowanej listy cyklicznej - tak by dostęp był wydajny oraz w celu zabezpieczenia się przed dużą zajętością pamięci. Czas życia punktu na liście Tabu określany jest mianem kadencji oraz trwa on pewną, z góry określoną, liczbę iteracji.

Dzięki obecności listy dotychczas sprawdzonych rozwiązań można tutaj mówić o dynamicznej zmianie sąsiedztwa. Może okazać się, że zbiór $\mathcal{N}(K_i)$, dla każdego K_i, nigdy nie będzie stały, ponieważ jego zawartość może ulec zmianie na podstawie punktów

zapamiętanych w pamięci. Algorytm *TS* został przedstawiony na poniższym pseudokodzie:

Algorytm 5: Algorytm przeszukiwania z listą Tabu

```
1  K_0 := generuj_losowo(X)
2  lista_tabu := [ ]
3  lider := K_0
4  repeat
5  |  najlepszy_sąsiad := max_{k∈N(K_0)−lista_tabu} (F_f(k))
6  |  K_0 := najlepszy_sąsiad
7  |  if F_f(K_0) > F_f(lider) then
8  |  |  lider := K_0
9  |  end
10 |  aktualizacja_listy(lista_tabu)
11 |  t := t + 1
12 until t < t_MAX
```

Największą zaletą tego algorytmu jest możliwość zapamiętywania oraz pomijania rozwiązań już odwiedzonych. Dzięki nakładaniu tego typu ograniczeń *TS* jest w stanie unikać maksym lokalnych. Ponadto technika ta przeszukuje całą przestrzeń rozwiązań z dala od punktu startowego.

Główną wadą tej metody jest dobór właściwej wartości parametru kadencji. Warto wspomnieć, że dłuższe kadencje zmniejszają ryzyko wpadnięcia algorytmu w cykl oraz poszerzają zakres przeszukiwania globalnej przestrzeni rozwiązań. Z drugiej strony krótsze kadencje, mimo, że zwiększają dokładność przeszukiwania, zwiększają prawdopodobieństwo wpadnięcia algorytmu w cykl. Inną wadą *TS* może być zakazanie niektórych, czasami nawet bardzo dobrych, ruchów, ze względu na nieco mniejszą wartość F_f.

Z najbardziej znanych modyfikacji algorytmu *TS* wymienić można **kryterium aspiracji**. Każdy punkt znajdujący się w sąsiedztwie $\mathcal{N}(K_i)$ może zostać odwiedzony, mimo, że znajduje się on na już liście Tabu. Definiowana jest specjalna funkcja aspiracji, decydująca kiedy zakazane rozwiązanie może zostać zaakceptowane [36].

Najczęściej spotykanym wariantem jest akceptacja kolejnego punktu, gdy charakteryzuje się on lepszą wartością F_f. Jest to sytuacja w której można zauważyć, że dowolne zakazane rozwiązanie jest zdecydowanie lepsze od aktualnie najlepszego punktu [36].

Zależność tą można zapisać jako:

$$Aspiracja(x, lider) = \begin{cases} TRUE, & \text{gdy } F_f(x) > F_f(lider), \\ FALSE & \text{w przeciwnym wypadku.} \end{cases} \tag{4.3}$$

Inna wersja kryterium akceptacji zakłada możliwość wykonania zabronionego ruchu w przypadku, gdy prowadzi on do punktu lepszego od aktualnie najlepszego o pewną ustaloną wartość α [36]. Funkcja realizująca tego typu kryterium została zdefiniowana następująco:

$$Aspiracja(x, lider) = \begin{cases} TRUE, & \text{gdy } (F_f(x) - F_f(lider)) \geqslant \alpha, \\ FALSE & \text{w przeciwnym wypadku.} \end{cases} \tag{4.4}$$

W literaturze można również spotkać się z całą masą modyfikacji związanymi z doborem parametru kadencji. Do najpopularniejszych z nich należą m.in. metody [36]:

- losowej kadencji - w każdej iteracji losuje się nową wartość parametru co wprowadza zmienną długość listy Tabu,
- kadencji adaptacyjnej - większa wartość F_f pociąga za sobą zmniejszenie parametru kadencji, co wpływa na lepszą eksploatację aktualnej podprzestrzeni rozwiązań,
- kadencję wybieraną cyklicznie - wartość kadencji wybierana jest z pewnego, z góry ustalonego, ciągu.

5

Algorytmy memetyczne w kryptoanalizie symetrycznych szyfrów blokowych

Za pomocą algorytmów memetycznych *MA* opracowano dwa autorskie ataki ewolucyjne, opierające się na wykorzystaniu metod kryptoanalizy różnicowej. Przygotowane ataki, skierowane są zarówno przeciwko szyfrogramom wygenerowanym za pomocą szyfru *FEAL*, jak i standardowi szyfrowania danych *DES*. W dalszej części rozdziału szczegółowo scharakteryzowano opracowane w ramach rozprawy ataki kryptoanalityczne.

5.1 Atak memetyczny skierowany na szyfr *FEAL*

Niemal wszystkie współczesne algorytmy szyfrujące składają się z pewnej liczby nieliniowych komponentów. Najczęściej jest to funkcja rundy f. Nie jest możliwe znalezienie jakiegokolwiek wzorca lub szablonu, za pomocą którego możliwe byłoby przewidzenie kolejnej wartości tej funkcji. Warto zaznaczyć, że tego typu funkcja nie może generować żadnych wartości pseudolosowych - w takim przypadku nie bylibyśmy w stanie zdeszyfrować żadnego kryptogramu, ponieważ funkcja byłaby nieodwracalna.

Jak już wspomniano w opisie metody kryptoanalizy różnicowej, dla każdej z różnic przypisuje się pewne prawdopodobieństwo, dla którego funkcja f będzie zawsze zwracała określoną wartość - mowa tutaj o charakterystykach. Wyznaczenie prawdopodobieństwa dla każdej różnicy pociąga za sobą wygenerowanie całych tablic charakterystyk,

na podstawie których można wyłonić najbardziej prawdopodobne podklucze [12].

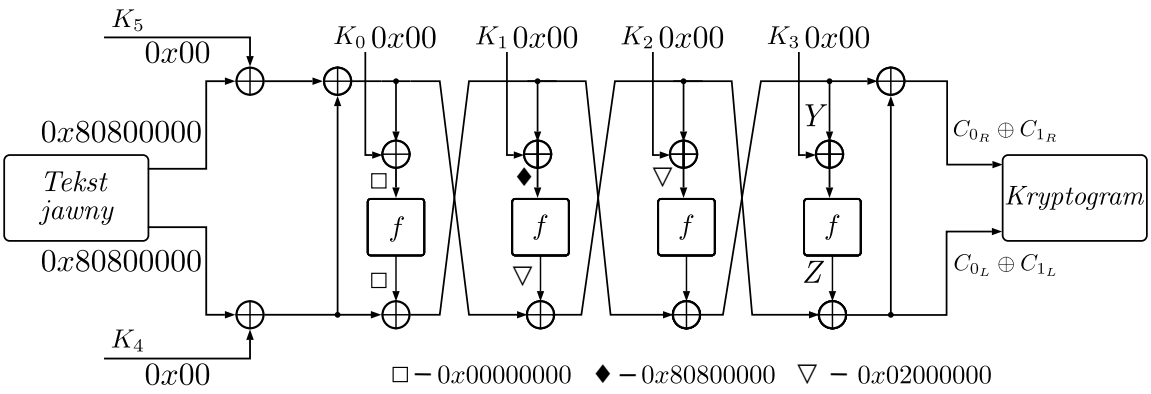
W przypadku ataku na szyfr *FEAL* ograniczono się do wykorzystania tylko jednej charakterystyki, aczkolwiek o największym możliwym stopniu prawdopodobieństwa, równym dokładnie 100%. Mowa tutaj o dowolnej parze bloków danych wejściowych o różnicy **0x80800000**. Na wyjściu funkcji rundy f zawsze zostanie zwrócona różnica o wartości **0x02000000** [94]. Szczegółowo opisany dowód, dotyczący prawdopodobieństwa wykorzystanej charakterystyki, można znaleźć w pracy [94] - Problem 28. Wspomnianą charakterystykę można zapisać jako:

$$\begin{aligned} Y &= Y_0 \oplus Y_1 = 0x80800000, \\ Z &= f(Y_0) \oplus f(Y_1) = 0x02000000, \end{aligned} \tag{5.1}$$

gdzie:

- Y_0 i Y_1 - oznaczają dwa bloki danych wejściowych, wchodzących do funkcji f,
- Y - jest różnicą bloków Y_0 oraz Y_1,
- Z - stanowi różnicę wartości funkcji dla wygenerowanych bloków.

Jeżeli znana jest już charakterystyka o odpowiednio wysokim prawdopodobieństwie możliwe staje się przeprowadzenie procesu analizy różnicowej szyfru. Na rys. 5.1 przedstawiono po raz kolejny schemat działania algorytmu *FEAL*. Tym razem wzbogacono go o różnicę symetryczną pomiędzy poszczególnymi blokami na każdym etapie szyfru.

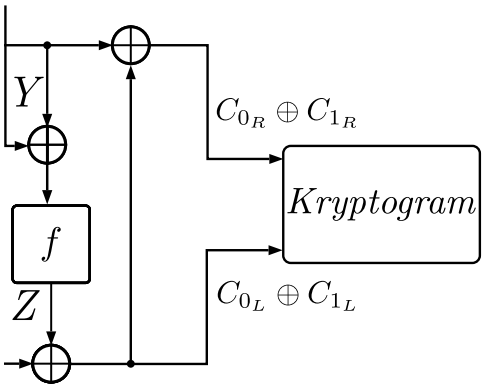


Rysunek 5.1: Analiza różnicowa szyfru *FEAL*

Dobór tekstów jawnych ograniczać się może do pseudolosowego wygenerowania jednego z nich, natomiast drugi tekst można uzyskać wykonując operację różnicy symetrycznej z wartością **0x8080000080800000**:

$$\Delta P = P_0 \oplus P_1 = 0x8080000080800000. \tag{5.2}$$

Analizowany schemat cały czas operuje na różnicach pomiędzy blokami, dlatego konieczne jest wykonanie operacji różnicy symetrycznej na samych podkluczach - dlatego też ich wartość zawsze będzie równa **0x00**. Przewidzenie wyniku każdej z operacji różnicy symetrycznej jest możliwe tylko do trzeciej rundy algorytmu szyfrującego, kiedy to do funkcji oceny przekazywana jest wartość **0x02000000**. Dla tej różnicy nie można jednoznacznie określić jaka będzie wartość funkcji rundy f. Problem ten można rozwiązać podążając od końca algorytmu szyfrującego, analizując wygenerowany szyfrogram (rys. 5.2).



Rysunek 5.2: Ostatnia runda algorytmu *FEAL*

Analizując parę wygenerowanych szyfrogramów można stwierdzić, że:

$$\begin{aligned} L &= C_{0L} \oplus C_{1L}, \\ Y_0 &= C_{0L} \oplus C_{0R}, \\ Y_1 &= C_{1L} \oplus C_{1R}, \\ Z &= (C_{0L} \oplus C_{1L}) \oplus 0x02000000. \end{aligned} \tag{5.3}$$

Na podstawie powyższych wyrażeń należy wyznaczyć pierwszy z podkluczy K_3. W przypadku zastosowania typowego algorytmu siłowego wiązać się to może z przeglądem wszystkich 2^{32} możliwych kombinacji, co stanowi około 4,3 miliarda podkluczy. W tym miejscu zdecydowano się na zastosowanie algorytmu memetycznego.

Każdy osobnik, z całej populacji rozwiązań, reprezentuje jeden 32-bitowy podklucz. Populacja początkowa $P(0)$ składa się z N losowo wygenerowanych podkluczy, które wraz ze wzrostem iteracji oraz pod wpływem działania operatorów genetycznych, będzie ewoluować w celu poprawy jakości swoich rozwiązań. Każdy z osobników musi zostać poddany ocenie przy pomocy funkcji przystosowania F_f w celu określenia jakości zaproponowanego rozwiązania. Funkcja przystosowania dla zaproponowanego ataku kryptoanalitycznego wygląda następująco:

$$F_f = \sum_{i=0}^{n} L - H((f(k \oplus Y_{0i}) \oplus f(k \oplus Y_{1i})), Z_i), \tag{5.4}$$

gdzie:

- L - długość podklucza,
- H - oznacza odległość Hamminga,
- f - funkcja rundy algorytmu szyfrującego,
- k - aktualnie analizowany podklucz,
- n - liczba wygenerowanych par szyfrogramów oraz odpowiadających nim tekstów jawnych.

Funkcja przystosowania wyznacza liczbę różnic pomiędzy wartością otrzymaną z funkcji rundy, a znaną różnicą Z, wyznaczoną na podstawie szyfrogramów. Większa wartość funkcji F_f oznacza bardziej atrakcyjniejszego osobnika.

Po wygenerowaniu i ocenie populacji początkowej uruchamiany jest proces ewolucji. Standardowo, składa się on z operatorów genetycznych, takich jak selekcja, krzyżowanie czy mutacja, aczkolwiek jest on również wzbogacony o dodatkowy element eksploatacyjny, odpowiedzialny za proces przeszukiwania lokalnego. Krok ten uruchamiany jest od razu po operatorze mutacji. Po wykonaniu wszystkich operacji powstaje nowa populacja potomna *O(t)*, która na nowo zostaje poddana ocenie. Proces powtarzany jest do momentu wykonania określonej liczby iteracji lub uzyskania satysfakcjonującego wyniku. Dodatkowo algorytm zapamiętuje najlepiej przystosowanego osobnika w populacji.

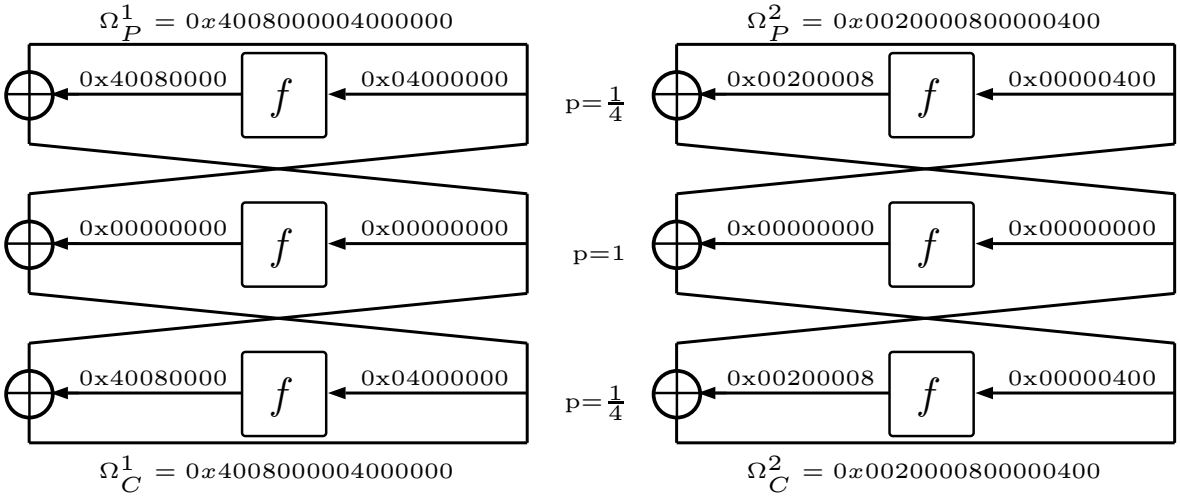
W zaproponowanym algorytmie zdecydowano się na wykorzystanie selekcji turniejowej, której zadaniem jest wyłonienie dwóch osobników rodziców do procesu krzyżowania. Rekombinacja osobników realizowana jest za pomocą operatora krzyżowania jednopunktowego. Następnie uruchamiany jest operator mutacji, który w sposób pseudolosowy, wybiera jeden bit podklucza i zamienia go na przeciwny. Proces przeszukiwania lokalnego opiera się na wykorzystaniu algorytmu symulowanego wyżarzania.

Pełny atak kryptoanalityczny polega na uruchomieniu *MA* czterokrotnie, w celu odgadnięcia wszystkich podkluczy K_0 - K_3.

5.2 Atak memetyczny skierowany na szyfr *DES*

Z punktu widzenia opracowanego ataku permutacje IP oraz IP^{-1} mogą zostać całkowicie pominięte - nie wnoszą one żadnej wartości dla procesu kryptoanalizy. Al-

gorytm rozpoczyna się od wyboru dwóch, najbardziej prawdopodobnych, 3-rundowych charakterystyk Ω_P^1 i Ω_P^2 wspomnianych w [94, 95]. Owe charakterystyki zostały przedstawione na rys. 5.3.



Rysunek 5.3: 3-rundowe charakterystyki Ω_P^1 i Ω_P^2 algorytmu *DES*

Prawdopodobieństwo każdej z charakterystyki wynosi $P_\Omega = \frac{1}{16}$. W czwartej rundzie algorytmu szyfrującego S-Bloki S_2, S_5, S_6, S_7, S_8 dla Ω_P^1 oraz S_1, S_2, S_4, S_5, S_6 dla Ω_P^2 dla pewnej wejściowej różnicy symetrycznej B'_j zwracają wyjściową różnicę symetryczną C'_j równą zero. Na podstawie tej obserwacji można wyznaczyć zbiory $I_1 = \{2, 5, 6, 7, 8\}$ dla Ω_P^1 oraz $I_2 = \{1, 2, 4, 5, 6\}$ dla Ω_P^2. Dalszy opis ataku jest identyczny dla Ω_P^1 i Ω_P^2 dlatego zdecydowano się go uogólnić wprowadzając zbiór I zawierający kolejno elementy ze zbioru I_1, a następnie elementy ze zbioru I_2.

Kolejnym etapem jest wygenerowanie zbioru par tekstów jawnych, których różnica symetryczna odpowiada charakterystykom Ω_P^1 oraz Ω_P^2. Na podstawie tego zbioru wyznaczany jest również zbiór kryptogramów. Liczba par obliczana jest za pomocą stosunku sygnału do szumu [11]:

$$S/N = \frac{m \cdot p}{m \cdot \alpha \cdot \beta / 2^k} = \frac{2^k \cdot p}{\alpha \cdot \beta} = \frac{2^{30} \cdot {}^1/_{16}}{4^5} = 2^{16}, \tag{5.5}$$

gdzie:

- m - liczba wygenerowanych par, nie mająca wpływu na S/N,
- p - prawdopodobieństwo wybranej charakterystyki Ω,
- k - liczba bitów poszukiwanego podklucza,
- α - średnia liczba podkluczy, sugerowana przez jedną parę,

- β - stosunek analizowanych par do wszystkich możliwych.

Zgodnie z sugestią opisaną w [11] dla $S/N = 2^{16}$ potrzebne jest 7 - 8 poprawnych par dla każdej z charakterystyk. Ze względu na prawdopodobieństwo P_Ω należy wygenerować minimum 120 par tekstów jawnych [11].

Zbiór wygenerowanych par zostaje dodatkowo poddany operacji filtracji. Dla każdej pary, na podstawie wyrażenia 2.13 z rozdziału 2, wyznaczany jest zbiór $test_j$. Jeżeli moc zbioru testowego, dla chociaż jednego elementu ze zbioru I, będzie równa 0, to para zostanie odrzucona:

$$\bigwedge_{j \in I} |test_j| > 0. \tag{5.6}$$

Zadaniem zaproponowanego ataku jest odgadnięcie ostatniego podklucza szyfrującego K_6. Jeżeli znana będzie różnica C' oraz część R_5, będziemy w stanie przeanalizować różne podklucze porównując wyjście S-bloków z C'. Zastosowanie ataku siłowego wymagałoby sprawdzenia wszystkich 2^{30} możliwych rozwiązań. Wykorzystanie MA może zostać tutaj wykorzystane jako doskonałe narzędzie optymalizacyjne, znajdujące poprawne rozwiązanie w znacznie krótszym czasie.

Każdy z osobników reprezentowany jest przez 30-bitowy podklucz K_j. Funkcja oceny wygląda następująco:

$$F_f = \sum_{i=0}^{n} L - \sum_{j \in I} H((S_j(B_j) \oplus S_j(B_j^*)), P^{-1}(R'_6 \oplus L'_3)), \tag{5.7}$$

gdzie:

- H - oznacza odległość Hamminga,
- L - długość podklucza.

Z prawdopodobieństwem P_Ω możliwe jest oszacowanie wartości L'_3, natomiast R'_6 można uzyskać analizując parę wygenerowanych szyfrogramów. F_f zlicza liczbę pokrywających się bitów pomiędzy różnicą otrzymaną z S-bloków a różnicą C'.

W algorytmie wykorzystano krzyżowanie jednopunktowe. Punkt przecięcia wybierany jest pseudolosowo z zakresu od 1 do 30. Nowo powstałe chromosomy mogą ulec operatorowi mutacji, którego zadaniem jest zamiana dwóch losowo wybranych bitów podklucza. Do poprawnie wykonanej operacji krzyżowania algorytm wykorzystuje selekcję turniejową. Spośród wszystkich podkluczy wybierany jest tylko jeden lider turnieju. Następnie proces selekcji jest powtarzany w celu wyłonienia drugiego rodzica.

Na końcu algorytmu aktywowany jest dodatkowy operator odpowiadający za proces przeszukiwania lokalnego. Za realizację tego kroku odpowiedzialny jest klasyczny algorytm symulowanego wyżarzania. Pseudokod ataku *MASA* (ang. *Memetic Algorithm Simulated Annealing*), dla charakterystyki Ω_P, przedstawiono poniżej. Ze względu na złożoność tego algorytmu, zdecydowano się na podzielenie całości na dwie części:

- pierwsza, pseudokod 6 - odpowiedzialna za wygenerowanie zbioru odfiltrowanych par tekstów jawnych, szyfrogramów oraz wyznaczenie zbioru testowego $test_j$ dla każdego z indeksów,
- druga, zaprezentowana na pseudokodzie 7 - opisująca algorytm memetyczny, wraz z procesami selekcji, krzyżowania, mutacji oraz eksploatacji, z uwzględnieniem pseudokodu podstawowego algorytmu symulowanego wyżarzania.

Algorytm 6: Pseudokod procesu przygotowania zbioru tekstów dla ataku *MASA*

```
1  Ω_P := znajdź_najbardziej_prawdopodobna_charakterystykę()
2  I := wyznacz_zbiór_indeksów
3  zbiór_par := wygeneruj_zbiór_par_tekstów_jawnych_i_szyfrogramów()
4  for i := 0 to rozmiar(zbiór_par) do
5      para := zbiór_par[i]
6      foreach j ∈ I do
7          test_j := wyznacz_zbiór_testowy(para)
8          if |test_j| == 0 then
9              odfiltruj_parę_ze_zbioru_par(zbiór_par, para)
10             break
11         end
12     end
13 end
```

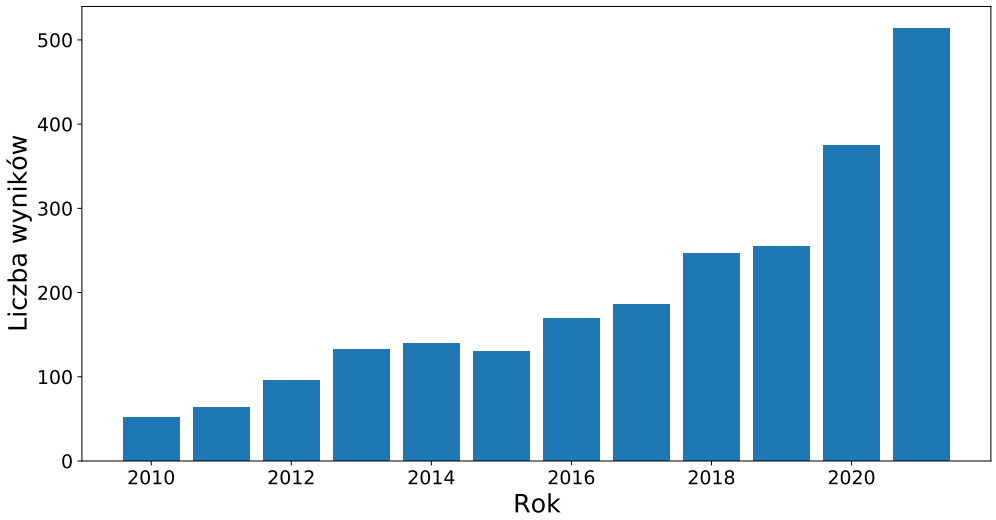
Uruchomienie algorytmu *MASA* dla Ω_P^1 umożliwi odgadnięcie 30 z 48 bitów podklucza K_6. Ponowne uruchomienie algorytmu, tym razem dla Ω_P^2, pozwala na odszukanie dodatkowych 12 bitów. W celu uzyskania pozostałych 6 bitów podklucza K_6 - pochodzących od S-bloku S_3, można posłużyć się metodą przeglądu zupełnego. Dysponując podkluczem K_6 możliwe jest odtworzenie 48 z 56 bitów klucza deszyfrującego poprzez odwrócenie procesu rozkładu klucza. Pozostałe 8 bitów można odgadnąć stosując po raz kolejny metodę przeglądu zupełnego.

Algorytm 7: Pseudokod ataku *MASA*

```
1  P(0) := zainicjuj_populację_początkową()
2  for i := 0 to liczba_iteracji do
3      oblicz_wartość_funkcji_oceny_dla_każdego_osobnika()
4      for j := 0 to rozmiar_populacji do
5          rodzic_A := selekcja_turniejowa()
6          rodzic_B := selekcja_turniejowa()
7          potomstwo := [rodzic_A, rodzic_B]
8          if random(0, 1) ⩾ prawdopodobieństwo_krzyżowania then
9              potomek_A, potomek_B := krzyżowanie(rodzic_A, rodzic_B)
10             if random(0, 1) ⩾ prawdopodobieństwo_mutacji then
11                 potomek_A := mutacja(rodzic_A)
12             end
13
14             if random(0, 1) ⩾ prawdopodobieństwo_mutacji then
15                 potomek_B := mutacja(rodzic_B)
16             end
17             potomstwo := [potomek_A, potomek_B]
18         end
19
20         foreach potomek ∈ potomstwo do
21             T = T_0
22             while T ⩾ T_MIN do
23                 nowy_potomek := zamiana_losowego_bitu(potomek)
24                 difference := nowy_potomek.fitness − potomek.fitness
25                 if difference > 0 or
26                   probability_fun(difference, T) > random(0, 1) then
27                     potomek := nowy_potomek
28                 end
29                 T := T · α
30             end
31         end
32     end
33 end
```

5.3 Inne algorytmy ewolucyjne w problemie kryptoanalizie szyfrów blokowych

W ramach tego rozdziału przeprowadzono analizę popularności tematyki metaheurystycznej kryptoanalizy różnicowej. Dane zostały zaczerpnięte za pomocą wyszukiwarki **Google Schoolar**, mającej dostęp do bazy danych publikacji naukowych. Poniżej zilustrowano liczbę wyników wyszukiwania, dotyczących terminu ewolucyjnej i metaheurystycznej kryptoanalizy.



Rysunek 5.4: Diagram popularności metaheurystycznej kryptoanalizy na przestrzeni ostatnich lat

Analizując rys. 5.4 można zauważyć wyraźny wzrost zainteresowania publikacjami dotyczącymi wykorzystania technik metaheurystycznych w kryptografii.

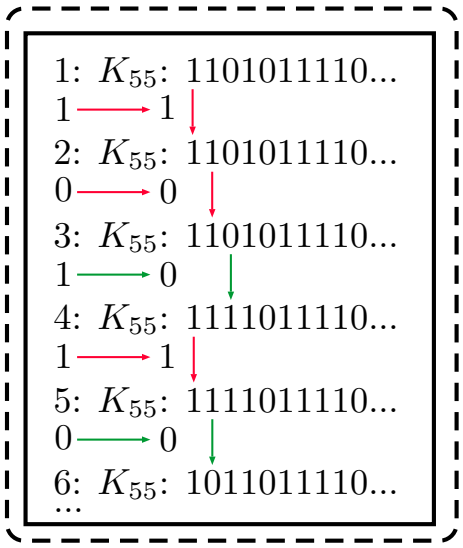
W dalszej części rozdziału znajduje się opis dwóch, pochodzących z literatury, ataków ewolucyjnych, skierowanych przeciwko szyfrogramom wygenerowanym za pomocą wspomnianych wyżej algorytmów szyfrujących.

5.3.1 Atak na szyfr *FEAL* z wykorzystaniem algorytmów ewolucyjnych

W artykule [22] przedstawiono pierwotny *EA*, służący do ewolucyjnej kryptoanalizy różnicowej. Ze względu na zbyt małą liczbę publikacji dotyczących ewolucyjnej kryptoanalizy szyfru *FEAL* zdecydowano się na porównanie z opracowanym przez samego siebie oryginalnym atakiem na ten szyfr. Algorytm *NEA* w dużej mierze pokrywa się z opisem ataku memetycznego opisanego w podrozdziale 5.1.

Podobnie jak wcześniej wykorzystano tutaj selekcję turniejową oraz wariant krzyżowania jednopunktowego. Operator mutacji losuje jeden gen chromosomu podklucza, który następnie poddawany jest operacji różnicy symetrycznej wraz z odpowiadającym mu genem z drugiego potomka.

Ponadto opisany algorytm został rozbudowany o dodatkowy element, tzw. **operator heurystycznej negacji**. Element ten został wprowadzony w celu poprawy jakości procesu eksploatacji *EA*. Zasada działania operatora heurystycznej negacji została zaprezentowana na rys. 5.5.



Rysunek 5.5: Zasada działania operatora heurystycznej negacji [22]

Przedstawiona modyfikacja pociąga za sobą negację każdego genu chromosomu, z pewnym, z góry ustalonym, prawdopodobieństwem P_n. Następnie, przy pomocy funkcji F_f, zmodyfikowany klucz zostaje poddany ocenie. Jeżeli wariant okaże się być korzystniejszy, zostaje on zapamiętany, w przeciwnym wypadku proces negacji bitu zostaje anulowany. Operator ten uruchamiany jest od razu po zakończeniu procesu mutacji.

Algorytm *NEA* sprawdza znacznie mniejszą liczbę możliwych podkluczy w przeciwieństwie do ataku siłowego oraz klasycznego algorytmu kryptoanalizy różnicowej. Warto również zaznaczyć, że wspomniany algorytm analizuje tylko jeden podklucz, zamiast wszystkich czterech, podczas każdego uruchomienia, dzięki czemu efekt lawinowy, popularny wśród znacznej większości symetrycznych szyfrów blokowych, zostaje wyeliminowany.

5.3.2 Atak na szyfr *DES* z wykorzystaniem algorytmu *DPSO*

W 2010 roku Ghonaim, Ghali oraz Hassanien zaprezentowali atak na czterorundowy szyfr *DES* z wykorzystaniem algorytmu optymalizacji stadnej cząsteczek [34]. Opracowany algorytm opiera się na podejściu ataku z szyfrogramem, w którym zakłada się, że atakujący znajduje się w posiadaniu pewnej liczby szyfrogramów, które zostały wygenerowane przy pomocy tego samego algorytmu oraz klucza szyfrującego. Zaprezentowane podejście wykorzystuje popularną w kryptologii metodę zwaną **analizą częstotliwości** (ang. *Frequency Analysis*). W ramach tej rozprawy algorytm zostanie oznaczony akronimem *DPSO*.

Analiza częstotliwości polega na wyznaczeniu częstości występowania wszystkich znaków, wchodzących w skład danego alfabetu A, w odszyfrowanym kryptogramie [8, 21]. Następnie dokonuje się porównania z najczęściej występującymi znakami w wybranym języku [21]. Proces analizy częstotliwości nie opiera się tylko i wyłącznie na porównywaniu częstości występowania pojedynczych znaków - zwanych dalej **unigramami**. Wprowadzono specjalną notację n-gramową, pozwalającą na porównywanie fragmentów słów, składających się kolejno z dwóch znaków - **bigramy** oraz trzech znaków - **trigramy** [8].

W algorytmie *DPSO* osobniki reprezentowane są jako cząsteczki, natomiast populacja nazywana jest rojem. Każda z cząsteczek porusza się po wielowymiarowej przestrzeni w celu znalezienia najbardziej optymalnego rozwiązania. Pozycje X_i poszczególnych cząsteczek i obliczane są na podstawie poprzednich pozycji cząsteczki, względem najlepszych osobników znajdujących się w sąsiedztwie oraz względem całego roju. Z każdą iteracją algorytmu aktualizowany jest wektor prędkości V_i cząsteczki, a następnie, na jego podstawie, przemieszcza się do nowej lokalizacji. W przypadku zaproponowanego ataku [33, 34], każda cząsteczka reprezentuje 56-bitowy ciąg binarny stanowiący klucz deszyfrujący. Prędkość cząsteczki reprezentowana będzie jako wektor $V_i = (V_1, V_2, ..., V_D)$, natomiast pozycja jako: $X_i = (X_1, X_2, ..., X_D)$, gdzie D oznacza długość poszukiwanego klucza deszyfrującego - w tym przypadku 56.

Na samym początku algorytmu, wektory pozycji oraz prędkości, dla każdej cząsteczki, generowane są losowo. Wraz z każdą następną iteracją aktualizowana jest wartość wektora pozycji na podstawie następującego wyrażenia [33, 34]:

$$V_{id}(t+1) = w \cdot V_{id}(t) + C_1 r_1 (pbest_{id} - X_{id}(t)) + C_2 r_2 (gbest_d - X_{id}(t)), \qquad (5.8)$$

gdzie:

- w - współczynnik bezwładności, określa wpływ prędkości z poprzedniej iteracji,
- C_1 - współczynnik dążenia do najlepszego rozwiązania lokalnego,
- C_2 - współczynnik dążenia do najlepszego rozwiązania globalnego,
- r_1, r_2 - wartości losowe spełniające zależność $r_1, r_2 \in [0, 1]$,
- $pbest_{id}$ - dotychczasowe najlepsze położenie cząsteczki,
- $gbest_d$ - dotychczasowe najlepsze położenie wśród wszystkich cząsteczek,
- t - aktualna iteracja algorytmu.

Przy czym $i = 1, 2, ..., N$, gdzie N oznacza rozmiar roju. Warto również zaznaczyć, że każda z wartości wewnątrz wektora prędkości nie może przekroczyć pewnej, z góry ustalonej, prędkości maksymalnej V_{max}. Jeżeli próg zostanie przekroczony nastąpi ograniczenie prędkości do postaci: $V_{id}(t + 1) = V_{max}$.

Na podstawie zaktualizowanego wektora prędkości V_i można wyznaczyć nową pozycję cząsteczki przy pomocy poniższego wzoru [33, 34]:

$$X_{id}(t + 1) = X_{id}(t) + V_{id}(t + 1). \tag{5.9}$$

Podczas działania algorytmu każda z cząsteczek zostanie poddana ocenie, na podstawie której będzie można określić jakość aktualnie zajmowanej pozycji. Funkcja oceny opiera się o wykorzystanie, wspomnianej wcześniej, analizy częstotliwości. Wzięte pod uwagę zostaną częstości występowania unigramów oraz bigramów w odszyfrowanym szyfrogramie, oraz zostaną one porównane z odpowiednimi statystykami n-gramów języka angielskiego - statystyki te można samodzielnie obliczyć na podstawie dowolnego tekstu napisanego w języku angielskim, aczkolwiek dłuższy tekst, pociąga za sobą bardziej wiarygodne statystyki. Funkcja przystosowania F_f została przedstawiona za pomocą następującego wyrażenia [33, 34]:

$$F_f = \alpha \sum_{i \in A} |K^u(i) - D^u(i)| + \beta \sum_{i,j \in A} |K^b(i, j) - D^b(i, j)|, \tag{5.10}$$

gdzie:

- A - alfabet języka angielskiego o postaci $A = \{A, B, ..., Z, \wp\}$, gdzie $\wp$ to oznacza symbol spacji,
- K - częstość występowania n-gramów w języku angielskim,
- D - częstość występowania n-gramów w języku odszyfrowanym kryptogramie,
- u, b - identyfikatory unigramów i bigramów,

- α, β - parametry wagowe, określają stopień użyteczności danej statystyki n-gramowej.

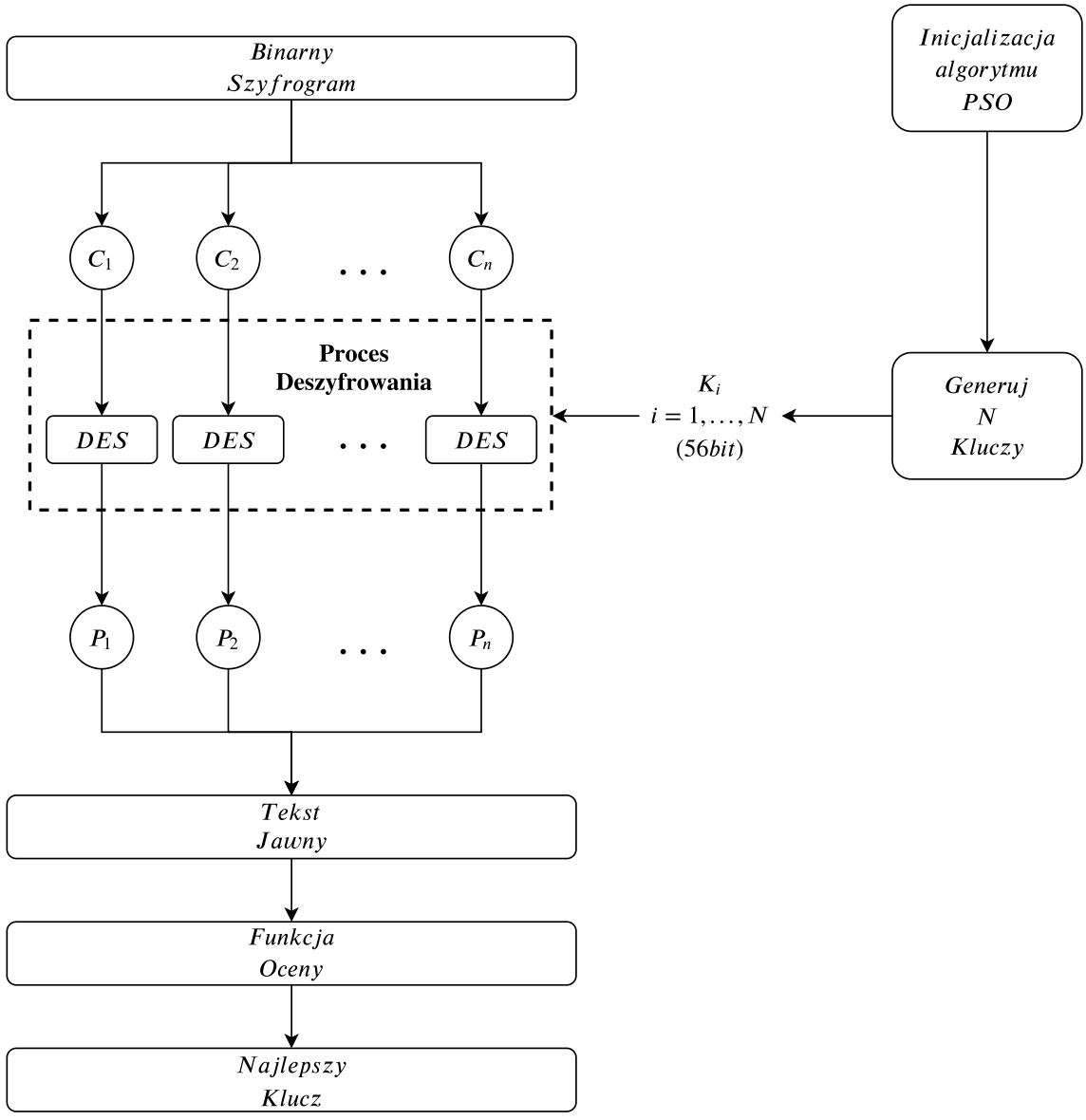
Warto zaznaczyć, że warunek $\alpha+\beta = 1$ musi zostać spełniony. Pseudokod algorytmu wygląda następująco (alg. 8):

Algorytm 8: Pseudokod algorytmu *DPSO*

```
1  P(0) := zainicjuj_populację_początkową()
2  for t := 0 to liczba_iteracji do
3      oblicz_wartość_funkcji_oceny_dla_każdej_cząsteczki()
4      for i := 0 to rozmiar_roju do
5          wyznacz_sąsiadów()
6          if F_f(X_id) < F_f(pbest_id) then
7              pbest_i := X_id
8          end
9          if F_f(X_id) < F_f(gbest_d) then
10             gbest_i := X_id
11         end
12         zaaktualizuj_pozycję()_oraz_prędkość()
13         zaaktualizuj_współczynnik_bezwładności()
14     end
15 end
```

Schemat działania całego algorytmu został przedstawiony na rys. 5.6. Więcej szczegółów na temat omówionego ataku można znaleźć w rozprawie doktorskiej autora [33] oraz publikacji [34].

Na podstawie przeprowadzonych badań oraz wniosków z eksperymentów, przedstawionych w artykule [34], można przeczytać, że opracowana wersja ataku, a w szczególności funkcja oceny, może zostać wykorzystana w kryptoanalizie innych symetrycznych szyfrach blokowych. Dzięki temu zapisowi zdecydowano się na wykorzystanie tego typu ataku przeciwko sześciorundowemu algorytmowi *DES*.



Rysunek 5.6: Schemat działania algorytmu *DPSO*
opracowano na podstawie [33, 34]

6

Badania eksperymentalne zaproponowanych algorytmów

W rozdziale tym przedstawiono badania eksperymentalne dotyczące zaproponowanych w rozprawie algorytmów kryptoanalitycznych. Ewolucyjny atak memetyczny *MASA*, dedykowany szyfrowi *FEAL*, zostanie porównany z algorytmem opartym na wykorzystaniu operatora heurystycznej negacji *NEA*, szczegółowo opisanym w [22]. Zmodyfikowana wersja ataku *MASA*, skierowana przeciwko algorytmowi szyfrującemu *DES*, zostanie skonfrontowana z atakiem *DPSO* [34], wykorzystującym algorytm optymalizacji stadnej cząsteczek, oraz zmodyfikowanym algorytmem *NEA*, skierowanym również przeciwko szyfrowi *DES*. W rozprawie, zdecydowano się również, o przedstawienie zestawienia z podstawowymi implementacjami wersjami algorytmów siłowych dla wyżej wspomnianych szyfrów. Część z wyników, z przeprowadzonych doświadczeń, została przedstawiona w dodatku A.

6.1 Przygotowanie zbiorów testowych

W rozpatrywanej tematyce nie udało się znaleźć żadnego, ogólnie dostępnego, repozytorium zbiorów testowych. Dlatego też koniecznym stało się wygenerowanie potrzebnych szyfrogramów samodzielnie. Opracowane szyfrogramy, oraz teksty jawne, konieczne będą do przeprowadzenia ataku z wykorzystaniem algorytmu *DPSO*. Zdecydowano się przetestować dwa niezależne od siebie teksty jawne:

- artykuł opisujący, jak zmieniło się życie w Korei Południowej podczas trwania

pandemii Koronawirusa [107]. Artykuł ten napisany został potocznym językiem angielskim, z jakim się można spotkać w lokalnych gazetach. Poniżej przedstawiono fragment wykorzystanego artykułu:

"But for many, life appears to be going back to a new normal. People are out and about on the streets, but having to get their temperature taken before being let into events or buildings."[...],

- fragment specjalistycznego artykułu naukowego, z dziedziny chemii, dotyczącego przepływu energii w reakcjach chemicznych [106]. Zdecydowano się na wybór tego typu artykułu w celu przetestowania sprawności algorytmu z znacznie trudniejszym tekstem, wzbogaconym o dodatkowe słownictwo specjalistyczne. Fragment artykułu wygląda następująco:

  "Beyond curiosity to understand the world, we hope that practically this can become useful in guiding thinking about transducing chemical energy for molecular motion in liquids, for nanorobotics, precision medicine and greener material synthesis"[...]

Każdy z tekstów jawnych, przed rozpoczęciem procesu kryptologicznego, został oczyszczony z wszystkich znaków specjalnych, cyfr oraz znaków białych (poza spacjami). Ponadto wszystkie znaki alfabetyczne zostały zunifikowane tylko i wyłącznie do małych liter. W celu uproszczenia procesu kryptoanalizy, wszystkie wybrane teksty jawne były w języku angielskim.

Dla każdego z powyższych tekstów wygenerowano osobny zbiór szyfrogramów z użyciem algorytmu szyfrującego *DES*. Przedstawiona poniżej tab. 6.1 przedstawia listę wykorzystanych w fazie badań kluczy szyfrujących:

Tabela 6.1: Lista wykorzystywanych podkluczy w kryptoanalizie szyfru *FEAL* i *DES*

Id	*FEAL*	*DES*
1	B43C851CBD7AFBCE	34E9F71A20756231
2	FA060EA204A83B1D	BF8980377505B539
3	0E8D8664F1E8FF91	0D8E27040A46F6C2
4	BF0CD23D2E59FB2F	A49ECC6471E38D65
5	6C3282FE66CB9FFA	EB2275E5FFB1F2C4

W celu zwizualizowania poziomu przekształcenia informacji przedstawiono poniżej fragment specjalistycznego artykułu naukowego [106], zaszyfrowanego z wykorzystaniem algorytmu *DES*:

úý ßv.¥Ó4.c.tëvS.¾OÖ. Bwg.RÈK¬«#.õ d.¿. .× çê.ñ.¸Yê¢©Ë?+Z CÂ.ã.í³O· H¼Èê ü#*½tÆ..Ä ..¿6kj.¢?xWû iãTÔ.6.-ÖT çB²CÒ”PNÊzJ!..}Dje ¢..×*ë?Û4±.FÄ ·,U)»±Äj Õ..îfz˚BxMXRCÔO:¼Iá.$. .ëé:}¡ßÁU.údÚuéï..¶,ä Ñ²g;A.z¶.¡ÁE.«a=˚ñf¼æ F¸8} fbØ8.Ã.ÏeÅ’½õ.ä »4²c)T´Ø º´*

Eksperymenty przeprowadzone zostały na dwóch algorytmach szyfrujących: *FEAL* oraz *DES*, całkowicie różniących się strukturą, złożonością oraz charakterystykami. Każdy z opracowanych algorytmów wymagał dostosowania do każdego z wybranych szyfrów. Eksperymenty przeprowadzone zostały na takich samych kryptogramach oraz kluczach szyfrujących. Wszystkie doświadczenia zostały wykonane na komputerze wyposażonym w procesor Intel i7 2,10 GHz, 8GB pamięci RAM oraz system operacyjny Windows 10. Wszystkie ataki zaimplementowane zostały z wykorzystaniem języka C#. Każdy szyfr poddany został ewolucyjnej kryptoanalizie różnicowej trzydziestokrotnie przez wszystkie zaimplementowane ataki.

6.2 Dobór wartości parametrów

Analiza zaproponowanego ataku memetycznego *MASA* pod kątem jakości oraz liczby uzyskiwanych rozwiązań stanowiła ważny element pracy naukowej. W szczególności istotne było sprawdzenie, czy zaproponowane algorytmy pozwalają na poprawę czasu odnalezienia rozwiązań dla wybranych algorytmów szyfrujących. Kolejnym istotnym aspektem pracy było sprawdzenie, czy algorytm memetyczny *MASA* pozwala na efektywniejszą, a co za tym idzie, skuteczniejszą - kryptoanalizę różnicową. Zbadany został m.in. wpływ wypunktowanych poniżej parametrów na zbieżność algorytmu oraz jakość uzyskiwanych rozwiązań:

- liczba iteracji dla algorytmów *MASA*, *NEA* i *DPSO*,
- rozmiar populacji dla algorytmów *MASA*, *NEA* i *DPSO*,
- liczba par tekstów jawnych oraz par szyfrogramów γ dla algorytmów *MASA*, *NEA*,
- prawdopodobieństwo heurystycznej negacji P_n dla algorytmu *NEA*,
- temperatura początkowa T_0, minimalna T_{MIN} oraz współczynnik wygaszania α dla algorytmu *MASA*,
- wartości parametrów wagowych α oraz β dla algorytmu *DPSO*.

W przeprowadzonych doświadczeniach wartości parametrów stosowane były w różnych kombinacjach, a dla kolejnych eksperymentów ustalone zostały potencjalnie najlepsze wartości pod względem czasu pracy algorytmu. Dla algorytmu memetycznego *MASA* parametry zostały ustalone zgodnie z tab. 6.2 przedstawioną poniżej:

Tabela 6.2: Lista parametrów algorytmu *MASA* dla szyfrów *FEAL* i *DES*

Id	Opis parametru	Symbol	*FEAL*	*DES*
1	Maksymalna liczba iteracji	It_{MAX}	70	100
2	Rozmiar populacji	N	30	10
3	Liczba wygenerowanych par tekstów jawnych	γ	30	200
4	Rozmiar turnieju	T_{SIZE}	5	10
5	Prawdopodobieństwo krzyżowania	P_c	0,8	0,9
6	Prawdopodobieństwo mutacji	P_m	0,02	0,02
7	Temperatura początkowa	T_0	1	1
8	Temperatura minimalna	T_{MIN}	0,1	0,1
9	Współczynnik wygaszania	α	0,9	0,9

Dobór wszystkich parametrów, dla algorytmu *DPSO*, został szczegółowo opisany w publikacji [34]. Lista najważniejszych metryk została przedstawiona poniżej w tab. 6.3. Niestety, w artykule [34] nie doszukano się szczegółowych informacji dotyczących wykorzystania współczynnika bezwładności w.

Tabela 6.3: Lista parametrów algorytmu *DPSO* dla szyfru *DES*

Id	Opis parametru	Symbol	*DES*
1	Maksymalna liczba iteracji	It_{MAX}	100
2	Rozmiar roju	N	50
3	Maksymalna prędkość cząsteczki	V_{MAX}	4
4	Współczynnik dążenia do najlepszego rozwiązania lokalnego	C_1	2
5	Współczynnik dążenia do najlepszego rozwiązania globalnego	C_2	2
6	Stopień użyteczności unigramów	α	0,2
7	Stopień użyteczności bigramów	α	0,8

Próby kontaktu z autorem publikacji nie przyniosły żadnej odpowiedzi. Wykonano dodatkowy przegląd literaturowy, dotyczący doboru owego współczynnika w implementacjach algorytmu *PSO*. Na podstawie wyników badań, opublikowanych w publikacji [6], zdecydowano się na wybór tzw. losowego współczynnika bezwładności, charakteryzującego się, wg. autorów publikacji, najlepszą wydajnością. Wartość owego współczynnika wyrażana jest na podstawie następującego wzoru:

$$w = 0,5 + \frac{Rand()}{2}. \tag{6.1}$$

Opis parametrów algorytmu *NEA* został szczegółowo opisany w publikacji [22]. W tab. 6.4 przedstawiono najważniejsze, z punktu widzenia rozprawy, parametry algorytmu *NEA*:

Tabela 6.4: Lista parametrów algorytmu *NEA* dla szyfru *FEAL* oraz *DES*

Id	Nazwa	Symbol	*FEAL*	*DES*
1	Maksymalna liczba iteracji	It_{MAX}	100	100
2	Rozmiar populacji	N	70	10
3	Liczba wygenerowanych par tekstów jawnych	γ	200	200
4	Rozmiar turnieju	T_{SIZE}	5	10
5	Prawdopodobieństwo krzyżowania	P_c	0,8	0,9
6	Prawdopodobieństwo mutacji	P_m	0,01	0,02
7	Prawdopodobieństwo heurystycznej negacji	P_n	0,25	0,25

6.3 Badanie jakości rozwiązań zaproponowanych algorytmów

W ramach dwóch następnych podrozdziałów przedstawiono szczegółowe wyniki badań dla każdego ataku kryptoanalitycznego skierowanego przeciwko szyfrogramom wygenerowanym za pomocą algorytmu *DES*. Wyniki badań dla szyfru *FEAL* były w dużej mierze zbliżone do tych, wygenerowanych z wykorzystaniem szyfru *DES*. Ze względu na ograniczenia związane z objętością pracy zdecydowano się na opis badań szyfru *DES*. Również w ramach tego podrozdziału zestawiono ze sobą wszystkie algorytmy kryptoanalityczne, których parametry zostały przedstawione w tab. 6.2, 6.3 i 6.4.

6.3.1 Kryptoanaliza z wykorzystaniem ataków *NEA* oraz *MASA*

Jak wspomniano wcześniej w pracy, na potrzeby badań wykorzystano uproszczoną wersję szyfru *DES*, w której ograniczono liczbę rund z 16 do 6. Wszystkie pozostałe procesy zachodzące w algorytmie szyfrującym, takie jak: generowanie podkluczy czy kompresja z wykorzystaniem S-bloków, pozostały niezmienione.

Poniżej przedstawiono wartość funkcji przystosowania F_f dla algorytmów *MASA* oraz *NEA*. Każdy algorytm przetestowano na wszystkich pięciu kluczach szyfrujących, przedstawionych w tab. 6.1. Poniżej, w tab. 6.5 i 6.6, przedstawiono pięć pierwszych wyników dla każdej charakterystyki. Pozostałe wyniki zamieszczono w dodatku A.

W poniższej tabeli pogrubioną czcionką zaznaczono testy w których nie udało się odgadnąć poprawnego podklucza deszyfrującego. W przedostatniej kolumnie, oznaczonej

Tabela 6.5: Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* oraz Ω_P^1

Klucz Szyfrujący	Id	*MASA*						*NEA*					
		Min.	Med.	Śr.	Maks	Odch. Std.	t	Bit	Min.	Med.	Śr.	Maks	Odch. Std.	t	Bit
34E9F71A20756231	1	954	1090	1061,6	1110	56,2	17	30	1029	1090	1080,4	1110	31,0	17	30
	2	945	1090	1067,5	1110	50,0	15	30	953	1058	1045,1	1110	57,7	18	30
	3	958	1089	1069,0	1110	39,8	5	30	962	1067	1063,5	1110	48,9	14	30
	4	922	1089	1064,1	1110	50,8	31	30	**1048**	**1089**	**1081,3**	**1090**	**16,2**	**99**	**27**
	5	1037	1090	1089,3	1110	14,6	13	30	989	1090	1081,5	1110	34,1	12	30
BF8980377505B539	1	758	856	832,6	856	35,5	22	30	741	835	813,2	856	46,7	39	30
	2	762	844	829,0	856	33,6	31	30	702	839	820,3	856	43,3	38	30
	3	784	856	845,1	856	21,3	16	30	649	798	778,7	856	84,4	19	30
	4	745	821	818,2	856	38,5	14	30	775	856	830,3	856	33,2	10	30
	5	679	814	801,3	856	52,2	94	30	705	801	805,1	856	52,5	45	30
0D8E27040A46F6C2	1	962	1035	1048,4	1106	45,4	36	30	936	1056	1046,8	1106	59,4	61	30
	2	918	1017	1038,3	1106	71,8	52	30	885	1064	1038,5	1106	82,1	55	30
	3	957	1082	1064,1	1106	49,5	22	30	**891**	**1002**	**1008,8**	**1101**	**64,4**	**99**	**28**
	4	945	994	1024,2	1106	65,2	20	30	920	1020	1031,8	1106	67,3	49	30
	5	871	1056	1034,3	1106	84,7	55	30	**954**	**1066**	**1054,4**	**1101**	**45,7**	**99**	**28**
A49ECC6471E38D65	1	908	983	987,8	1038	44,6	13	30	904	976	984,9	1038	41,4	57	30
	2	954	1002	1002,5	1038	27,6	17	30	903	1021	998,2	1038	45,6	43	30
	3	956	985	996,0	1038	31,5	72	30	**921**	**990**	**974,2**	**1004**	**30,3**	**99**	**29**
	4	903	981	983,8	1038	44,9	27	30	886	966	975,8	1038	56,2	30	30
	5	897	1015	996,2	1038	48,1	70	30	952	977	989,6	1038	26,9	54	30
EB2275E5FFB1F2C4	1	879	935	951,4	1014	51,2	11	30	862	907	930,8	1014	49,6	87	30
	2	908	973	966,6	1014	40,1	27	30	**876**	**978**	**964,6**	**998**	**43,9**	**99**	**28**
	3	888	947	959,5	1014	37,6	23	30	**931**	**986**	**982,5**	**1014**	**27,8**	**99**	**28**
	4	906	942	955,7	1014	44,1	32	30	932	982	987,6	1014	27,3	42	30
	5	940	967	983,6	1014	31,3	47	30	898	967	972,0	1014	39,2	45	30

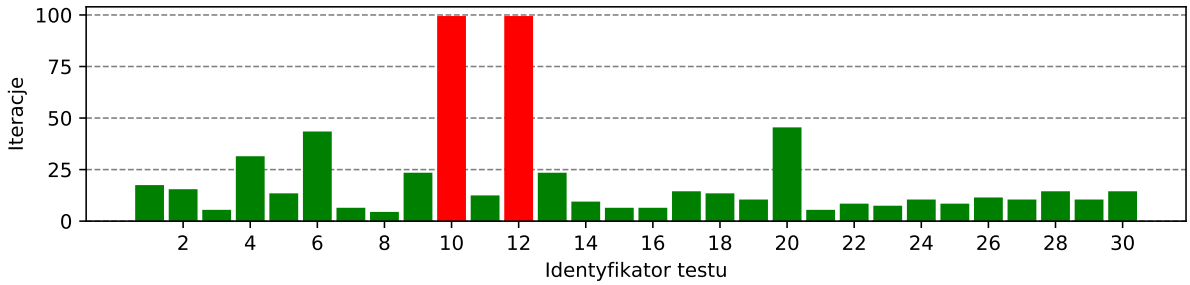
Tabela 6.6: Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* oraz Ω_P^2

Klucz Szyfrujący	Id	*MASA*						*NEA*					
		Min.	Med.	Śr.	Maks	Odch. Std.	t	Bit	Min.	Med.	Śr.	Maks	Odch. Std.	t	Bit
34E9F71A20756231	1	1049	1060	1063,7	1085	11,2	32	30	**857**	**1002**	**993,6**	**1060**	**64,6**	**99**	**28**
	2	882	1000	1002,7	1085	67,1	61	30	1002	1043	1044,8	1085	26,6	38	30
	3	1052	1055	1061,2	1085	12,2	7	30	890	1001	994,9	1085	57,1	14	30
	4	912	989	999,3	1085	63,6	4	30	990	1047	1043,4	1085	27,8	99	30
	5	972	1053	1044,3	1085	33,8	53	30	965	1039	1031,5	1085	37,8	78	30
BF8980377505B539	1	855	959	965,3	1056	71,6	24	30	872	917	938,5	1056	66,4	42	30
	2	884	961	982,7	1056	62,6	40	30	990	1041	1037,4	1056	17,6	40	30
	3	836	958	951,5	1056	73,7	5	30	860	951	957,8	1056	65,6	28	30
	4	854	985	961,4	1056	71,6	7	30	918	1019	1011,6	1056	45,9	82	30
	5	983	991	1009,0	1056	28,3	68	30	**866**	**971**	**959,8**	**1041**	**66,2**	**99**	**27**
0D8E27040A46F6C2	1	908	989	996,9	1051	43,9	31	30	904	983	987,4	1051	43,5	51	30
	2	906	976	995,9	1051	54,5	19	30	916	1010	992,7	1051	49,5	30	30
	3	906	977	975,1	1051	56,1	25	30	974	1027	1022,6	1051	25,7	51	30
	4	892	1020	997,8	1051	45,6	63	30	907	988	982,7	1051	51,2	69	30
	5	976	1020	1018,3	1051	17,1	60	30	837	983	973,7	1051	79,1	18	30
A49ECC6471E38D65	1	978	999	1033,1	1101	45,4	39	30	**978**	**1059**	**1040,2**	**1095**	**48,0**	**99**	**28**
	2	984	1068	1055,9	1101	39,1	52	30	979	1057	1053,2	1101	42,6	64	30
	3	971	1052	1041,5	1101	46,1	32	30	971	1040	1049,1	1101	46,6	72	30
	4	978	1067	1052,7	1101	41,1	26	30	989	1089	1062,1	1101	42,6	11	30
	5	967	1032	1044,3	1101	40,6	61	30	**895**	**1018**	**1013,4**	**1089**	**54,6**	**99**	**26**
EB2275E5FFB1F2C4	1	965	1037	1025,6	1095	38,2	27	30	908	978	993,7	1095	61,6	66	30
	2	941	1054	1036,3	1095	53,5	32	30	**955**	**1048**	**1031,5**	**1074**	**38,7**	**99**	**29**
	3	**912**	**967**	**976,2**	**1048**	**39,0**	**99**	**23**	933	1010	1019,4	1095	50,5	35	30
	4	954	1068	1047,1	1095	40,4	58	30	952	1023	1036,1	1095	45,1	70	30
	5	898	1038	1031,5	1095	57,8	72	30	**927**	**1024**	**1004,6**	**1054**	**48,7**	**99**	**25**

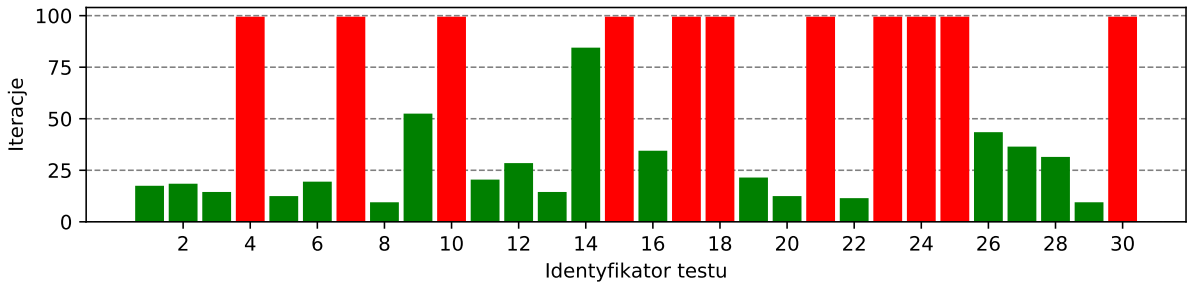
symbolem t, znajdują się wartości odpowiadające numerom iteracji, w których udało się znaleźć właściwy podklucz. W przypadku tego szyfru wszystkie przedstawione wyniki dotyczą zastosowania pierwszej z charakterystyk Ω_P^1.

Prawdopodobieństwo każdej z charakterystyk nie jest w przypadku tego szyfru stuprocentowe. Oznacza to, że pomimo dążenia do jak najlepszej wartości funkcji przystosowania, wartość maksymalna nigdy nie zostanie osiągnięta. Brak możliwości otrzymania tej wartości powoduje, że nie jesteśmy w stanie przerwać działania algorytmu wcześniej, niż po zakończeniu wszystkich ustalonych z góry iteracji.

Na rys. 6.1 oraz 6.2 przedstawiono zestawienie wszystkich poprawnie odgadniętych bitów dla podklucza K_6 dla algorytmów *MASA* i *NEA* oraz pierwszego klucza szyfrującego.



Rysunek 6.1: Zestawienie poprawnie odgadniętych bitów alg. *MASA* - klucz #1

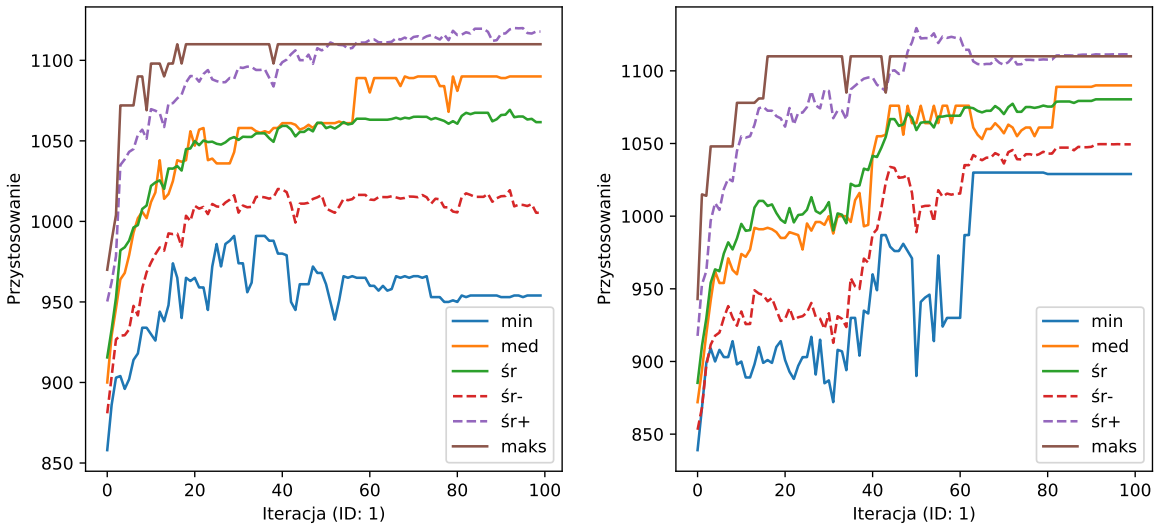


Rysunek 6.2: Zestawienie poprawnie odgadniętych bitów alg. *NEA* - klucz #1

Na podstawie powyższych wykresów słupkowych, oraz wyników przedstawionych w tab. 6.1, można jednoznacznie stwierdzić, że algorytm *MASA* poradził sobie w tym przypadku znacznie lepiej. Atak *MASA* w większości przypadków znajduje właściwy podklucz w pierwszych 25 iteracjach. W trzech przypadkach algorytm znalazł rozwiązanie wykorzystując połowę dostępnych iteracji, natomiast w dwóch pozostałych przypadkach (testy 10 oraz 12), atak nie poradził sobie z szyfrem.

W przypadku algorytmu *NEA* szyfru nie udało się złamać 11 razy - co stanowi ponad 37% wszystkich możliwych podejść. W pozostałych 63% testów udało się złamać algorytmem szyfrującym. W dużej części przypadków udało się odgadnąć poprawny podklucz deszyfrujący posługując się tylko 25 iteracjami algorytmu.

Kolejnym etapem badań było przenalizowania przebiegu wartości funkcji przystosowania za pomocą wykresów zbieżności, które zostały przedstawione na rys. 6.3.

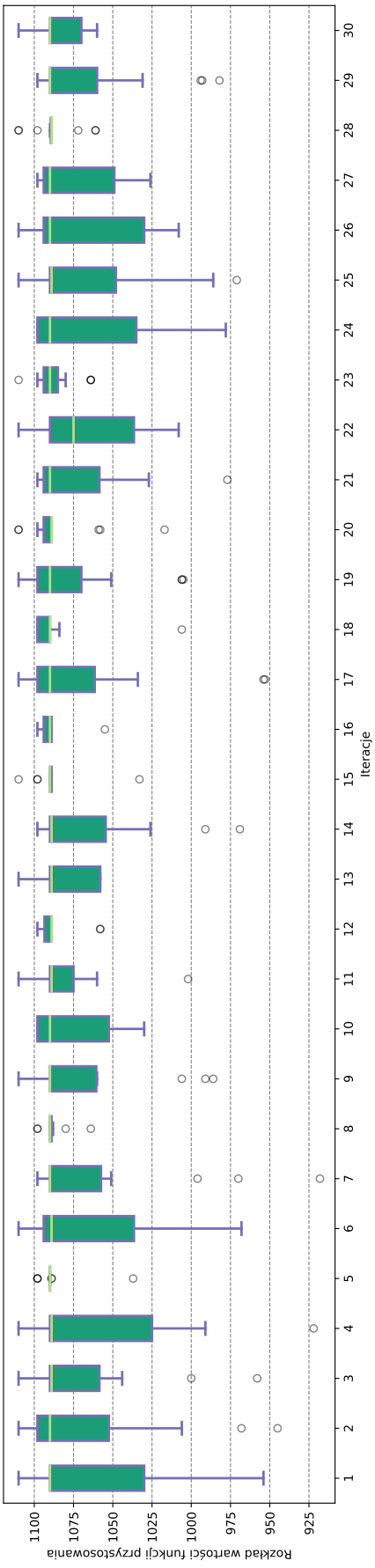


Rysunek 6.3: Przebieg wartości F_f testu #1 dla alg. *MASA* i *NEA* - klucz #1

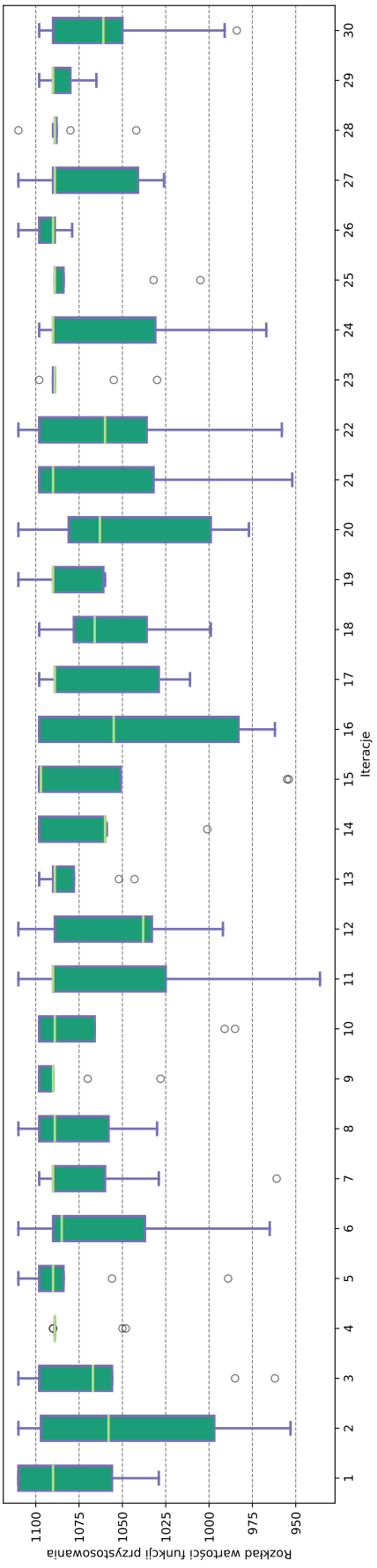
W ramach wykresu, znajdującego się na rys. 6.3, przedstawiono pierwszy przypadek testowy, dla każdego z ataków, z uwzględnieniem wartości minimalnych, maksymalnych, median i średnich - oraz wartości średnich powiększonych i zmniejszonych o wartości odchylenia standardowego funkcji przystosowania. W każdym przypadku zdecydowano się przedstawienie jednego, losowo wybranego eksperymentu, dla którego zaprezentowano wykresy zbieżności każdego algorytmu kryptoanalitycznego.

W przypadku algorytmu *MASA* można zauważyć szybki wzrost maksymalnej wartości F_f już na samym początku działania algorytmu. W dalszych iteracjach zdarzają się pojedyncze spadki tej wartości, aczkolwiek są to pojedyncze iteracje, po zakończeniu których wartość maksymalna zostaje ustabilizowana. W połowie działania algorytmu zaobserwowano gwałtowny wzrost wartości mediany przystosowania. W przypadku algorytmu *NEA* można zauważyć wysoką wartość mediany w połowie zadanych iteracji oraz pod sam koniec działania algorytmu. Wartość minimalna w przypadku pierwszego testu gwałtownie rośnie, natomiast pod sam koniec algorytmu zaczyna się stabilizować.

Analizując wykres zbieżności algorytmu *NEA* można zaobserwować gwałtowny wzrost wartości maksymalnej - podobnie jak to miało miejsce w poprzednim algorytmie war-



Rysunek 6.4: Rozkład wartości F_f dla alg. $MASA$ - klucz #1

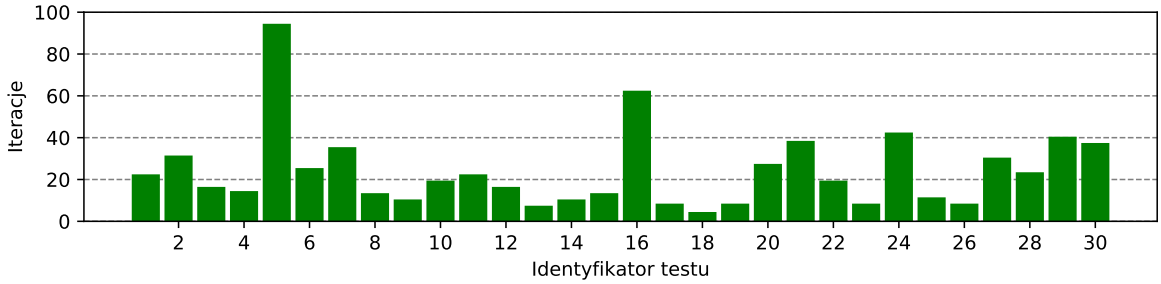


Rysunek 6.5: Rozkład wartości F_f dla alg. NEA - klucz #1

tość ta szybko się stabilizuje. Mediana rośnie wolniej, niż w przypadku algorytmu *MASA*. Pod sam koniec działania ataku zaczyna się ona stabilizować. Na początku pracy algorytmu *NEA* pojawiają się znaczne wahania stanu minimalnej, aczkolwiek od połowy jego działania, zyskuje ona bardzo dużą wartość, która bez wątpienia ma wpływ na wzrost średniej.

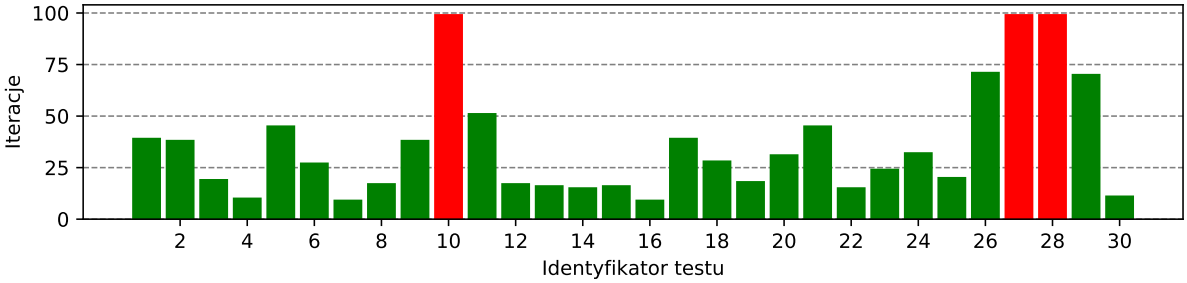
Ostatnim etapem badań był przegląd rozkładu wartości funkcji przystosowania w ostatniej iteracji każdego z ataków - rozkład został zaprezentowany na rys. 6.4 oraz 6.5. W przypadku algorytmu *MASA* niektóre z testów (np. 5, 8 lub 28) charakteryzują się wysokim stopniem jednorodności, co oznacza, że populacja odznacza się niską różnorodnością osobników. W pozostałych eksperymentach można zauważyć dużą wartość mediany, dążącą do wartości maksymalnej, a tym samym właściwego podklucza. Analizując algorytm *NEA* można zaobserwować większy stopień różnorodności pomiędzy osobnikami, o czym bez wątpienia informuje wartość mediany, zmieniająca swoje położenie pomiędzy pierwszym, a trzecim kwartylem.

Na kolejnym wykresie przedstawiono zestawienie liczby poprawnie odgadniętych bitów dla obydwu ataków oraz drugiego klucza szyfrującego - mowa o rys. 6.6 oraz 6.7.



Rysunek 6.6: Zestawienie poprawnie odgadniętych bitów alg. *MASA* - klucz #2

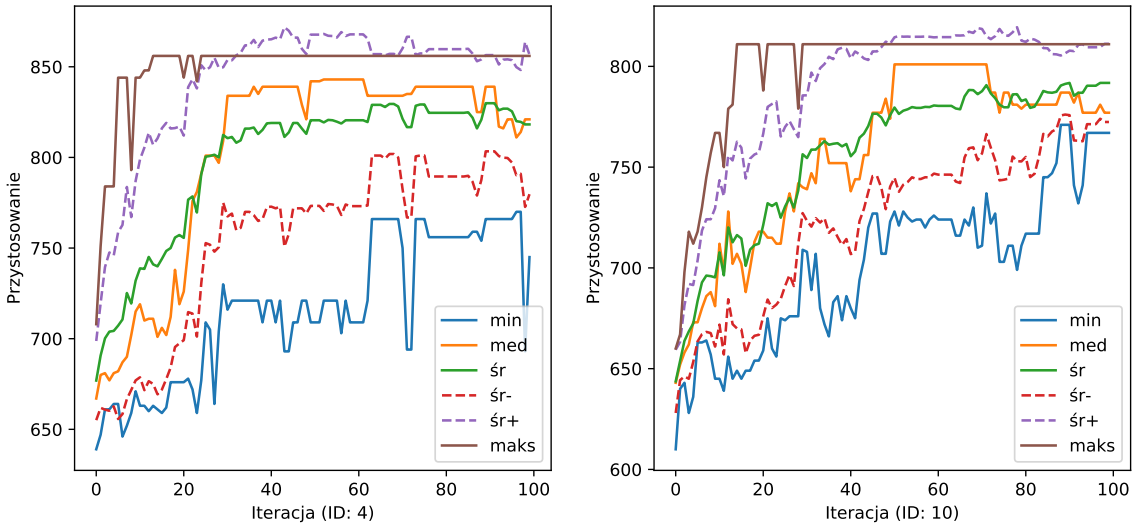
Analizując wspomniane wykresy słupkowe można zauważyć znaczną poprawę skuteczności jednego i drugiego ataku kryptoanalitycznego. Algorytm *MASA* za każdym razem znalazł właściwy podklucz. W połowie przypadków udało mu się to osiągnąć już w 20% wszystkich dostępnych iteracji. Na szczególną uwagę zasługuje 5 przypadek, w którym pod sam koniec działania algorytmu również udało się znaleźć poprawne rozwiązanie.



Rysunek 6.7: Zestawienie poprawnie odgadniętych bitów alg. *NEA* - klucz #2

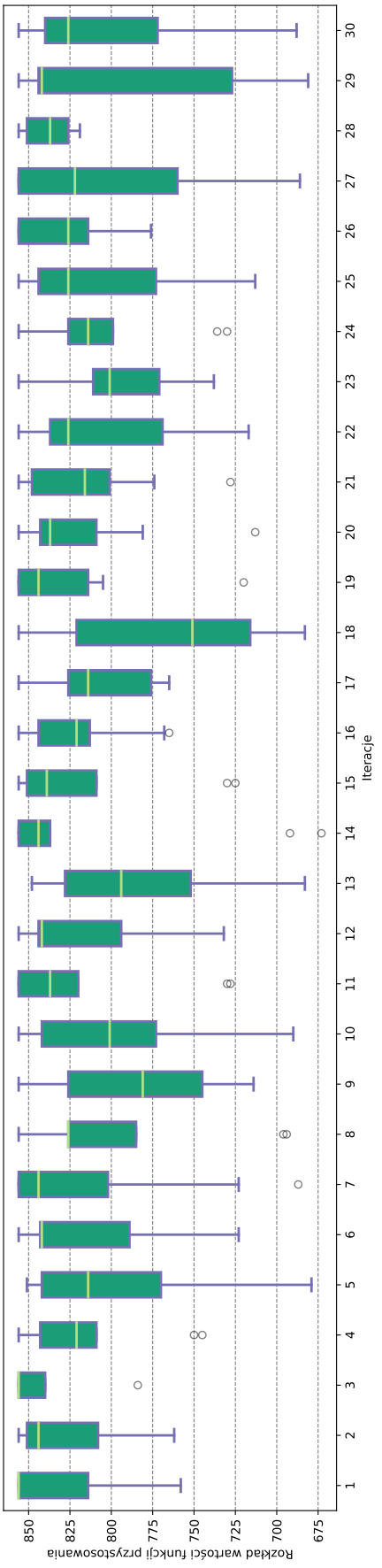
W przypadku ataku *NEA* mamy znacznie większą skuteczność, niż jak to miało miejsce w poprzednich eksperymentach. Poprawnego podklucza nie udało się złamać tylko w 3 przypadkach, co stanowi 10% wszystkich możliwych prób. Zazwyczaj do 50 iteracji algorytmu możemy mówić o poprawnie wykonanej kryptoanalize podklucza K_6.

Na rys. 6.8 zaprezentowano przebieg wartości funkcji przystosowania dla drugiego klucza szyfrującego. Tym razem zdecydowano się zaprezentować test 5 dla algorytmu *MASA* oraz 10 dla algorytmu *NEA*.

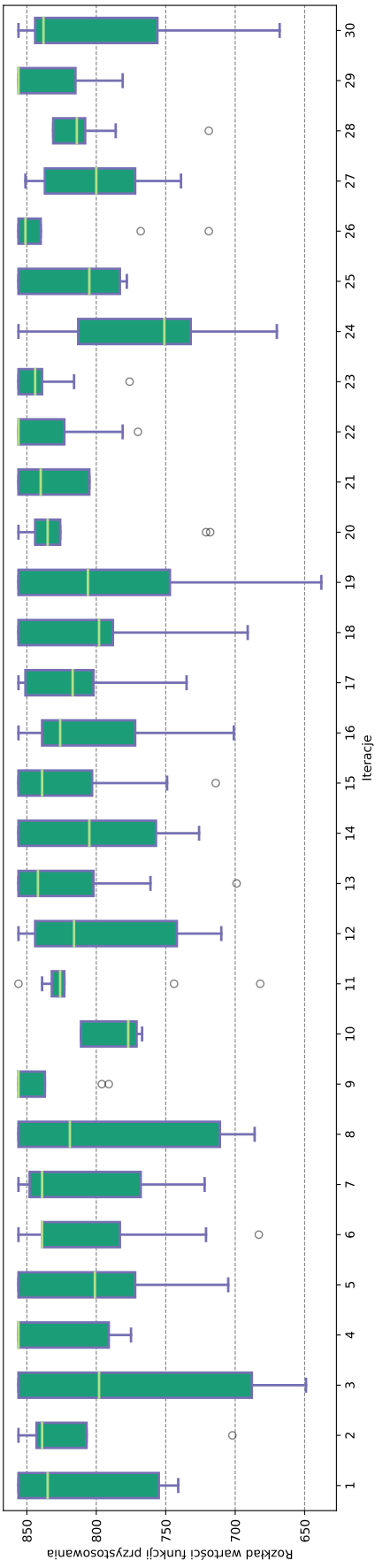


Rysunek 6.8: Przebieg wartości F_f testów #4 i #10 dla alg. *MASA* i *NEA* - klucz #2

Analizując przypadek 5, dla ataku *MASA*, warto zwrócić uwagę na wartość maksymalną. Od samego początku można zauważyć szybki wzrost, aczkolwiek nie tak nagły, jak w przypadku poprzedniego eksperymentu. Już w 20 iteracji maksimum zaczyna się stabilizować - wyjątek stanowi minimalny wzrost w około 30 iteracji i tak zostaje prawie do końca algorytmu. W około 80 iteracji dochodzi do wybicia z maksimum lokalnego. W przebiegu mediany można zauważyć duże wahania podczas działania algorytmu. Wartość minimalna pod sam koniec ulega znacznemu spadkowi.



Rysunek 6.9: Rozkład wartości F_f dla alg. $MASA$ - klucz #2

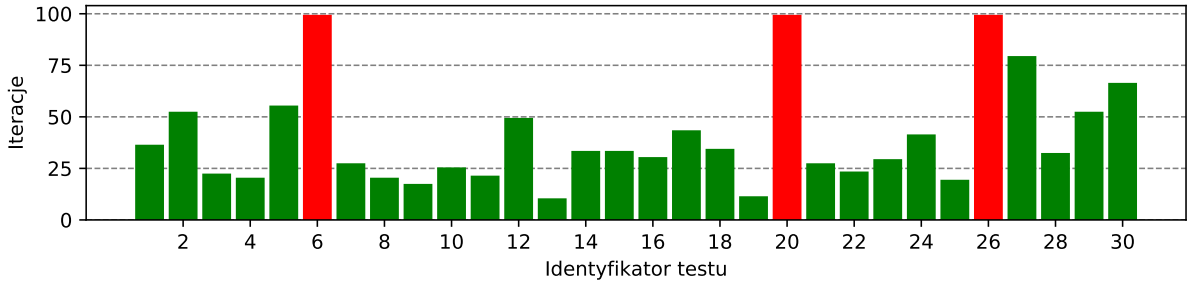


Rysunek 6.10: Rozkład wartości F_f dla alg. NEA - klucz #2

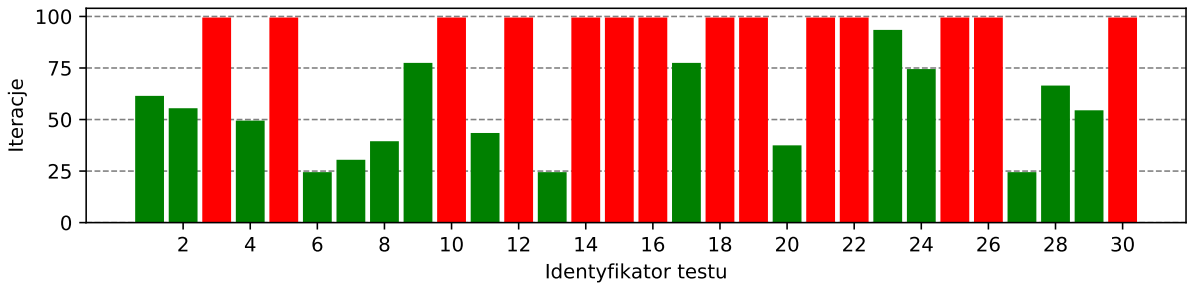
Analizując przebieg dla algorytmu *NEA*, odpowiadający testowi 10, można zauważyć sytuację, w której nie udało się odgadnąć właściwego podklucza K_6. Już na początku działania algorytmu można zaobserwować gwałtowny wzrost wartości maksymalnej - na tym etapie osiągnięto optimum lokalne, którego niestety nie udało się opuścić.

Na rys. 6.9 oraz 6.10 przedstawiono rozkład wartości funkcji przystosowania omawianych algorytmów dla drugiego rozpatrywanego klucza. Analizując algorytm *MASA* można zauważyć dużą różnorodność osobników w populacji.

Zestawienie poprawnie odgadniętych bitów dla algorytmów *MASA* oraz *NEA* dla klucza trzeciego przedstawiono na rys. 6.11 i 6.12.

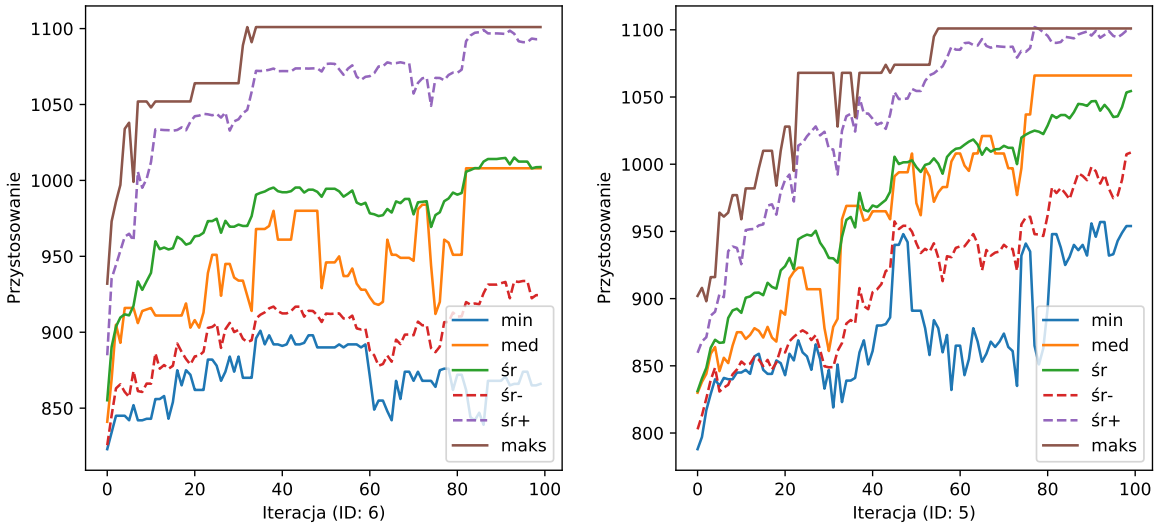


Rysunek 6.11: Zestawienie poprawnie odgadniętych bitów alg. *MASA* - klucz #3



Rysunek 6.12: Zestawienie poprawnie odgadniętych bitów alg. *NEA* - klucz #3

Skuteczność algorytmu *MASA* wynosiła w tym przypadku 90% (nie udało się znaleźć poprawnego rozwiązania w 3 przypadkach), natomiast w skuteczność ataku *NEA* była wyjątkowo niska, mianowicie 53% (w przypadku 14 eksperymentów nie udało się odgadnąć poprawnego podklucza). Na przedstawionym poniżej rys. 6.13 zaprezentowano przebieg wartości F_f dla testów 6 dla algorytmu *MASA* oraz 5 dla ataku *NEA*.



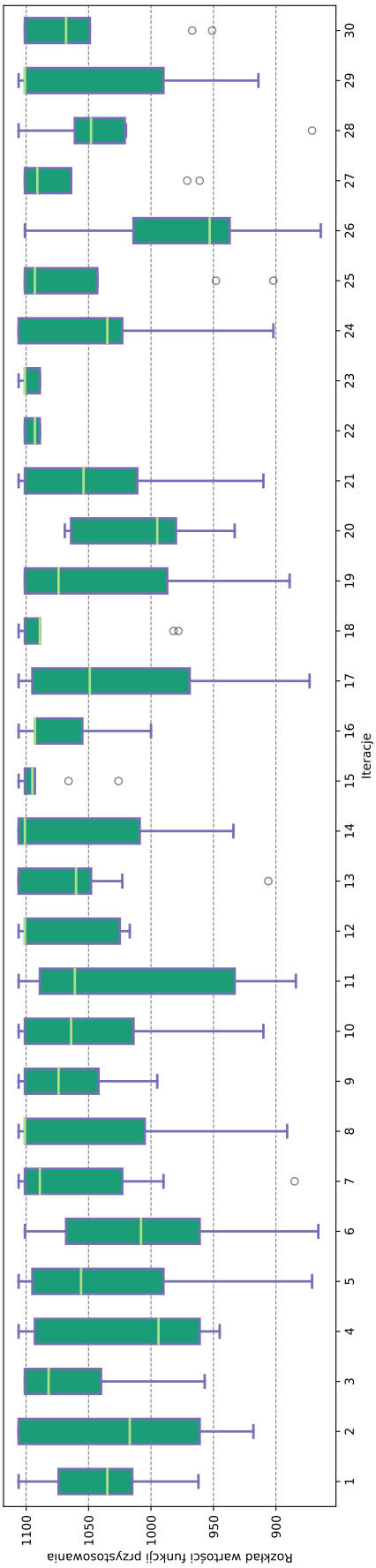
Rysunek 6.13: Przebieg wartości F_f testów #6 i #5 dla alg. *MASA* i *NEA* - klucz #3

W przypadku algorytmu *MASA* można zauważyć, spotykany już wcześniej, gwałtowny wzrost wartości maksymalnej - spowodowany prawdopodobnie obecnością procesu lokalnej optymalizacji. W okolicy 30 iteracji widoczny jest również znaczny wzrost wartości minimalnej oraz mediany. Niestety, mimo stopniowego zwiększania wartości maksymalnej, początkowo w około 20, a później w około 30 iteracji, nie udało się uzyskać właściwego podklucza szyfrującego.

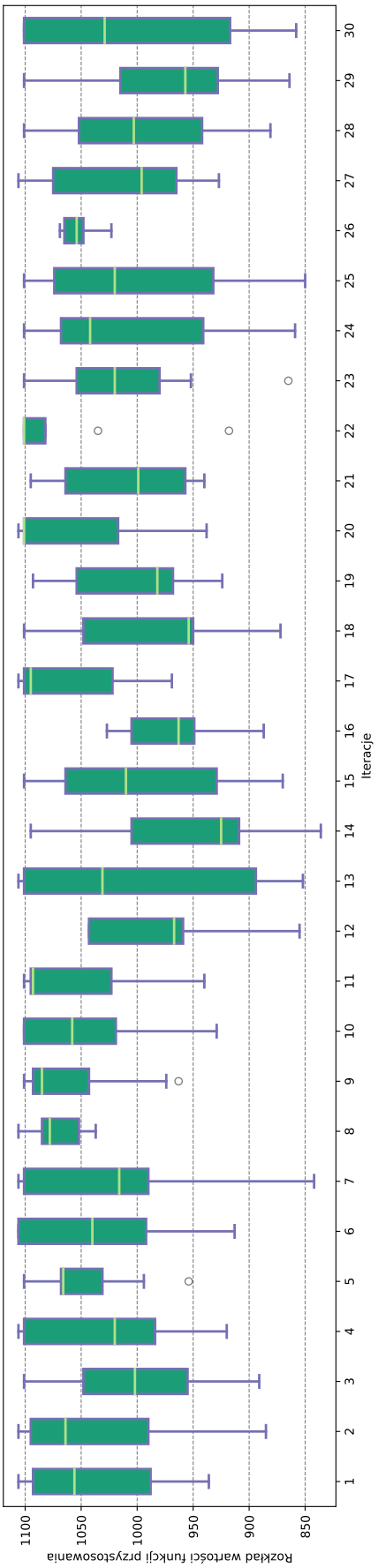
Analizując eksperyment piąty, dla algorytmu *NEA*, wzrost wartości maksymalnej jest znacznie wolniejszy. Co kilka iteracji można zauważyć minimalny wzrost, aż do 20 iteracji, gdzie skok jest bardziej widoczny. Następnie, gdzieś w okolicach 50 iteracji widoczny jest znaczny wzrost wartości maksimum, aczkolwiek to dalej za mało, aby odgadnąć właściwy podklucz. Ciekawą anomalią jest nagły wzrost wartości minimalnej w około 45 iteracji.

Na rys. 6.14 oraz rys. 6.15 przedstawiono rozkład wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla trzeciego klucza deszyfrującego. Podobnie, jak to miało miejsce w poprzednim przykładzie, atak *MASA* charakteryzuje się wysoką różnorodnością osobników w populacji, jednakże w kwestii algorytmu *NEA* można zauważyć zbieżność populacji do najbardziej przystosowanego osobnika.

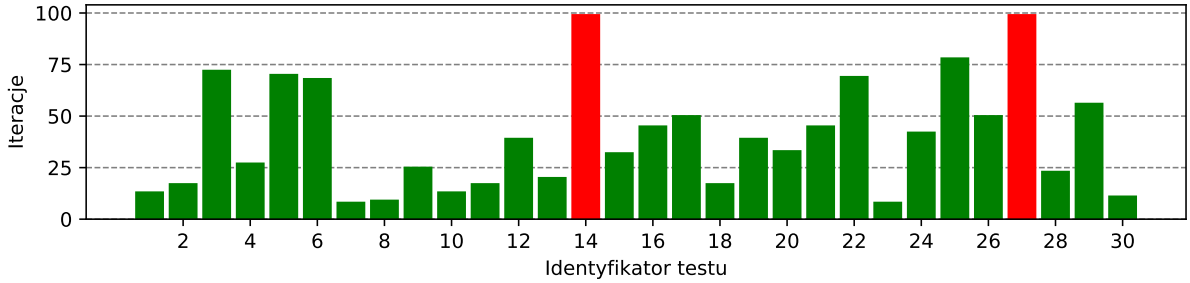
Na rys. 6.16 oraz rys. 6.17 zaprezentowano porównanie wszystkich poprawnie odgadniętych bitów dla czwartego klucza szyfrującego, zarówno dla ataku *MASA*, jak i dla ataku *NEA*.



Rysunek 6.14: Rozkład wartości F_f dla alg. $MASA$ - klucz #3

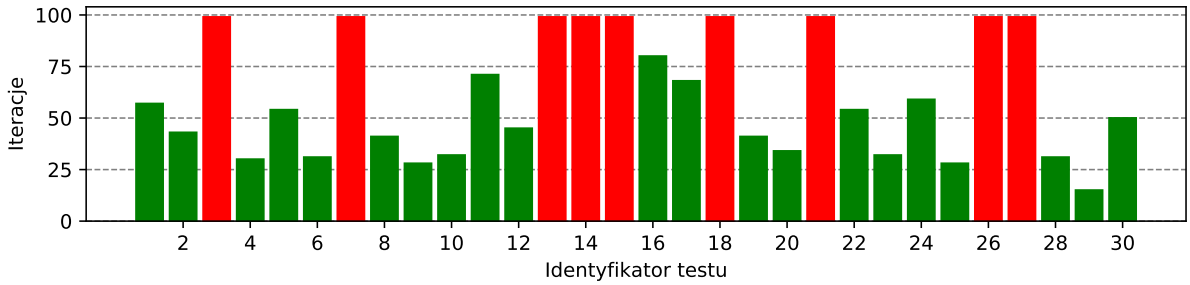


Rysunek 6.15: Rozkład wartości F_f dla alg. NEA - klucz #3



Rysunek 6.16: Zestawienie poprawnie odgadniętych bitów alg. *MASA* - klucz #4

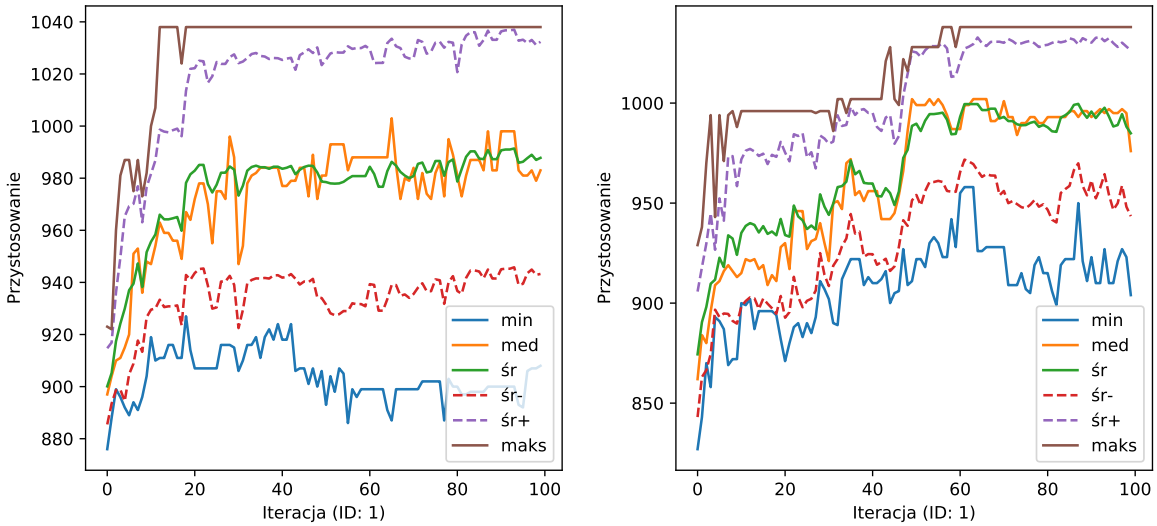
Sytuacja wygląda podobnie, jak w przypadku poprzednich eksperymentów. Skuteczność ataku *MASA* wynosi ponad 93%. Nie udało się odgadnąć poprawnego podklucza w 2 przypadkach testowych).



Rysunek 6.17: Zestawienie poprawnie odgadniętych bitów alg. *NEA* - klucz #4

Algorytm *NEA* poradził sobie całkiem nieźle, aczkolwiek, w porównaniu do ataku *MASA*, skuteczność na poziomie 70% (21 poprawnie rozpoznanych podkluczy) nie jest obiecującym wynikiem.

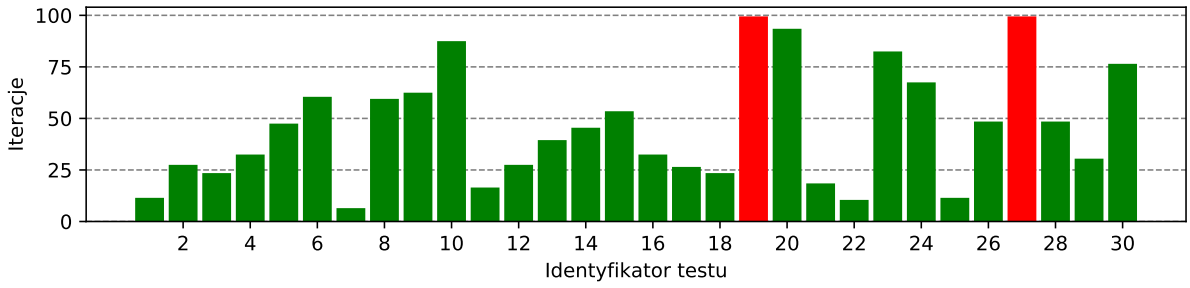
Na rys. 6.18 przedstawiono przebieg wartości funkcji przystosowania dla pierwszego testu obydwu algorytmów. Po wstępnej analizie, w przypadku algorytmu *MASA*, można zauważyć standardowy gwałtowny wzrost wartości maksymalnej funkcji przystosowania. Na uwagę zasługuje początkowo zwiększająca się wartość minimalna funkcji, która od około 40 iteracji zaczyna ulegać znacznemu spadkowi, co bez wątpliwości ma wpływ na przebieg wszystkich wartości średnich. Już od 10 iteracji możemy mówić o ustabilizowaniu się algorytmu, jeżeli chodzi o wartość maksymalną funkcji przystosowania.



Rysunek 6.18: Przebieg wartości F_f testu #1 dla alg. *MASA* i *NEA* - klucz #4

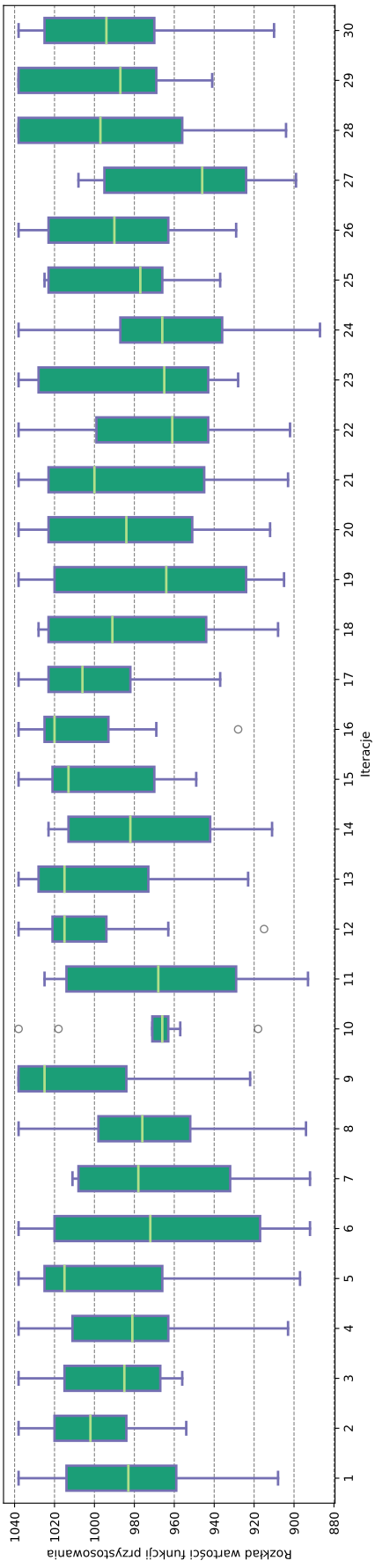
W przypadku ataku *NEA* na samym początku działania algorytmu, wartość maksimum F_f ma charakter skokowy. Dopiero w około 10 iteracji dochodzi do ustabilizowania, aczkolwiek już w 30, a następnie w 40 i 50 iteracji dochodzi do znacznego wzrostu wartości maksymalnej wartości funkcji przystosowania. Od 50 iteracji można zauważyć gwałtowny wzrost wartości mediany, który utrzymuje się prawie do końca działania algorytmu - na samym końcu dochodzi do gwałtownego spadku wartości minimalnej funkcji, a co za tym idzie wszystkich pozostałych metryk, poza wartością maksymalną. Na rys. 6.19 oraz rys. 6.20 zaprezentowano rozkład wartości F_f dla czwartego klucza szyfrującego.

W dalszej sekcji przestawiono ostatnie zestawienie poprawnie odgadniętych bitów dla ataku *MASA* oraz *NEA* dla piątego klucza szyfrującego (rys. 6.21 oraz rys. 6.22).

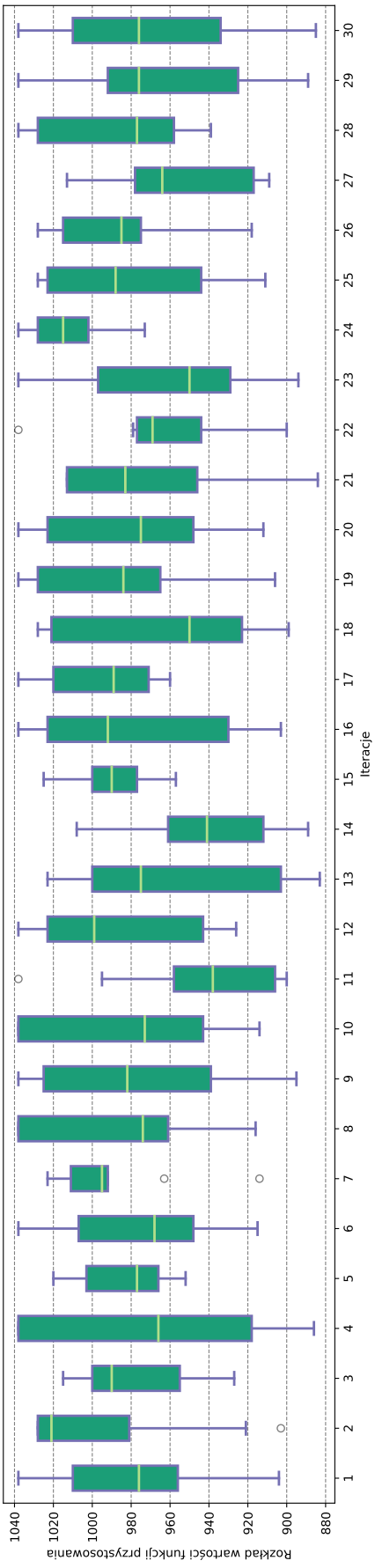


Rysunek 6.21: Zestawienie poprawnie odgadniętych bitów alg. *MASA* - klucz #5

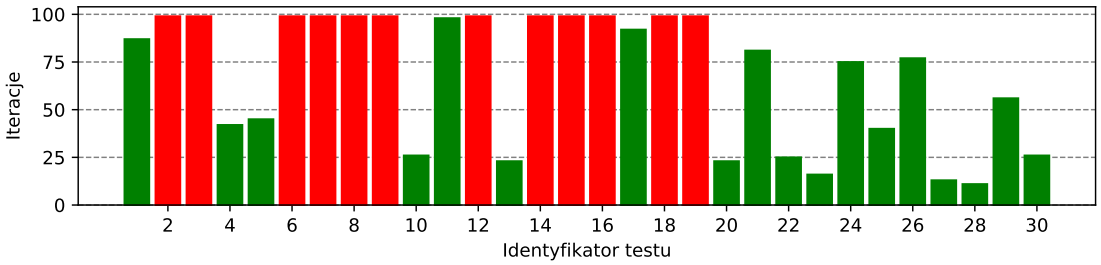
Niemalże identycznie, jak to miało miejsce w poprzednim zestawie eksperymentów, algorytm *MASA* znalazł poprawny klucz deszyfrujący w ponad 93% przypadków (nie udało się znaleźć idealnego rozwiązania tylko w dwóch sytuacjach).



Rysunek 6.19: Rozkład wartości F_f dla alg. $MASA$ - klucz #4

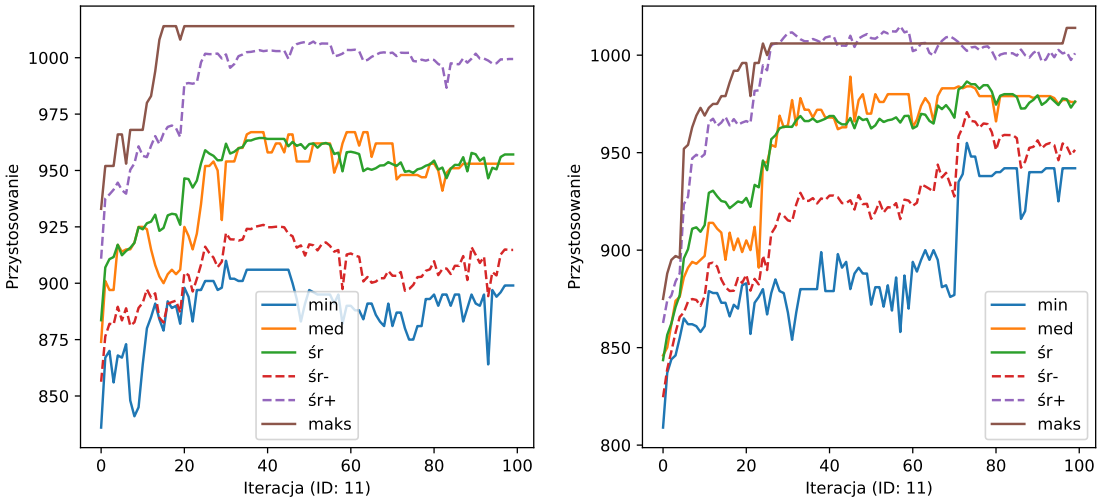


Rysunek 6.20: Rozkład wartości F_f dla alg. NEA - klucz #4



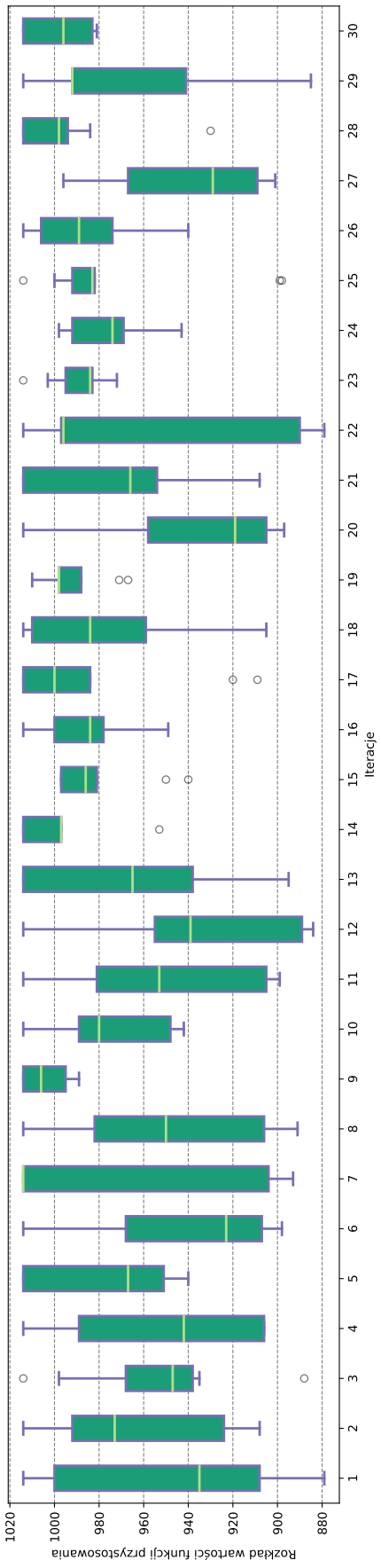
Rysunek 6.22: Zestawienie poprawnie odgadniętych bitów alg. *NEA* - klucz #5

Algorytm *NEA* uzyskał skuteczność na poziomie 60%. W 12 przypadkach nie udało się odgadnąć poprawnego klucza deszyfrującego. Na szczególną uwagę zasługują również testy o identyfikatorach: 1, 11 oraz 17. Można w nich zauważyć bardzo dużą liczbę wykorzystanych iteracji (w niektórych przypadkach ponad 90), co oznacza, że algorytm *NEA* pod sam koniec działania algorytmu znalazł rozwiązanie. Na rys. 6.23 przedstawiono przebieg wartości funkcji przystosowania dla testu 11 każdego z algorytmów.

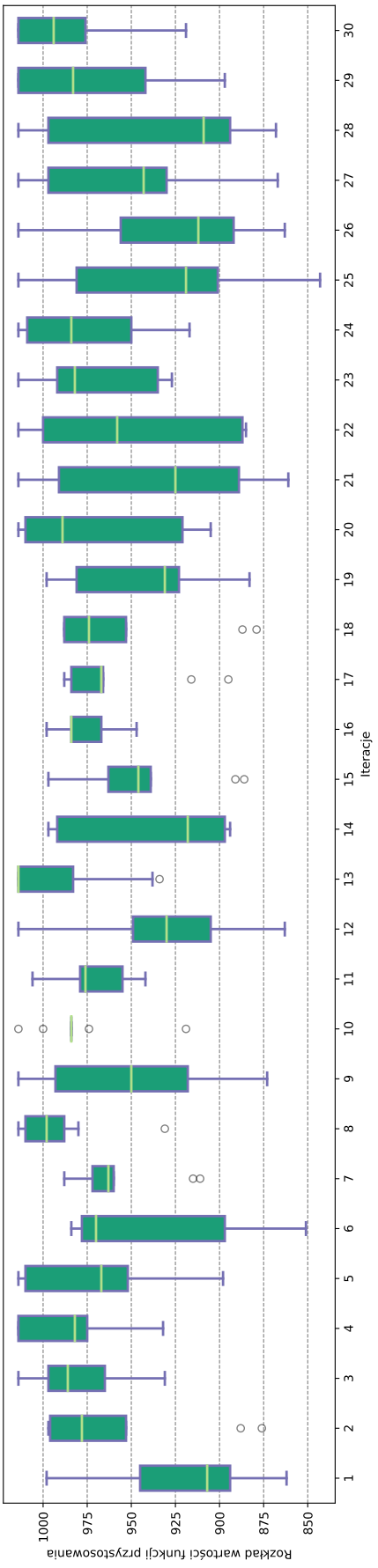


Rysunek 6.23: Przebieg wartości F_f testu #11 dla alg. *MASA* i *NEA* - klucz #5

W przypadku algorytmu *MASA* mamy do czynienia z gwałtownym wzrostem wartości maksymalnej F_f oraz odgadnięciem właściwego rozwiązania już w początkowych iteracjach algorytmu. Analizując wykres zbieżności ataku *NEA* można zauważyć utknięcie algorytmu od około 25 do 95 iteracji na tej samej wartości maksymalnej wartości funkcji przystosowania. Dopiero pod sam koniec algorytm opuszcza optimum lokalne, dzięki czemu udaje się odgadnąć właściwe rozwiązanie. Ciekawą anomalię można zauważyć w przebiegu wartości minimum od około 70 iteracji algorytmu. Na rys. 6.24 oraz 6.25 przedstawiono ostatnie porównanie rozkładu wartości funkcji przystosowania dla algorytmów *MASA* oraz *NEA*.



Rysunek 6.24: Rozkład wartości F_f dla alg. $MASA$ - klucz #5



Rysunek 6.25: Rozkład wartości F_f dla alg. NEA - klucz #5

6.3.2 Kryptoanaliza z wykorzystaniem ataku *DPSO*

W tym podrozdziale przedstawiono szczegółowe wyniki algorytmu *DPSO*. Ze względu na bardzo zbliżone wartości zarówno dla pierwszego, jak i dla drugiego tekstu jawnego, zdecydowano się na przedstawienie wyników jedynie dla fragmentu artykułu opisującego życie w Korei Południowej w trakcie trwania pandemii Koronawirusa.

W tab. 6.7 zaprezentowano wartości funkcji przystosowania F_f dla algorytmu *DPSO*. Podobnie jak to miało miejsce w przypadku ataków *NEA* oraz *MASA* algorytm przetestowano na pięciu kluczach szyfrujących. W tab. 6.7 znajdującej się poniżej przedstawiono pięć pierwszych wyników dla każdego klucza.

Tabela 6.7: Wartości funkcji przystosowania algorytmu *DPSO* dla szyfru *DES*

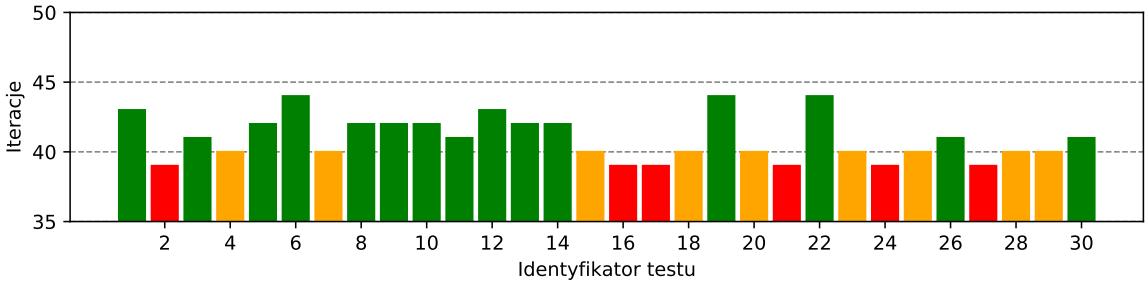
Klucz Szyfrujący	Id	*DPSO*					
		Min.	Med.	Śr.	Max.	Odch. Std.	Bit.
34E9F71A20756231	1	52567,4	52755,2	52755,3	52879,6	64,6	43
	2	**52624,2**	**52744,8**	**52745,7**	**52858,4**	**58,6**	**39**
	3	52598,6	52724,8	52726,4	52865,8	62,1	41
	4	52626,0	52749,4	52748,0	52888,8	63,7	40
	5	52614,4	52747,0	52746,2	52936,8	63,1	42
BF8980377505B539	1	52609,6	52735,8	52730,3	52881,8	62,8	43
	2	52611,2	52724,2	52726,4	52903,2	60,1	43
	3	**52602,0**	**52746,8**	**52742,5**	**52867,6**	**66,3**	**39**
	4	52617,4	52719,2	52729,2	52854,4	63,6	42
	5	**52542,4**	**52722,8**	**52719,1**	**52922,0**	**75,5**	**39**
0D8E27040A46F6C2	1	**52600,8**	**52731,6**	**52733,9**	**52864,6**	**63,9**	**39**
	2	52600,8	52735,0	52741,3	52871,8	64,9	40
	3	52602,8	52749,2	52743,3	52918,0	69,5	40
	4	**52569,4**	**52751,2**	**52744,3**	**52955,8**	**80,2**	**39**
	5	52593,0	52736,8	52740,7	52885,6	63,5	40
A49ECC6471E38D65	1	52538,4	52716,4	52719,9	52856,2	62,6	40
	2	52632,4	52743,4	52744,4	52889,2	63,2	42
	3	52635,2	52759,2	52753,5	52899,0	63,0	41
	4	52617,8	52740,0	52736,2	52882,2	57,2	41
	5	52506,0	52726,4	52728,0	52880,0	73,4	40
EB2275E5FFB1F2C4	1	52637,2	52737,2	52739,2	52848,6	51,2	44
	2	52590,2	52712,6	52729,8	52907,6	73,4	45
	3	52626,0	52737,2	52739,6	52859,6	62,4	44
	4	**52597,4**	**52742,2**	**52735,8**	**52900,0**	**60,2**	**39**
	5	**52588,2**	**52757,4**	**52749,2**	**52916,4**	**69,1**	**39**

Algorytm został przebadany z wykorzystaniem każdego z pięciu kluczy szyfrują-

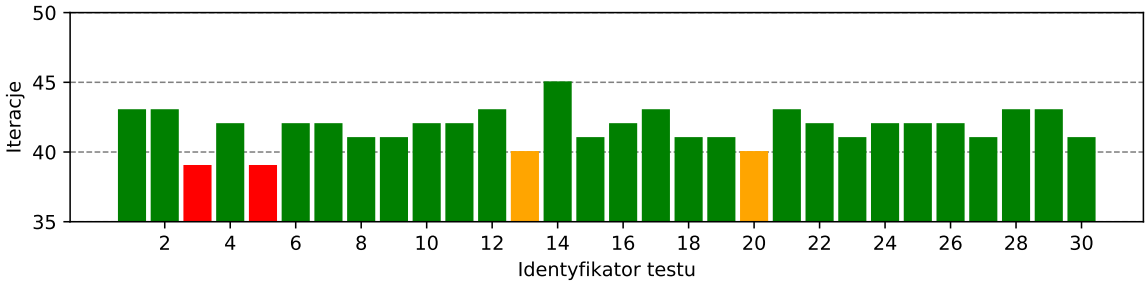
cych wyszczególnionych w tab. 6.1. Pełne zestawienie wszystkich przeprowadzonych eksperymentów można znaleźć w dodatku A.

W przypadku powyższej tab.6.7 zdecydowano się pominąć kolumnę t - algorytm *DPSO* ani razu nie odgadnął poprawnego klucza szyfrującego. W najlepszych uruchomieniach tego ataku udało się znaleźć około 43-45 bitów klucza K. W powyższej tabeli, pogrubioną czcionką, zostały wyróżnione najsłabsze eksperymenty, podczas których nie udało się odgadnąć nawet 40 bitów.

Na rys. 6.26 oraz 6.27 przedstawiono zestawienie wszystkich poprawnie odgadniętych bitów klucza K w poszczególnych eksperymentach algorytmu *DPSO* dla dwóch pierwszych kluczy:



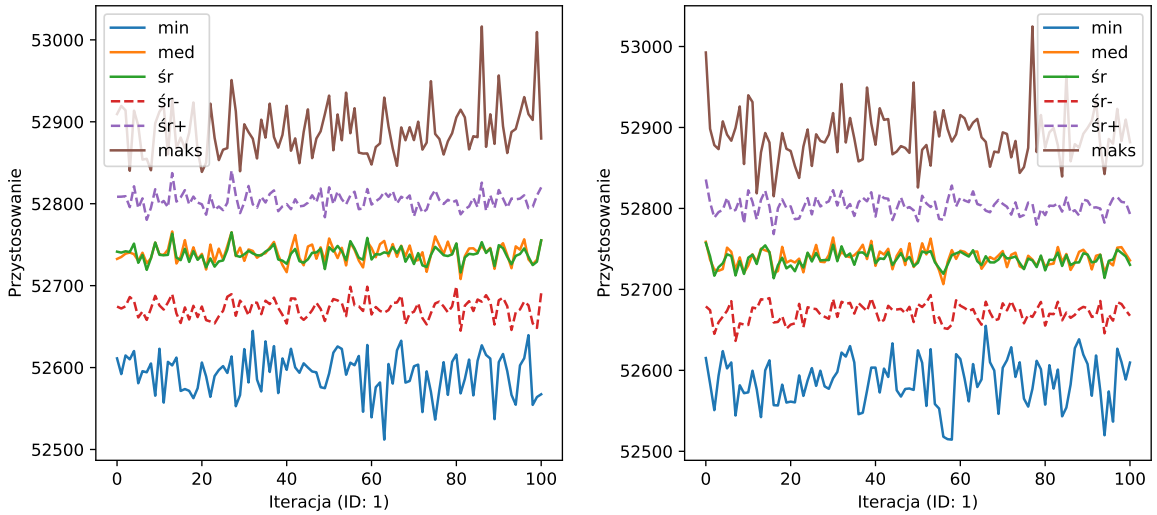
Rysunek 6.26: Zestawienie poprawnie odgadniętych bitów alg. *DPSO* - klucz #1



Rysunek 6.27: Zestawienie poprawnie odgadniętych bitów alg. *DPSO* - klucz #2

Kolorem czerwonym zostały oznaczone eksperymenty, w przypadku których nie udało się znaleźć nawet 40 bitów klucza K (70% wszystkich dostępnych bitów). Kolorem pomarańczowym wyróżniono testy „graniczne", w których udało się odgadnąć dokładnie 40 bitów, natomiast kolorem zielonym przypadki, w których udało się przekroczyć postawiony próg 70% bitów. Na podstawie przedstawionych powyżej wykresów słupkowych można zauważyć, że najlepszy możliwy wynik udało się osiągnąć w przypadku czternastego testu dla drugiego klucza - 45 bitów. Na ogół algorytm znajduje od około 38 - 43 bitów klucza.

W dalszym etapie badań przeanalizowano przebieg wartości funkcji przystosowania. Wykresy zbieżności dla algorytmu *DPSO* przedstawione zostały na rys. 6.28:

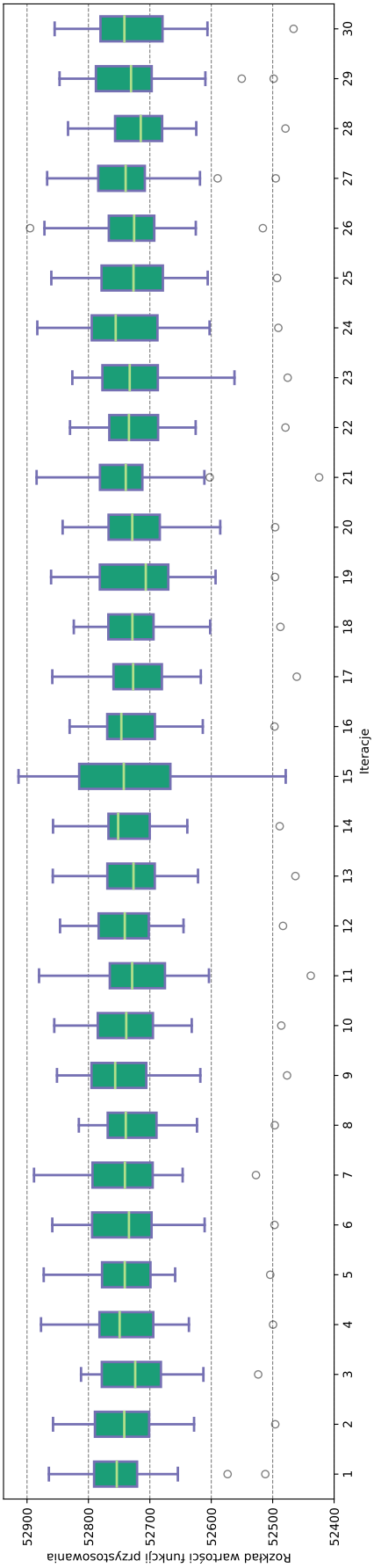


Rysunek 6.28: Przebieg wartości F_f testu #1 dla alg. *DPSO* - klucz #1 i #2

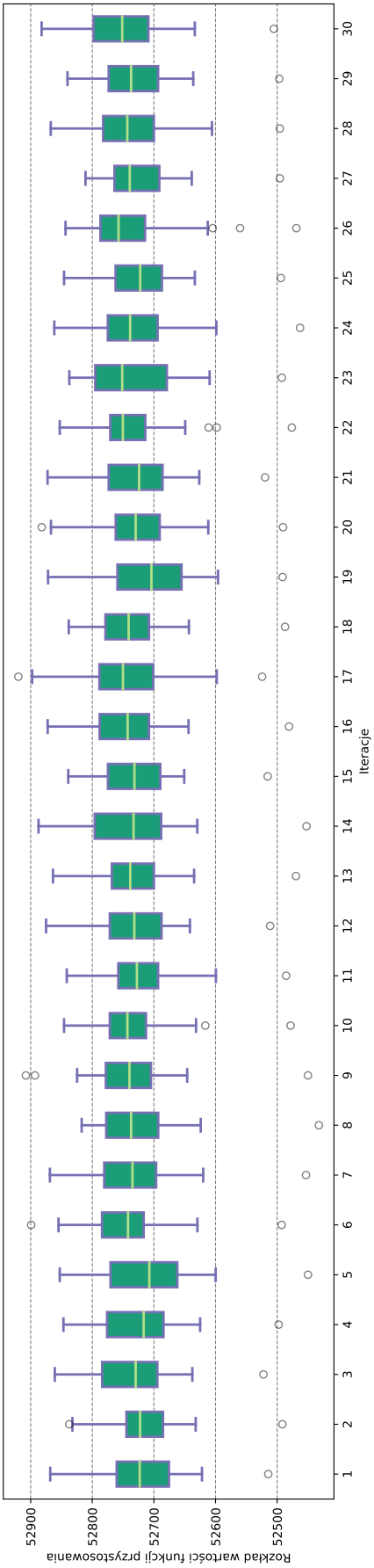
Na powyższych wykresach zaprezentowano wykres zbieżności dla pierwszego eksperymentu oraz dla dwóch pierwszych kluczy szyfrujących, przedstawionych w tab. 6.1. W ramach wykresu uwzględniono wartości minimalne, maksymalne, medianę oraz średnie arytmetyczne kolejno powiększone oraz zmniejszone o wartości odchylenia standardowego.

W przypadku jednego i drugiego klucza można zauważyć straszne wahania podczas pracy algorytmu. Każda z przedstawionych na wykresie wartości nie jest w stanie się ustabilizować - wydawać by się mogło, że algorytm *DPSO* działa w sposób losowy.

Na ostatnim etapie badań przeanalizowano rozkład wartości funkcji przystosowania w ostatniej iteracji algorytmu *DPSO* - rozkład dla pierwszego i drugiego klucza został zaprezentowany kolejno na rys. 6.29 oraz 6.30. Analizując zamieszczone wykresy można doszukać się dużego podobieństwa pomiędzy poszczególnymi rozkładami. Oznaczać to może, że populacja końcowa, w niemalże każdym z eksperymentów, odznacza się niskim stopniem różnorodności osobników.

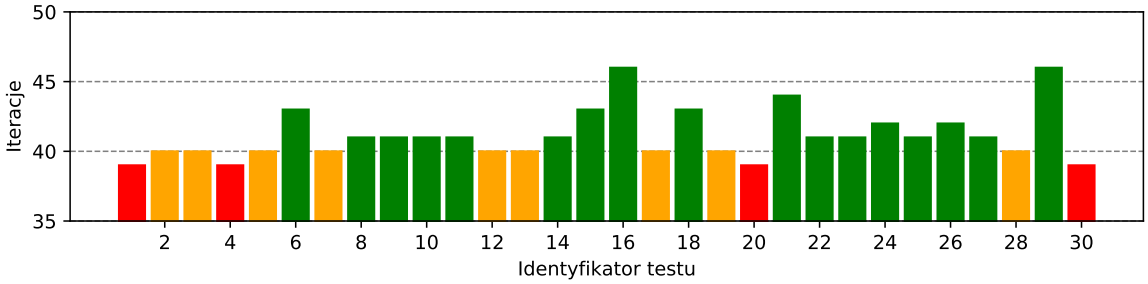


Rysunek 6.29: Rozkład wartości F_f dla alg. *DPSO* - klucz #1

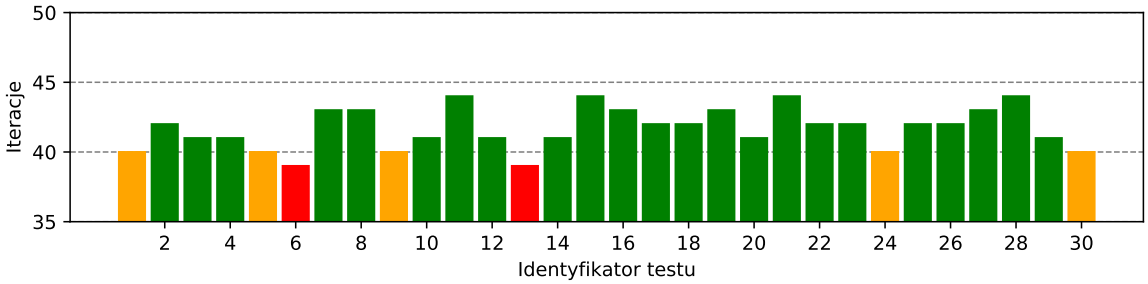


Rysunek 6.30: Rozkład wartości F_f dla alg. *DPSO* - klucz #2

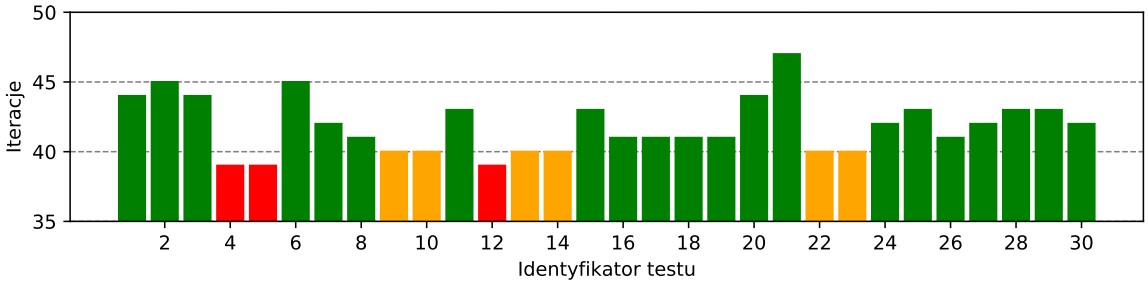
Na rys. 6.31, 6.32 oraz 6.33 przedstawiono zestawienie liczby poprawnie odgadniętych bitów dla trzech kolejnych kluczy szyfrujących:



Rysunek 6.31: Zestawienie poprawnie odgadniętych bitów alg. *DPSO* - klucz #3



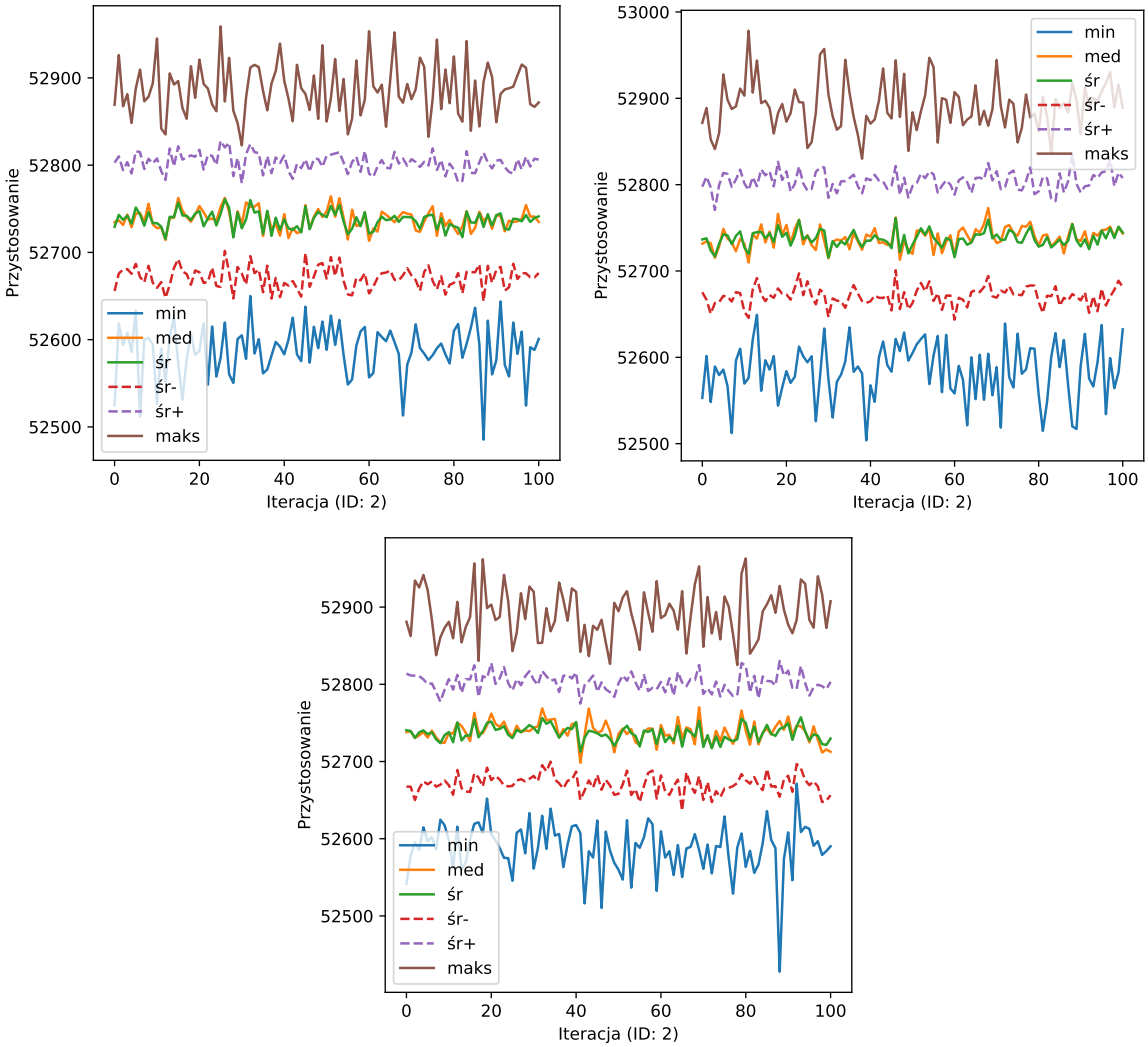
Rysunek 6.32: Zestawienie poprawnie odgadniętych bitów alg. *DPSO* - klucz #4



Rysunek 6.33: Zestawienie poprawnie odgadniętych bitów alg. *DPSO* - klucz #5

Najlepsze wartości udało się uzyskać dla eksperymentów: 16 oraz 29 (klucz 3), gdzie liczba odgadniętych bitów osiągnęła poziom 46, oraz dla eksperymentu numer 21 (klucz 5), gdzie udało się odszukać 47 bitów klucza szyfrującego. W większości przypadków algorytm odgaduje 40 - 43 bitów. Niestety, w żadnym z wyszczególnionych eksperymentów nie udało się odgadnąć poprawnego klucza deszyfrującego.

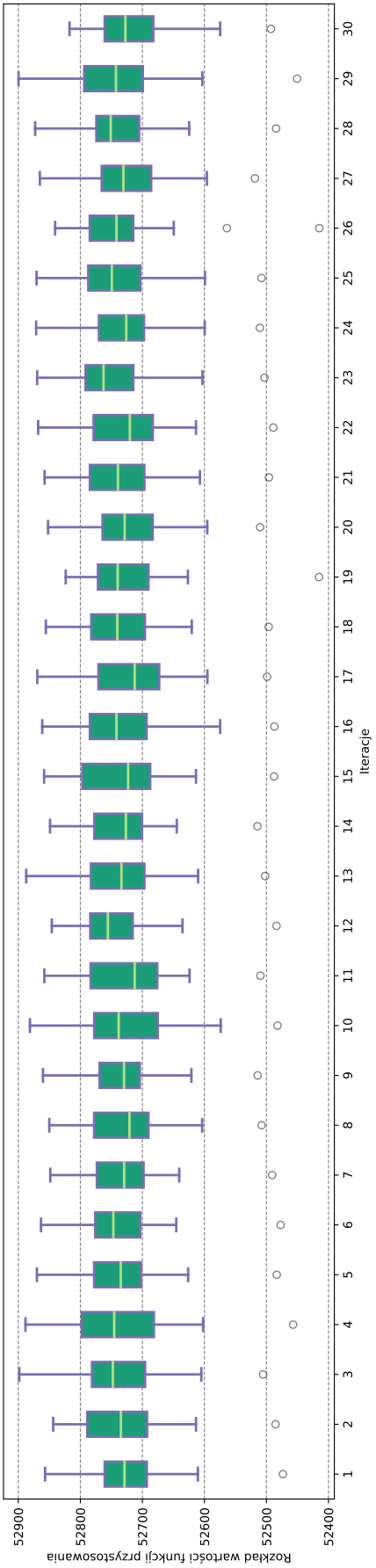
Na rys. 6.34 przedstawiono kolejne wykresy zbieżności, w tym przypadku dla eksperymentu drugiego, dla trzech kolejnych kluczy szyfrujących:



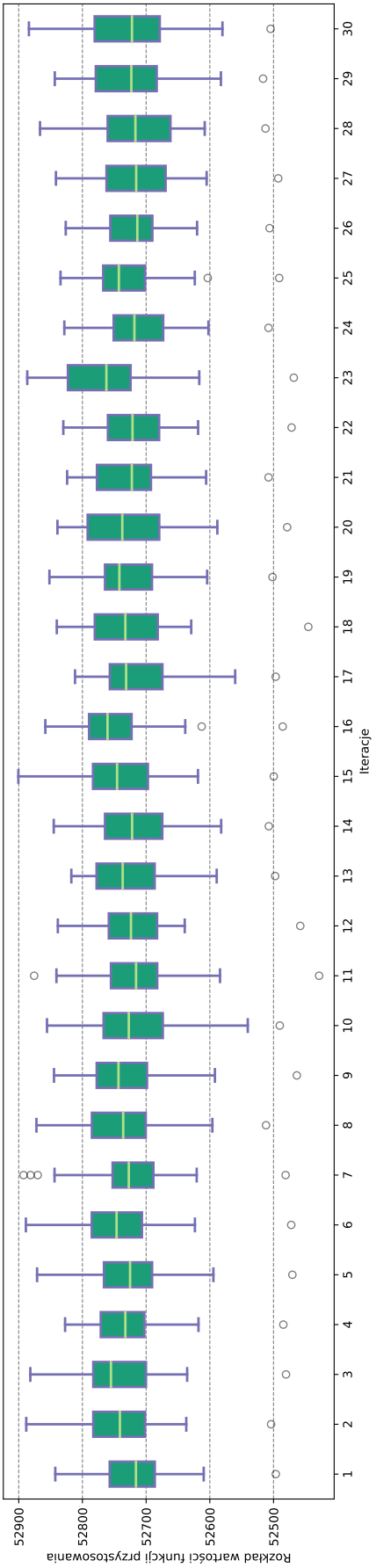
Rysunek 6.34: Przebieg wartości F_f testu #2 dla alg. *DPSO* - klucze #3, #4 i #5

Podobnie, jak miało to miejsce w przypadku poprzednich testów można doszukać się dużych wahań pomiędzy poszczególnymi wartościami funkcji przystosowania na wykresach. Podczas działania ataku *DPSO* nie jesteśmy w stanie zauważyć żadnej poprawny w kolejnych iteracjach algorytmu. Wartości minimum oraz maksimum F_f, które powinny ulegać stopniowemu zwiększeniu, mają charakter „skokowy", zarówno na początku jak i na końcu działania algorytmu.

Na rys. 6.35 i 6.36 zaprezentowano rozkład wartości funkcji przystosowania dla testów #3 i #4 - test #5 nie wprowadzał żadnych innowacji, więc zdecydowano się go pominąć.



Rysunek 6.35: Rozkład wartości F_f dla alg. *DPSO* - klucz #3



Rysunek 6.36: Rozkład wartości F_f dla alg. *DPSO* - klucz #4

7

Analiza skuteczności opracowanych algorytmów oraz wnioski z przeprowadzonych eksperymentów

W tym rozdziale przedstawiono szczegółowy opis analizy skuteczności pomiędzy wszystkimi zaimplementowanymi atakami, tj. *MASA*, *NEA* i *DPSO* oraz klasycznym atakiem siłowym. Część tą podzielono na podrozdziały, w których zajęto się analizą pod kątem liczby sprawdzonych rozwiązań oraz czasem odszyfrowania poprawnego kryptogramu. Podobnie jak to miało miejsce w poprzednim rozdziale, dotyczącym badań eksperymentalnych, ze względu na niemalże identyczne rozbieżności pomiędzy wszystkimi algorytmami, ograniczono się do opisu analizy skuteczności dla szyfru *DES*.

7.1 Porównanie algorytmów pod kątem liczby sprawdzonych rozwiązań

Jednym z najważniejszych kryteriów porównawczych była liczba sprawdzonych rozwiązań. W zależności od typu ataku rozważane będą zarówno podklucze ostatniej, szóstej rundy algorytmu szyfrującego - ataki *MASA* oraz *NEA*, jak i całe klucze szyfrujące, z ewentualnym pominięciem bitów parzystości - atak *DPSO*. Ponadto, zdecydowano się również na zamieszczenie dwóch wersji ataku siłowego - klasycznego, testującego każdy 56-bitowy klucz szyfrujący oraz ataku opierającego się na zastosowaniu kryptoanalizy różnicowej, podczas której testowane będą 48-bitowe podklucze. Szersze zestawienie,

odzwierciedlające liczbę sprawdzonych rozwiązań, dla każdego ataku opierającego się na wykorzystaniu metod kryptoanalizy różnicowej, oraz odpowiadającej mu charakterystyki Ω_P dla ostatniej rundy szyfrującej, zostało przedstawione w tab. 7.1:

Tabela 7.1: Zestawienie liczby sprawdzonych podkluczy, potrzebnych do odgadnięcia poprawnego klucza, alg. *MASA* i *NEA* dla szóstej rundy szyfru *DES*

Id		*MASA*		*NEA*		*Atak siłowy*
		Suma	Średnia	Suma	Średnia	*(mln.)*
1	Ω_P^1	227823	7594,1	135621	4520,7	~ 432
	Ω_P^2	264371	8812,4	172646	5754,8	~ 929,4
	Suma	492194	—	308267	—	~ 1361,4
2	Ω_P^1	170355	5678,5	92402	3080,1	~ 161,6
	Ω_P^2	280634	9354,5	157218	5240,6	~ 740,8
	Suma	450989	—	249620	—	~ 902,4
3	Ω_P^1	282939	9431,3	188596	6286,5	~ 283,7
	Ω_P^2	255812	8527,1	122854	4095,1	~ 37,9
	Suma	538751	—	311450	—	~ 321,6
4	Ω_P^1	280766	9358,87	156055	5201,8	~ 598
	Ω_P^2	376258	12541,9	177490	5916,3	~ 781
	Suma	657024	—	333545	—	~ 1379
5	Ω_P^1	318016	10600,5	174376	5812,5	~ 925,2
	Ω_P^2	364715	12157,2	168360	5612,0	~ 954,1
	Suma	682731	—	342736	—	~ 1879,3
Suma		~ 2,8 mln	—	~ 1,5 mln	—	~ 5843,7

Liczba sprawdzonych rozwiązań zliczana była do momentu odgadnięcia poprawnego podklucza w szóstej rundzie algorytmu szyfrującego. Dla każdego ataku wyznaczono sumę sprawdzonych podkluczy dla wszystkich trzydziestu uruchomień zarówno dla charakterystyki Ω_P^1, jak i Ω_P^2 - kolumna *Suma*. Następnie wyznaczono średnią liczbę sprawdzonych rozwiązań, spośród wszystkich uruchomień, dla obydwu charakterystyk. Kolejnym etapem było zsumowanie liczby podkluczy dla każdej charakterystyki, dzięki czemu obliczono całkowitą liczbę sprawdzonych rozwiązań dla danego podklucza. Ostatecznie, możliwe stało się wyznaczenie całkowitej liczby wszystkich sprawdzonych rozwiązań w szóstej rundzie. W celu polepszenia czytelności powyższej tabeli wartości w kolumnie, dotyczącej ataku siłowego, wyrażone zostały w milionach.

Podczas działania algorytmu do ataku przekazano dodatkową informację na temat ekstremum globalnego w postaci poprawnego podklucza. Oczywiście wiedza ta nie miała żadnego wpływu na jakość procesu kryptoanalizy któregokolwiek z ataków, a jedynie została wykorzystania do sprawdzenia czy w trakcie działania algorytmu w populacji znajduje się osobnik o identycznej postaci. Dzięki temu podejściu możliwe stało się wy-

znaczenie ile podkluczy musiało zostać sprawdzonych aż poprawne rozwiązanie zostało odgadnięte.

W ramach analizy zdecydowano się również o zamieszczenie zestawienia całkowitej liczby wszystkich sprawdzonych rozwiązań bez zatrzymywania algorytmu po odnalezieniu poprawnego podklucza. Zestawienie to zostało zamieszczone w tab. 7.2:

Tabela 7.2: Zestawienie liczby wszystkich sprawdzonych podkluczy alg. *MASA* i *NEA* dla szóstej rundy szyfru *DES*

Id		*MASA*		*NEA*		*Atak siłowy*
		Suma	Średnia	Suma	Średnia	
1	Ω_P^1	687536	22917,8	253387	8446,2	$30 \cdot 2^{30}$
	Ω_P^2	687893	22929,7	252870	8429,0	$30 \cdot 2^{30}$
	Suma	1375429	—	506257	—	$30 \cdot 2^{31}$
2	Ω_P^1	687765	22925,5	252319	8410,6	$30 \cdot 2^{30}$
	Ω_P^2	687961	22932,0	252093	8403,1	$30 \cdot 2^{30}$
	Suma	1375726	—	504412	—	$30 \cdot 2^{31}$
3	Ω_P^1	687931	22931,0	251679	8389,3	$30 \cdot 2^{30}$
	Ω_P^2	686311	22877,0	252690	8423	$30 \cdot 2^{30}$
	Suma	1374242	—	504369	—	$30 \cdot 2^{31}$
4	Ω_P^1	687817	22927,2	252819	8427,3	$30 \cdot 2^{30}$
	Ω_P^2	687927	22930,9	252798	8426,6	$30 \cdot 2^{30}$
	Suma	1375744	—	505617	—	$30 \cdot 2^{31}$
5	Ω_P^1	687917	22930,6	252322	8410,73	$30 \cdot 2^{30}$
	Ω_P^2	687866	22928,9	252565	8418,83	$30 \cdot 2^{30}$
	Suma	1375783	—	504887	—	$30 \cdot 2^{31}$
Suma		~ 6,9 mln	—	~ 2,5 mln	—	~ 321 mld

W powyższej tabeli przedstawiono liczbę wszystkich zweryfikowanych podkluczy aż do momentu osiągnięcia warunku stopu algorytmu - którym było osiągnięcie zadanej liczby iteracji. Po zakończeniu działania ataku możliwe stało się przeanalizowanie najlepiej dopasowanych osobników, dzięki czemu najlepszy wygenerowany podklucz został uznany za potencjalne rozwiązanie.

Poniżej, w tab. 7.3, zaprezentowano również zestawienie całkowitej i średniej liczby wszystkich sprawdzonych rozwiązań, wraz z zapotrzebowaniem na pamięć dla każdego z ataków. W obliczeniach uwzględniono wszystkie rundy szyfru, każdej z charakterystyk Ω_P, dla kompletu rozpatrywanych kluczy szyfrujących.

Tabela 7.3: Całkowita i średnia liczba wszystkich kluczy każdego z ataków

Algorytm	Całkowita liczba sprawdzonych rozwiązań	Średnia liczba sprawdzonych rozwiązań na klucz	Zużycie pamięci
MASA	41.298.312	275.322	148 MB
NEA	15.150.312	101.002	54 MB
Kryptoanaliza różnicowa	$150 \cdot (12 \cdot 2^{30} + 1024)$	$12 \cdot 2^{30} + 1024$	225 GB
DPSO	750.000	5.000	5 MB
Atak siłowy	$150 \cdot 2^{56}$	2^{56}	65,62 EB

Całkowita liczba rozwiązań, dla algorytmów kryptoanalizy różnicowej oraz ataku siłowego, została zapisana za pomocą prostego działania matematycznego. Stała 150 odzwierciedla iloczyn liczby przeprowadzonych testów (30) dedykowanych każdemu kluczowi szyfrującemu (5). Wartość 12 obliczona została na podstawie iloczynu liczby rund szyfru (6) dla każdej z charakterystyk Ω_P (2). Przy pomocy zapisu 2^x wyrażono liczbę wszystkich możliwych rozwiązań, w przypadku kryptoanalizy różnicowej, podkluczy (2^{30}) oraz, w sytuacji ataku siłowego, kluczy szyfrujących (2^{56}). Ostatnia liczba - 1024, obliczona została na podstawie iloczynu brakujących 6 bitów każdego podklucza, dla sześciu rund algorytmu szyfrującego oraz dwóch charakterystyk Ω_P, które odgadnięte zostaną przy pomocy metody przeglądu zupełnego ($2^6 \cdot 12$). Dzięki temu możliwe będzie odgadnięcie 48 z 56 bitów klucza głównego. Koniecznym będzie ponowne przeprowadzenie przeglądu zupełnego, tym razem w celu odgadnięcia pozostałych ośmiu bitów klucza szyfrującego ($2^6 \cdot 12 + 2^8$).

Zużycie pamięci zostało obliczone na podstawie liczby bitów w każdym osobniku pomnożonej przez liczbę zweryfikowanych rozwiązań. Następnie przekalkulowany iloczyn został podzielony przez 8 w celu konwersji bitów na bajty. Tak wygenerowany wynik dzielony był przez 1024 aż do momentu otrzymania prostej i czytelnej do zapisu wartości.

7.2 Porównanie algorytmów pod kątem czasu odszyfrowania kryptogramu

Następnym kryterium porównawczym był czas działania algorytmu. Każdy szyfr możne zostać złamany za pomocą najprostszego ataku siłowego. Należy jednak pamiętać, że atak ten, opiera się na eksploracji zbioru wszystkich możliwych rozwiązań, aż

do momentu odgadnięcia właściwego klucza deszyfrującego. Weryfikacja każdego rozwiązania polega na uruchomieniu pełnego algorytmu deszyfrującego, a następnie przeanalizowaniu otrzymanego tekstu, co w przypadku dzisiejszych szyfrów jest procesem czasochłonnym. W przypadku niektórych szyfrów, np. *AES-256* może zająć miliardy lat.

Podobnie, jak to miało miejsce w przypadku poprzedniego kryterium, rozważane będą zarówno podklucze ostatniej rundy szyfru *DES* - mowa tutaj o atakach *MASA* i *NEA*, oraz pełne, 56-bitowe klucze szyfrujące, wykorzystywane w algorytmie *DPSO*. Szacunkowe czasy klasycznego ataku siłowego oraz kryptoanalizy różnicowej również zostały uwzględnione. Zestawienie wszystkich algorytmów zostało przedstawione w tab. 7.4.

Tabela 7.4: Zestawienie czasu działania, potrzebnego do odgadnięcia poprawnego podklucza, alg. *MASA* i *NEA* dla szóstej rundy szyfru *DES*

Id		*MASA*		*NEA*		*Atak siłowy*
		Suma	Średnia	Suma	Średnia	
1	Ω_P^1	231,08 s	7,70 s	137,40 s	4,58 s	46,47 min
	Ω_P^2	296,66 s	9,82 s	201,39 s	6,71 s	21,60 min
	Suma	527,74 s	—	338,79 s	—	68,07 min
2	Ω_P^1	130,77 s	4,36 s	67,38 s	2,25 s	8,08 min
	Ω_P^2	303,12 s	10,10 s	175,84 s	5,86 s	37,04 min
	Suma	433,89 s	—	243,22 s	—	45,12 min
3	Ω_P^1	276,20 s	9,21 s	177,23 s	5,91 s	14,18 min
	Ω_P^2	284,56 s	9,49 s	152,38 s	5,08 s	1,90 min
	Suma	560,76 s	—	329,61 s	—	16,08 min
4	Ω_P^1	274,89 s	9,16 s	153,70 s	5,12 s	29,90 min
	Ω_P^2	391,33 s	14,04 s	214,62 s	7,15 s	39,05 min
	Suma	666,22 s	—	368,32 s	—	68,95 min
5	Ω_P^1	299,49 s	9,98 s	170,03 s	5,67 s	46,26 min
	Ω_P^2	379,25 s	12,64 s	200,87 s	6,70 s	47,70 min
	Suma	678,74 s	—	370,90 s	—	93,96 min
Suma		47,96 min	—	27,51 min	—	292,18 min

Czas mierzony był do momentu odgadnięcia poprawnego podklucza. Powyższa tabela przedstawia sumę czasów dla wszystkich trzydziestu uruchomień każdej z charakterystyk. Ponadto wyznaczono średni czas kryptoanalizy ostatniego podklucza dla każdego z ataków. Dzięki temu możliwe było obliczenie całkowitego czasu działania wszystkich algorytmów w ostatniej rundzie szyfru. W celu utrzymania przejrzystości tabeli czas działania ataku siłowego został wyrażony w minutach (*min*), natomiast czasy algorytmów *MASA* i *NEA* w sekundach (*s*).

Wyniki ataków, zaprezentowane w tab. 7.4, opierały się o wykorzystanie dodatkowej informacji na temat poprawnego podklucza. Wiedza ta w żaden sposób nie miała wpływu na proces kryptoanalizy czy działanie samego ataku. W tab. 7.5 zamieszczono zestawienie całkowitego czasu działania bez zatrzymywania algorytmu po odgadnięciu podklucza:

Tabela 7.5: Zestawienie całkowitego czasu działania alg. *MASA* oraz *NEA* dla szóstej rundy szyfru *DES*

Id		*MASA*		*NEA*		*Atak siłowy*
		Suma	Średnia	Suma	Średnia	
1	Ω_P^1	614,94 s	20,50 s	257,32 s	8,58 s	53,69 min
	Ω_P^2	757,15 s	25,24 s	281,33 s	9,38 s	53,69 min
	Suma	1372,09 s	—	538,65 s	—	107,38 min
2	Ω_P^1	516,13 s	17,20 s	186,81 s	6,23 s	53,69 min
	Ω_P^2	723,75 s	24,13 s	289,53 s	9,65 s	53,69 min
	Suma	1239,88 s	—	476,34 s	—	107,38 min
3	Ω_P^1	666,05 s	22,20 s	237,31 s	7,91 s	53,69 min
	Ω_P^2	745,78 s	24,86 s	297,14 s	9,90 s	53,69 min
	Suma	1411,83 s	—	534,45 s	—	107,38 min
4	Ω_P^1	670,33 s	22,34 s	247,47 s	8,25 s	53,69 min
	Ω_P^2	706,06 s	23,54 s	301,35 s	10,05 s	53,69 min
	Suma	1376,39 s	—	548,82 s	—	107,38 min
5	Ω_P^1	643,86 s	21,46 s	243,53 s	8,12 s	53,69 min
	Ω_P^2	750,99 s	25,03 s	299,39 s	9,98 s	53,69 min
	Suma	1394,85 s	—	542,92 s	—	107,38 min
Suma		113,25 min	—	44,02 min	—	536,90 min

W tabeli powyżej przedstawiono całkowity czas działania do momentu osiągnięcia przez każdy algorytm warunku stopu, w tym przypadku zadanej liczby iteracji. Najbardziej przystosowany osobnik został uznany za potencjalne rozwiązanie.

W przestawionej poniżej tab. 7.6 zaprezentowano zestawienie całkowitego i średniego czasu działania wszystkich ataków. W tabeli zostały uwzględnione wszystkie rundy szyfru dla każdej z charakterystyk Ω_P i rozpatrywanego klucza.

Tabela 7.6: Szacowany całkowity i średni czas kryptoanalizy każdego z ataków

Algorytm	Szacowany całkowity czas wszystkich sprawdzonych rozwiązań	Szacowany średni czas jednego kompletnego rozwiązania na klucz
MASA	679,5 min	4,53 min
NEA	264,12 min	1,76 min
Kryptoanaliza różnicowa	53 h	21,47 min
DPSO	352,5 h	2,35 h
Atak siłowy	~ 34 273 lat	~ 228,5 lat

W powyższej tabeli czasy dla algorytmów *MASA* i *NEA* zostały wyrażone w minutach (*min*), dla kryptoanalizy różnicowej oraz ataku *DPSO* w większości przypadków w godzinach (*h*), natomiast dla klasycznego ataku siłowego w latach. Oczywiście, czas ataku siłowego, został oszacowany na podstawie kilku początkowych eksperymentów, a następnie przeskalowany do całkowitej liczby wszystkich możliwych kluczy oraz uruchomień.

7.3 Statystyczna weryfikacja poprawności opracowanych algorytmów

Ataki *MASA* oraz *NEA* charakteryzują się pewnym elementem pseudolosowości. W celu przeprowadzenia statystycznej weryfikacji zaprezentowanych algorytmów posłużono się nieparametrycznym testem Wilcoxona, stosowanym do porównania wyników dwóch prób niezależnych. Na potrzeby testu postawione zostały następujące hipotezy:

- H_0 - brak różnicy przy porównywaniu obydwóch próbek,
- H_1 - istnieje różnica pomiędzy porównywanymi próbkami.

Do przeprowadzenia testu posłużono się następującymi kryteriami: wartością funkcji przystosowania - najlepiej przystosowanych podkluczy, oraz liczbą sprawdzonych rozwiązań. Waga każdego z kryteriów była taka sama i wynosiła 0,5.

Dla wszystkich przeprowadzonych analiz hipoteza H_0 została odrzucona dla wartości $p < 0,05$. Oznacza to, że wyniki uzyskane przez algorytm *MASA* są znacznie lepsze od wyników ataku *NEA*.

W celu sprawdzenia powtarzalności dwóch zaprezentowanych ataków podobnej analizie zostało poddanych 150 uruchomień każdego z algorytmów. We wszystkich przypadkach wynik testu wskazywał na brak różnic pomiędzy próbkami (p miał dużą wartość),

dzięki czemu można stwierdzić, że próbki charakteryzują się podobnymi rezultatami, czyli wyniki są powtarzalne.

7.4 Wnioski

Na podstawie opisanych w pracy badań, na przykładzie szyfru *DES*, można stwierdzić, że zaproponowany algorytm memetyczny *MASA* znacznie usprawnia proces kryptoanalizy symetrycznych szyfrów blokowych. Wyniki poszczególnych eksperymentów przestawione zostały w rozdziale 7 - charakterystyka Ω_P^1, oraz w dodatku A - charakterystyka Ω_P^2.

Algorytm *MASA* zwiększył częstotliwość poprawnie odgadniętych bitów ostatniego podklucza K_6, dzięki czemu proces kryptoanalizy w ponad 90% przypadków został zakończony sukcesem, w porównaniu do wcześniej opracowanego ataku *NEA*, oraz zaproponowanego w literaturze algorytmu *DPSO*. Algorytm *MASA* został również porównany z dwoma klasycznymi atakami kryptoanalitycznymi, takimi jak atak siłowy czy kryptoanaliza różnicowa.

Skuteczność zaproponowanego podejścia jest większa, a tym samym algorytm sprawdza znacznie mniejszą liczbę podkluczy. W porównaniu do klasycznych metod kryptoanalitycznych - tab. 7.3, łatwo można zauważyć mniejsze zapotrzebowanie na pamięć algorytmu *MASA*. W tabeli widoczne jest porównanie ataków, już na samym poziomie jednostek pamięci tj. pomiędzy zużytymi megabajtami (*MB*) algorytmu *MASA*, a gigabajtami metody kryptoanalizy różnicowej (*GB*), a nawet eksabajtami (*EB*, 10^{18}) klasycznego ataku siłowego. Opracowany wcześniej algorytm *NEA* zużywa nieco mniej pamięci, aczkolwiek z punktu widzenia skuteczności, każdego z ataków, różnica ta - około 100 MB, może zostać pominięta. Najmniejszym zapotrzebowaniem na pamięć charakteryzuje się atak *DPSO*. Niestety, ze względu na skuteczność kryptoanalizy nie jest ono zbyt istotne.

Z punktu widzenia czasu działania każdego z algorytmów - tab. 7.6, również można zauważyć znaczną przewagę algorytmu *MASA* nad metodami klasycznymi i atakiem *DPSO*. 150 uruchomień algorytmu memetycznego zajmuje około 12 godzin, gdzie podstawowa metoda kryptoanalizy różnicowej będzie potrzebowała ponad 50 godzin, natomiast zaproponowany w literaturze atak *DPSO* dwóch tygodni (352,5 godzin), a klasyczny atak siłowy aż 35 tysięcy lat. Jedynie zaproponowany wcześniej algorytm *NEA* realizuje proces kryptoanalizy nieco szybciej - o średnio 2 - 3 minuty, aczkolwiek jest to tak krótki czas, który warto poświecić, aby uzyskać pewniejsze rozwiązanie przy

pomocy algorytmu *MASA*.

Zwiększoną skuteczność bez wątpienia zawdzięczamy znacznie bardziej zaawansowanemu procesowi eksploatacji, zrealizowanego za pomocą algorytmu symulowanego wyżarzania. Pociąga on za sobą sprawdzenie średnio trzy razy większej liczby podkluczy, niż algorytm *NEA*, aczkolwiek różnice, pod kątem czasowym i pamięciowym, są na tyle małe, że mogą zostać pominięte na poczet skuteczniejszego procesu metaheurystycznej kryptoanalizy różnicowej.

8

Podsumowanie

Niniejsza rozprawa poświęcona została zagadnieniom związanym z zastosowaniem algorytmów metaheurystycznych w kryptoanalizie symetrycznych szyfrów blokowych. Głównym celem rozprawy było opracowanie autorskich ataków kryptoanalitycznych w oparciu o wybrane algorytmy metaheurystyczne skierowane przeciwko kryptogramom wygenerowanym za pomocą szyfrów *FEAL* i *DES*. W przypadku każdego z opracowanych ataków omówione zostały wszystkie istotne zagadnienia teoretyczne. W ramach pracy zrealizowano również część praktyczną opierającą się na implementacji zaproponowanych ataków, wraz z algorytmami opisanymi w literaturze. Następnie przeprowadzono szczegółowe badania i eksperymenty dla każdego z nich oraz wykonano analizę otrzymanych wyników.

Teza pracy „Zastosowanie algorytmów memetycznych, w procesach kryptoanalizy różnicowej, dla wybranych symetrycznych szyfrów blokowych, poprawia efektywność samego ataku, a tym samym jego skuteczność." została potwierdzona. W trakcie realizacji celów zostały opracowane i przebadane różne wersje algorytmu memetycznego pełniącego rolę metaheurystycznego ataku kryptoanalitycznego. Algorytm *MASA* został opisany w podrozdziałach 5.1 oraz 5.2, pod kątem szyfrów *FEAL* i *DES*, natomiast opis ataku ewolucyjnego *NEA* zawarty został w podrozdziale 5.3. Szczegółowo opisano również atak *DPSO* - podrozdział 5.3.2, skierowany na szyfr *DES* zaproponowany w literaturze. W wyniku przeprowadzonych badań zrealizowane zostały następujące cele:

1. Przeanalizowano istniejące metody ataków kryptoanalitycznych, na przykładzie szyfrów *FEAL* oraz *DES*. Szczególny nacisk postawiono tutaj na rozpoznanie najpopularniejszych technik, takich jak kryptoanaliza różnicowa i kryptoanaliza liniowa, a następnie dokonano przeglądu najpopularniejszych algorytmów meta-

heurystycznych, wykorzystywanych w ewolucyjnej kryptoanalizie wspomnianych szyfrów. Uwzględniono również najpopularniejsze modyfikacje oraz szereg wariantów hybrydowych. Koniecznym stało się rozpoznanie dużej liczby terminów i problemów obecnych w tematyce kryptografii. Dzięki czemu możliwym stało się wskazanie najsłabszych elementów symetrycznych szyfrów blokowych oraz najlepszych technik ataku, co stanowiło preludium do opracowania własnego ataku metaheurystycznego.

2. Kolejnym etapem było przemodelowanie zagadnień kryptoanalitycznych w klasyczny problem optymalizacji funkcji, uwzględniającej opracowanie własnej funkcji oceny każdego z rozwiązań. Pozwoliło to na wykorzystanie wybranego algorytmu metaheurystycznego, jako narzędzia ataku, pozwalającego zoptymalizować złożony i czasochłonny proces kryptoanalizy różnicowej.

3. Następnie przeanalizowano i przetestowano kilka najpopularniejszych metaheurystyk, takich jak algorytmy ewolucyjne, ze szczególnym uwzględnieniem algorytmów genetycznych, optymalizację stadną cząsteczek, algorytmy mrowiskowe czy w końcu algorytmy memetyczne. Proces analizy każdej z metaheurystyk rozpoczynał się od realizacji ataków na najprostsze szyfry kryptografii klasycznej - szyfr przestawieniony i podstawieniowy, poprzez uproszczone wersje symetrycznych szyfrów blokowych - *SDES*, kończąc na oryginalnych wersjach algorytmów *FEAL* oraz *DES* z zredukowaną liczbą rund. Podejście to zaowocowało stopniowym przeanalizowaniem każdego z algorytmów szyfrujących, zaczynając od najprostszych i najmniej zaawansowanych, kończąc na praktycznych implementacjach stosowanych w systemach informatycznych.

4. Kolejnym elementem było zaprojektowanie i zaimplementowanie specjalnego oprogramowania - platformy informatycznej, umożliwiającej automatyczną krypoanalizę różnicową. Dzięki zrealizowanemu oprogramowaniu możliwe stało się stworzenie harmonogramu uruchomień wszelkiego rodzaju ataków, skierowanych przeciwko zdefiniowanym w szablonie szyfrom wraz z możliwością skonfigurowania parametrów każdego algorytmu. Pozwoliło to na uruchomienie ciągłego, trwającego tygodniami, procesu ewolucyjnej kryptoanalizy, realizowanej według ustalonego harmonogramu.

5. Przegląd literaturowy najpopularniejszych ataków, wraz z udziałem platformy do automatycznej kryptoanalizy, pozwolił na ustanowienie najlepszych wartości

parametrów opracowanych ataków dla każdego z szyfrów. Najistotniejszymi parametrami są rozmiar populacji N, liczba wygenerowanych par tekstów jawnych γ oraz w przypadku algorytmu *MASA* współczynnik wygaszania α, natomiast dla algorytmu *NEA* prawdopodobieństwo heurystycznej negacji P_n. Parametry zostały dobrane w sposób empiryczny.

6. Badania przedstawione zostały w rozdziale 6, natomiast analiza skuteczności opracowanych algorytmów w rozdziale 7. Zdecydowano się przeprowadzić eksperymenty na pięciu różnych kluczach dla każdego z szyfrów. Analiza dotyczyła wielu aspektów z uwzględnieniem jakości otrzymanych rozwiązań oraz liczby par tekstów jawnych. W podrozdziale dotyczącej analizy skuteczności wskazano wyższość opracowanych ataków metaheurystycznych nad atakiem *DPSO* zaproponowanym w literaturze oraz nad klasycznymi metodami kryptoanalitycznymi - kryptoanalizą różnicową i atakiem siłowym. Wykazano tutaj słabe strony klasycznych metod pod kątem liczby sprawdzonych rozwiązań. Następnym zagadnieniem był czas działania algorytmów. Ponownie wskazano wyższość opracowanych ataków nad klasycznymi metodami oraz algorytmem *DPSO*.

Wszystkie cele postawione w rozprawie zostały zrealizowane. Otrzymane wyniki potwierdzają, że zastosowanie technik metaheurystycznych, w tym przypadku algorytmów memetycznych, są w pełni zasadne. Znacznie skrócono czas, potrzebny na odgadnięcie poprawnego klucza deszyfrującego - tab. 7.5, oraz ograniczono zbiór potencjalnych rozwiązań, co sprowadza się do mniejszego zapotrzebowania algorytmów na pamięć - tab. 7.2.

Wszystkie zaprezentowane w rozprawie wyniki zostały zweryfikowane za pomocą testu statystycznego. Są one w pełni reprodukowane za pomocą, przedstawionej w dodatku B, listy par tekstów jawnych i szyfrogramów dla charakterystyki Ω_1.

Analiza wyników pozwala wysunąć szereg wniosków, co bez wątpienia stanowi dobry punkt wyjścia do dalszych badań. W przyszłości planuje się prace nad kolejnymi, bardziej zaawansowanymi atakami oraz modyfikacjami, skierowanymi na szyfry takie jak *AES* czy *GOST*. Duża skuteczność opracowanych ataków może oznaczać, że inne algorytmy metaheurystyczne są w stanie uzyskać równie satysfakcjonujące wyniki, jak zaprezentowane w rozprawie algorytmy memetyczne. Implementacja równoległa opracowanych algorytmów również powinna mieć znaczny wpływ na wydajność, a tym samym skrócenie czasu kryptoanalizy przedstawionych ataków, co bez wątpienia stanowi kierunek przyszłych prac.

Bibliografia

[1] S. Abraham, I. Kiss, S. Sanyal, M. Sanglikar. *Steepest Ascent Hill Climbing For A Mathematical Problem*. Advanced Engineering and Applied Management, Hunedoara, 2010.

[2] J. Arabas. *Wykłady z algorytmów ewolucyjnych. Wydanie II*. Wydawnictwo Naukowo-Techniczne, Warszawa, 2004.

[3] R. Anderson. *On Fibonacci KeyStream Generators*. Fast Software Encryption, Springer-Verlag, wolumen 1008, strony 346-352, 1995.

[4] W. Alsaeedan, M. E. B. Menai, S. Al-Ahmadi. *A hybrid genetic-ant colony optimization algorithm for the word sense disambiguation problem*. Information Sciences 417, strony 20-38, 2017.

[5] J. P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, C. Winnerlein. *BLAKE2: simpler, smaller, fast as MD5*. International Conference on Applied Cryptography and Network Security, Springer-Verlag, strony 119-135, 2013.

[6] J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon, A Abraham. *Interia Weight Strategies in Particle Swarm Optimization*. Third World Congress on Nature and Biologically Inspired Computing, IEEE, strony 640-647, 2011.

[7] K. Bartyzel. *Kryptografia asymetryczna i jej zastosowanie w algorytmach komunikacji*. V Konferencja Informatyki Stosowanej, 2006. Dostępne w Internecie: http://www.kis.pwszchelm.pl/publikacje/V/Bartyzel.pdf [dostęp: 2022-02-02 15:00].

[8] F. L. Bauer. *Decrypted Secrets: Methods and Maxims of Cryptology*. Springer, Monachium, 2002.

[9] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. *Keccak sponge function family main document*. Zgłoszenie do konkursu NIST na algorytm SHA-3, 2009.

[10] W. Bieluch. Wpływ operatorów mutacji na skuteczność podzukiwań AE. Materiały do przedmiotu Obliczenia Ewolucyjne, Politechnika Śląska. Dostępne w Internecie: http://www.imio.polsl.pl/Dopobrania/Lab%20OE%2004%20(mutacja).pdf [dostęp: 2022-02-01 15:00].

[11] E. Biham, A. Shamir. *Differential cryptanalysis of DES-like cryptosystems*. Journal of Cryptology, wolumen 4(1), strony 3-72, 1991.

[12] E. Biham, A. Shamir. *Differential cryptanalysis of DES*. Springer-Verlag, Nowy Jork, 1993.

[13] I. Blake, G. Seroussi, N. Smart. *Krzywe eliptyczne w kryptografii*. Wydawnictwo Naukowo-Techniczne, Warszawa, 2004.

[14] A. Chrzęszczyk. *Algorytmy teorii liczb i kryptografii w przykładach*. Wydawnictwo BTC, Legionowo, 2010.

[15] O. Cordón, S. Damas, J. Santamaría. *A CHC evolutionary algorithm for 3D image registration*. International Fuzzy Systems Association World Congress, Springer-Verlag, strony 404-411, 2003.

[16] J. Daemen, V. Rijmen. *Rijndael: The Advanced Encryption Standard*. Dr. Dobb's Journal, wolumen 26, numer 3, strony 137-139, 2001.

[17] J. Daemen, V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag, Nowy Jork, 2002.

[18] L. Davis. *Job-shop Scheduling with Genetic Algorithms*. International Conference on Genetic Algorithms and Their Applications, strony 136-140, 1985.

[19] W. Diffie, M. E. Hellman. *New Directions in Cryptography*. IEEE Transactions on Information Theory. Wolumen IT-22, numer 6, strony 644-654, 1976. Computational Collective Intelligence. Technologies and Applications. Lecture Notes in Computer Science, wolumen 9876, strony 102-112, 2016.

[20] M. Dorigo. *Optimization, learning and natural algorithms*. Phd Thesis, 1992.

[21] K. Dworak. *Zastosowanie algorytmów ewolucyjnych w kryptoanalizie klasycznej*. Systemy Inteligencji Obliczeniowej, Instytut Informatyki Uniwersytetu Śląskiego, strony 45-55, 2014.

[22] K. Dworak, U. Boryczka. *Differential cryptanalysis of FEAL4 using evolutionary algorithm*. Computational Collective Intelligence. Technologies and Applications. Lecture Notes in Computer Science, wolumen 9876, strony 102-112, 2016.

[23] K. Dworak, U. Boryczka. *Differential cryptanalysis of symmetric block ciphers using memetic algorithms*. Intelligent Information and Database Systems. Lecture Notes in Computer Science, wolumen 11432, strony 275-286, 2019.

[24] K. Dworak, U. Boryczka. *Genetic algorithm as optimization tool for differential cryptanalysis of DES6*. Computational Collective Intelligence. Technologies and Applications. Lecture Notes in Computer Science, wolumen 10449, strony 107-116, 2017.

[25] K. Dworak, U. Boryczka. *Breaking Data Encryption Standard with a Reduced Number of Rounds Using Metaheuristics Differential Cryptanalysis*. Entropy, wydanie spacjalne: „Entropy in Real-World Datasets and Its Impact on Machine Learning", wolumen 23, numer 12, strony 1-21, 2021.

[26] T. ElGamal. *A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. Advances in Cryptology: Proceedings of CRYPTO 84. Springer-Verlag, strony 10-18, 1985.

[27] A. P. Engelbrecht. *Computational Intelligence An Introduction*. John Wiley and Sons, 2007.

[28] H. Feistel. *Cryptography and Computer Privacy*. Scientific American, wolumen 228, numer 4, strony 15-23, 1973.

[29] C. Fernandes, R. Tavares, C. Munteanu, A. Rosa. *Using assortative mating in genetic algorithms for vector quantization problems*. Symposium on Applied Computing, strony 361-365, 2001.

[30] R. M. French, A. Messinger. *Genes, Phenes and the Baldwin Effect: Learning and Evolution in a Simulated Population*. Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems ArtificialLifeIV, strony 277-282, MIT Press, 1994.

[31] L. J. Fogel, A. J. Owens, M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley and Sons, 1966.

[32] S. Kirkpatrick, Jr. C. D. Gelatt, M. P. Vecchi. *Optimization by simulated annealing*. Science 220(4598), strony 671-680, 1983.

[33] W. A. Ghonaim. *Evolutionary Computation in Cryptanalysis*. Rozprawa doktorska. Department of Mathematics Faculty of Science (Girls) Al-Azhar University, 2013.

[34] W. A. Ghonaim, N. I. Ghali, A. E. Hassanien. *Known-Ciphertext Cryptanalysis Approach for the Data Encryption Standard Technique*. 6th IEEE International Conference on Next Generation Web Services Practices, strony 23-25, 2010.

[35] F. Glover. *Future Paths for Integer Programming and Links to Artificial Intelligence*. Computers & Operations Research, Wolumen 13, strony 533-549, 1986.

[36] F. Glover, M. Laguna. *Tabu search*. Handbook of combinatorial optimization, strony 2093-2229, 1998.

[37] F. Glover, C. McMillan. *The general employee scheduling problem. An integration of MS and AI*. Computers & operations research, 13(5), strony 563-573, 1986.

[38] D. E. Goldberg. *Algorytmy genetyczne i ich zastosowania*. Wydawnictwo Naukowo-Techniczne, Warszawa, 2003.

[39] D. E. Goldberg, R. Lingle. *Alleles, loci, and the traveling salesman problem*. International Conference on Genetic Algorithms and Their Applications, strony 154-159, 1985.

[40] T. D. Gwiazda. *Algorytmy genetyczne kompedium. Operator krzyżowania dla problemów numerycznych*. Wydawnictwo Naukowe PWN SA, Warszawa, 2007.

[41] T. D. Gwiazda. *Algorytmy genetyczne kompedium. Operator mutacji dla problemów numerycznych*. Wydawnictwo Naukowe PWN SA, Warszawa, 2007.

[42] B. Harris. *Improved Arcfour Modes for the Secure Shell (SSH) Transport Layer Protocol*. Network Working Group, Request for Comments: 4345, Category: Standards Track, 2006. Dostępne w Internecie: https://tools.ietf.org/html/rfc4345 [dostęp: 2022-02-02 15:00].

[43] W. E. Hart, N. Krasnogor, J. E. Smith. *Memetic Evolutionary Algorithms*. Recent Advances in Memetic Algorithms, Studies in Fuzziness and Soft Computing, Springer, Wolumen 166, strony 3 - 31, 2005.

[44] L. S. Hill. *Cryptography in an Algebraic Alphabet*. The American Mathematical Monthly. Wolumen 36, strony 306-312, 1929.

[45] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[46] G. Jones *Genetic and Evolutionary Algorithms*. Encyclopedia of Computational Chemistry, 1998.

[47] A. Joux. *Algorithmic Cryptanalysis*. Chapman & Hall /CRC, Boca Raton, 2009.

[48] D. Kahn. *Łamacze kodów: historia kryptologii*. Wydawnictwo Naukowo-Techniczne, Warszawa, 2004.

[49] D. Karaboga. *An idea based on honey bee swarm for numerical numerical optimization*. Technical Report-TR06, 2005.

[50] M. Karbowski. *Podstawy kryptografii. Wydanie III*. Helion, Gliwice, 2015.

[51] J. Katz, Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC Press, Boca Raton, 2007.

[52] K. Kenan. *Kryptografia w bazach danych. Ostatnia linia obrony*. Wydawnictwo Naukowe PWN SA, Warszawa, 2007.

[53] A. Kerckhoffs. *La cryptographie militaire*. Journal des sciences militaires, volumen IX, strony 5–38, 1883.

[54] L. R. Knudsen. *Block ciphers-analysis, design and applications*. Department of Computer Science, 1994.

[55] M. Komosiński, M. Hapke. *Optymalizacja. Przeszukiwanie lokalne*. Instytut Informatyki, Politechnika Poznańska. Dostępne w Internecie: https://www.cs.put.poznan.pl/mkomosinski/materialy/optymalizacja/LS.pdf [dostęp: 2019-07-15 15:00].

[56] K. Kowalczyk. *Szyfr Beaufort'a*. [online]. Dostępne w Internecie: http://www.crypto-it.net/pl/proste/szyfr-beaufort.html?tab=1 [dostęp: 2022-02-02 15:00].

[57] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press Cambridge, Massachusetts, 1992.

[58] X. Lai, J. Massey. *A Proposal for a New Block Encryption Standard*. Advances in Cryptology, EuroCrypt, Springer-Verlag, strony 389-404, 1991.

[59] S. K. Langford, M. E. Hellman. *Differential-Linear Cryptanalysis*. Advances in Cryptology: Proceedings of CRYPTO'84, Lecture Notes in Computer Science 839, Springer-Verlag, strony 17-25, 1994.

[60] P. Larranaga, c. Kuijpers, R. H. Murga, I. Inza, S.Dizdarevic. *Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators*. Artificial Intelligence Review, wolumen 13(2), strony 129-170, 1999.

[61] M. Matsu. *Linear cryptanalysis method for DES cipher*. Advances in Cryptology, Proceedings of EuroCrypt'93, Lecture Notes in Computer Science 765, Springer-Verlag, strony 386-397, 1994.

[62] M. Matsu. *The first experimental cryptanalysis of the data encryption standard*. Advances in Cryptology: Proceedings of CRYPTO'84, Lecture Notes in Computer Science 839, Springer-Verlag, strony 1-11, 1994.

[63] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone. *Kryptografia stosowana*. Wydawnictwo Naukowo-Techniczne, Warszawa, 2005.

[64] R. C. Merkle, M. Hellman. *Hiding Information and Signatures in Trapdoor Knapsacks*. IEEE Transactions and Information Theory, wolumen 24, numer 5, strony 525-532, 1978.

[65] P. Merz. *Memetic Algorithms for. Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies*. Department of Electrical Engineering and Computer Science at the University of Siegen, 2006.

[66] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. Teller, E. Teller. *Wuation of state calculation by fast computing machines*. Journal of Chemical Physics, wolumen 21, strony 1087-1091, 1953.

[67] Z. Michalewicz. *Algorytmy genetyczne + struktury danych = programy ewolucyjne*. Wydawnictwo Naukowo-Techniczne, Warszawa, 2003.

[68] Z. Michalewicz, C. Z. Janikow. *GENOCOP: A Genetic Algorithm for Numerical Optimization Problems with Linear Constraints*. Communications of the ACM, wolumen 39(12), strony 175-201, 1996.

[69] Z. Michalewicz, T. Logan, S. Swaminathan. *Evolutionary operators for continuous convex parameter spaces*. Annual conference on Evolutionary Programming, strony 84-97, 1994.

[70] M. Moradi, M. Abedinie. *A combination of Genetic Algorithm and Particle Swarm Optimization for optimal DG location and sizing in distribution systems*. International Journal of Electrical Power & Energy Systems, wolumen 34, strony 66-74, 2010.

[71] P. Moscato. *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms*. Technical Report C3P 826, 1989.

[72] National Bureau of Standards. *Data Encryption Standard*. National Bureau of Standards, U.S. Department of Commerce, 1977.

[73] F. Neri, C. Cotta, P. Moscato. *Handbook of Memetic Algorithms*. Springer-Verlag, wolumen 379, 2012.

[74] D. J. Wheeler, R. M. Needham. *Tea extensions*. Raport techniczny, Cambridge University, 1997.

[75] T. Nomura. *An Analysis on Crossovers for Real Numbe r Chromosomes in an Infinite Population Size*. ATR Human Information Processing Research Laboratories, strony 936-941, 1997.

[76] J. Pieprzyk, T. Hardjono, J. Seberry. *Teoria bezpieczeństwa systemów komputerowych*. Helion, Gliwice, 2005.

[77] S. C. Pohlig, M. E. Hellman. *An Improved Algorithm for Computing Logarithms in GF(p) and Its Cryptographic Significance*. IEEE Transactions and Information Theory, wolumen 24, numer 1, strony 106-111, 1978.

[78] M. O. Rabin. *Digital Signatures and Public-Key Functions as Intractable as Factorization*. MIT Laboratory of Computer Science, Technical Report, 1979. Communications of the ACM, wolumen 21, numer 2, strony 120-126, 1978.

[79] R. L. Rivest, B. Agre, B. V. Bailey, Ch. Crutchfield i inni. *The MD6 hash function - a proposal to NIST for SHA-3*. Zgłoszenie do konkursu NIST na algorytm SHA-3, 2008.

[80] R. L. Rivest, A. Shamir, L. M. Adleman. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Communications of the ACM, wolumen 21, numer 2, strony 120-126, 1978.

[81] R. L. Rivest. *The RC5 encryption algorithm*. International Workshop on Fast Software Encryption, strony 86-96, 1994.

[82] D. Rodriguez-Clark. *Rail Fence Cipher*. [online]. Dostępne w Internecie: https://crypto.interactive-maths.com/rail-fence-cipher.html [dostęp: 2019-01-04 15:00].

[83] P. Rogaway, D. Coppersmith. *A Software-Oriented Encryption Algorithm*. Fast Software Encryption, Springer-Verlag, strony 56-63, 1994. Communications of the ACM, wolumen 21, numer 2, strony 120-126, 1978.

[84] Security Algorithms Group of Experts (SAGE). *Report on the specification and evaluation of the GSM cipher algorithm A5/2*. Raport techniczny, 1996.

[85] B. Schneier. *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)*. Fast Software Encryption, Springer-Verlag, strony 191-204, 1994.

[86] B. Schneier. *Kryptografia dla praktyków. Protokoły, algorytmy i programy źródłowe w języku C*. Wydawnictwo Naukowo-Techniczne, Warszawa, 2002.

[87] B. Schneier. *The Blowfish Encryption Algorithm*. Dr. Dobb's Journal, wolumen 19, numer 4, strony 38-40, 1994.

[88] B. Schneier, J. Kelsey, D. Whiting, D.Wagner, Ch. Hall. *Twofish: A 128-Bit Block Cipher*. Dostępne w Internecie: https://www.schneier.com/twofish-analysis-shiho.pdf [dostęp: 2019-01-04 15:00].

[89] *Evolution Strategy and Numerical Optimization*. Dissertation, Technical University of Berlin, 1974.

[90] C. E. Shannon. *Communication theory of secrecy systems*. Bell System Technical Journal, wolumen 28, strony 656-715, 1949.

[91] A. Shimizu, S. Miyaguchi. *Fast Data Encipherment Algorithm FEAL*. Advances in Cryptology, EuroCrypt, Springer-Verlag, strony 267-278, 1988.

[92] D. Sklyarov *Łamanie zabezpieczeń programów*. Wydawnictwo Read Me, Warszawa, 2009.

[93] W. Stallings. *Kryptografia i bezpieczeństwo sieci komputerowych. Matematyka szyfrów i techniki kryptologii*. Helion, Gliwice, 2011.

[94] M. Stamp, R. M. Low. *Applied Cryptanalysis. Breaking Ciphers in the Real World*. John Wiley and Sons, 2007.

[95] D. R. Stinson. *Kryptografia*. Wydawnictwo Naukowo-Techniczne, Warszawa, 2005.

[96] R. Storn, K. Price. *Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces*. Journal of Global Optimization 11 (4), strony 341-359, 1997.

[97] D. J. Wheeler, R. M. Needham. *TEA, a Tiny Encryption Algorithm*. Fast Software Encryption, Springer-Verlag, strony 363-366, 1994.

[98] D. Whitley, V. S. Gordon, K. Mathias. *Lamarckian Evolution, The Baldwin Effect and Function Optimization*. Parallel Problem Solving From Nature - PPSN III, strony 6-15, 1994.

[99] R. Wieczorkowski. *Algorytmy stochastyczne w optymalizacji dyskretnej przy zaburzonych wartościach funkcji*. Matematyka Stosowana 19(3), strony 119-153, 1995.

[100] A. H. Wright. *Genetic algorithms for real parameter optimization. In Foundations of genetic algorithms*. Foundations of Genetic Algorithms, strony 205-218, 1991.

[101] X. S. Yang. *A new metaheuristic bat-inspired algorithm*. Nature Inspired Cooperative Strategies for Optimization, strony 65-74, 2010.

[102] X. S. Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.

[103] X. S. Yang, S. Deb. *Cuckoo Search via Lévy Flights*. World Congress on Nature & Biologically Inspired Computing, strony 210-214, 2009.

[104] P. Zieliński. *Metody przeszukiwania lokalnego*. Katedra Informatyki, Politechnika Wrocławska. Dostępne w Internecie: http://cs.pwr.edu.pl/zielinski/lectures/om/localsearch.pdf [dostęp: 2022-02-02 15:00].

[105] B. Żółtak. *VMPC One-Way Function and Stream Cipher*. Fast Software Encryption, Springer-Verlag, strony 210-225, 2004.

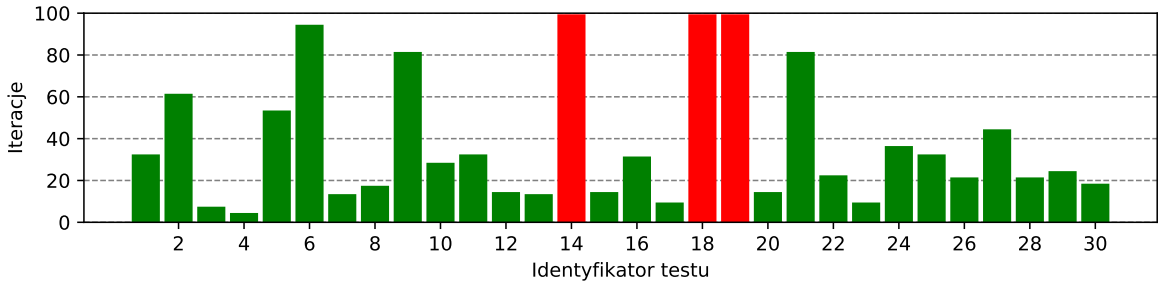
[106] Science Daily. *Challenging a central dogma of chemistry: Energy flow in chemical reactions*. Institute for Basic Science, Science Daily. Dostępne w Internecie: https://www.sciencedaily.com/releases/2020/07/200730141259.htm [dostęp: 2022-02-02 16:00].

[107] BBC.com. *Coronavirus and South Korea: How lives changed to beat the virus*. BBC World, https://www.bbc.com. Dostępne w Internecie: https://www.bbc.com/news/world-asia-52482553 [dostęp: 2022-02-02 16:00].

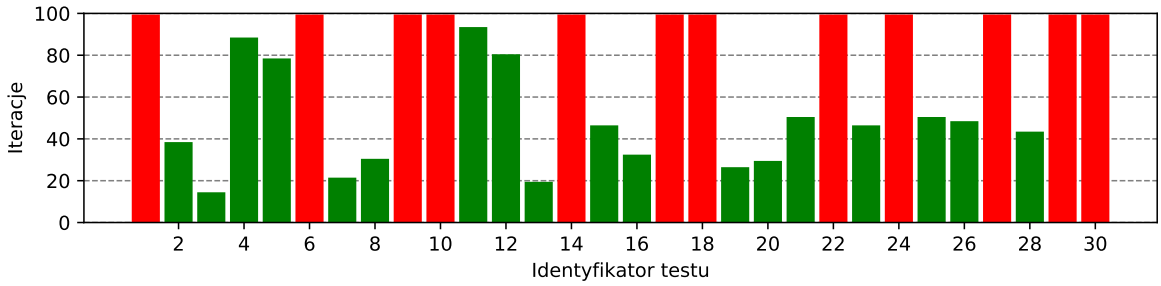
A

Dodatek

W dodatku tym przedstawiono szczegółowe wyniki badań dla algorytmów *MASA* i *NEA*, zarówno dla pierwszej Ω_1 oraz drugiej Ω_2 charakterystyki, wraz z wynikami algorytmu *DPSO* kolejno w tabelach A.1 - A.13. Ponadto, zamieszczono zestawienie poprawnie odgadniętych bitów dla każdego podklucza oraz ataku kryptoanalitycznego - rys. A.1 - A.10.



Rysunek A.1: Zestawienie poprawnie odgadniętych bitów alg. *MASA* dla Ω_2 - klucz #1



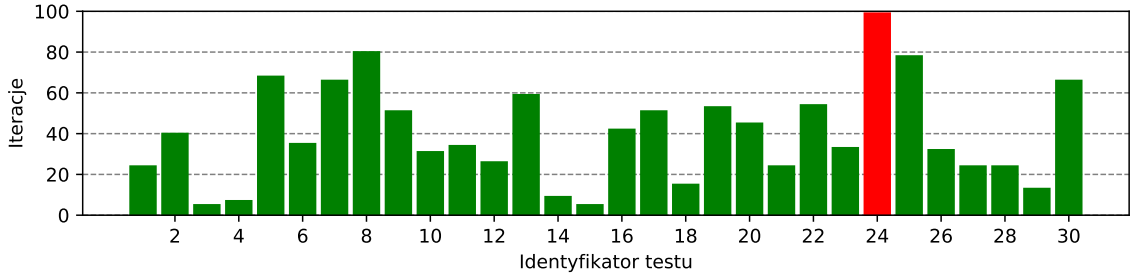
Rysunek A.2: Zestawienie poprawnie odgadniętych bitów alg. *NEA* dla Ω_2 - klucz #1

Tabela A.1: Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_1 - klucz #1

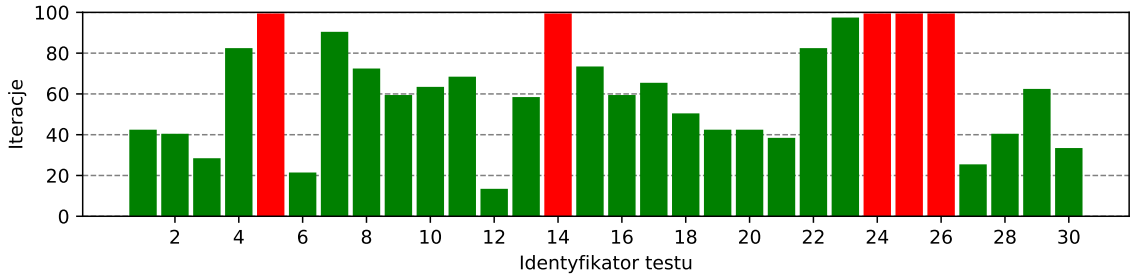
Klucz Szyfrujący	Id	*MASA*							*NEA*						
		Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.	Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.
34E9F71A20756231	6	968	1089	1067,7	1110	42,0	43	30	965	1085	1064,6	1110	49,3	19	30
	7	918	1090	1063,6	1110	52,7	6	30	**961**	**1090**	**1070,3**	**1098**	**41,9**	**99**	**28**
	8	1064	1090	1089,6	1110	9,1	4	30	1030	1089	1080,3	1110	27,9	9	30
	9	986	1090	1067,9	1110	38,4	23	30	1028	1090	1087,0	1110	21,9	52	30
	10	**1030**	**1090**	**1078,2**	**1098**	**27,4**	**99**	**28**	**985**	**1089**	**1070,1**	**1098**	**42,0**	**99**	**27**
	11	1002	1089	1082,2	1110	25,4	12	30	936	1090	1066,1	1110	52,7	20	30
	12	**1058**	**1089**	**1087,9**	**1098**	**12,0**	**99**	**27**	992	1038	1056,8	1110	42,9	28	30
	13	1058	1089	1081,1	1110	18,6	23	30	1043	1089	1082,8	1110	19,4	14	30
	14	969	1089	1069,3	1110	39,4	9	30	1001	1060	1073,4	1110	30,7	84	30
	15	1033	1090	1089,6	1110	16,2	6	30	**954**	**1097**	**1059,9**	**1098**	**55,7**	**99**	**27**
	16	1055	1090	1090,8	1110	10,8	6	30	962	1055	1048,8	1110	55,1	34	30
	17	953	1090	1071,3	1110	49,5	14	30	**1011**	**1089**	**1068,4**	**1098**	**32,9**	**99**	**29**
	18	1006	1090	1088,1	1110	22,0	13	30	**999**	**1066**	**1061,9**	**1098**	**32,3**	**99**	**28**
	19	1005	1090	1077,3	1110	36,9	10	30	1060	1090	1085,0	1110	17,7	21	30
	20	1017	1089	1085,9	1110	22,9	45	30	977	1063	1051,7	1110	46,2	12	30
	21	977	1090	1074,7	1110	34,4	5	30	**952**	**1090**	**1061,9**	**1098**	**47,4**	**99**	**28**
	22	1008	1075	1067,9	1110	32,1	8	30	958	1060	1059,8	1110	45,0	11	30
	23	1005	1090	1081,2	1098	22,9	7	30	**1030**	**1089**	**1081,7**	**1098**	**20,7**	**99**	**28**
	24	978	1090	1068,6	1110	39,3	10	30	**967**	**1090**	**1061,0**	**1098**	**44,8**	**99**	**28**
	25	971	1089	1069,4	1110	41,2	8	30	**1005**	**1089**	**1075,1**	**1098**	**29,1**	**99**	**27**
	26	1008	1090	1068,1	1110	34,9	11	30	1079	1090	1095,0	1110	9,3	43	30
	27	1026	1090	1077,1	1110	27,5	10	30	986	1084	1063,9	1098	36,7	36	30
	28	1061	1089	1089,0	1110	14,3	14	30	1042	1089	1087,6	1110	17,8	31	30
	29	982	1090	1070,1	1110	41,8	10	30	1065	1090	1087,5	1110	11,4	9	30
	30	1060	1089	1080,2	1098	14,1	14	30	**984**	**1061**	**1061,9**	**1098**	**40,8**	**99**	**28**

Tabela A.2: Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_2 - klucz #1

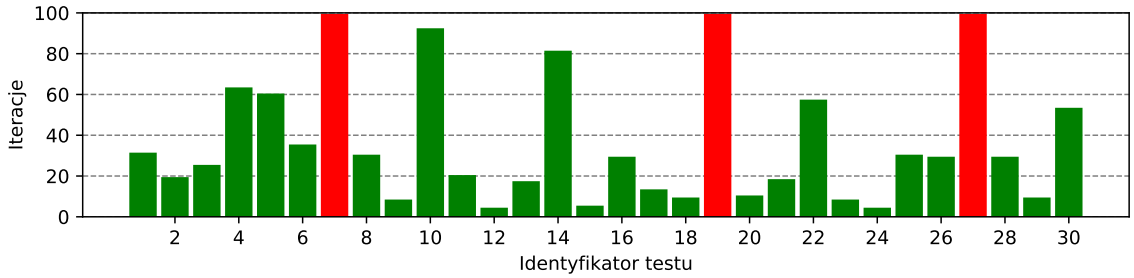
Klucz Szyfrujący	Id	*MASA*							*NEA*						
		Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.	Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.
34E9F71A20756231	6	942	998	1004,7	1085	43,0	94	30	**942**	**971**	**1000,6**	**1061**	**49,7**	**99**	**28**
	7	1007	1046	1051,9	1085	26,3	13	30	908	983	994,4	1053	54,4	21	30
	8	927	995	1015,0	1085	56,5	17	30	987	1046	1039,1	1085	35,1	30	30
	9	951	1002	1014,6	1085	40,4	81	30	**946**	**1037**	**1022,7**	**1061**	**37,7**	**99**	**29**
	10	995	1055	1048,6	1085	28,8	28	30	**899**	**1055**	**1007,9**	**1061**	**66,4**	**99**	**29**
	11	977	1005	1022,2	1085	39,8	32	30	874	1060	1027,7	1085	59,3	93	30
	12	876	1040	1009,3	1085	75,7	14	30	924	989	999,5	1085	66,4	80	30
	13	988	1054	1048,6	1085	32,3	13	30	975	1018	1031,9	1085	38,6	19	30
	14	**931**	**1001**	**1007,0**	**1053**	**33,8**	**99**	**28**	**944**	**1032**	**1023,0**	**1061**	**38,2**	**99**	**28**
	15	957	1055	1049,6	1085	42,9	14	30	985	1068	1052,8	1085	37,4	46	30
	16	958	989	1025,2	1085	46,5	31	30	871	999	1008,9	1085	75,4	32	30
	17	908	1040	1024,6	1085	59,6	9	30	**889**	**991**	**984,1**	**1061**	**55,2**	**99**	**28**
	18	**963**	**995**	**1015,3**	**1060**	**34,2**	**99**	**28**	**965**	**1001**	**1003,3**	**1033**	**18,4**	**99**	**28**
	19	**898**	**1006**	**1004,6**	**1061**	**55,5**	**99**	**28**	977	1053	1047,8	1085	37,7	26	30
	20	958	1053	1034,8	1085	39,3	14	30	880	998	982,5	1085	67,5	29	30
	21	973	1026	1027,3	1085	34,8	81	30	967	1010	1022,4	1085	40,4	50	30
	22	969	1030	1029,6	1085	34,4	22	30	**884**	**1009**	**991,4**	**1061**	**69,5**	**99**	**28**
	23	894	1041	1019,9	1085	57,0	9	30	896	989	1010,0	1085	56,1	46	30
	24	992	1054	1043,0	1085	28,0	36	30	**925**	**997**	**999,5**	**1053**	**38,7**	**99**	**27**
	25	961	1000	1011,9	1085	42,4	32	30	985	1036	1036,0	1085	35,3	50	30
	26	949	1025	1024,6	1085	47,6	21	30	1031	1060	1058,1	1085	13,1	48	30
	27	909	1011	1015,7	1085	50,5	44	30	**907**	**990**	**1012,7**	**1060**	**47,1**	**99**	**27**
	28	982	1009	1019,1	1085	36,0	21	30	971	1003	1023,9	1085	37,5	43	30
	29	978	985	1028,0	1085	46,6	24	30	**922**	**999**	**998,4**	**1061**	**40,0**	**99**	**28**
	30	904	1040	1011,6	1085	63,2	18	30	**994**	**1008**	**1024,6**	**1061**	**28,6**	**99**	**28**



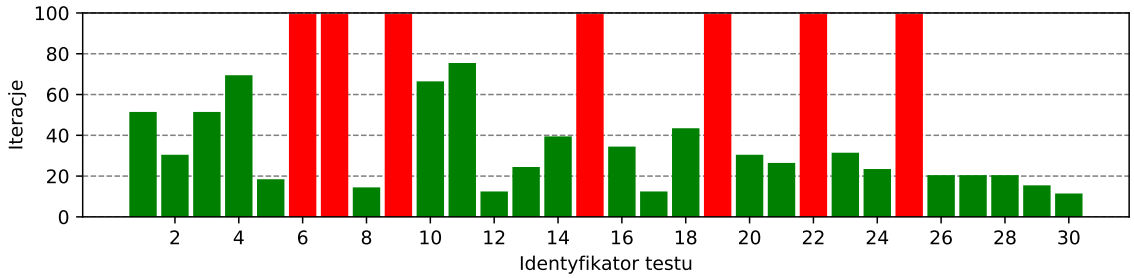
Rysunek A.3: Zestawienie poprawnie odgadniętych bitów alg. *MASA* dla Ω_2 - klucz #2



Rysunek A.4: Zestawienie poprawnie odgadniętych bitów alg. *NEA* dla Ω_2 - klucz #2



Rysunek A.5: Zestawienie poprawnie odgadniętych bitów alg. *MASA* dla Ω_2 - klucz #3



Rysunek A.6: Zestawienie poprawnie odgadniętych bitów alg. *NEA* dla Ω_2 - klucz #3

Tabela A.3: Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_1 - klucz #2

Klucz Szyfrujący	Id	*MASA*							*NEA*						
		Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.	Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.
BF8980377505B539	6	723	842	818,2	856	48,3	25	30	683	839	805,5	856	58,5	27	30
	7	687	844	815,6	856	58,6	35	30	722	839	810,8	856	50,5	9	30
	8	694	826	800,6	856	58,3	13	30	686	819	794,0	856	70,5	17	30
	9	714	781	789,0	856	52,3	10	30	791	856	839,9	856	24,3	38	30
	10	690	801	804,7	856	53,4	19	30	**767**	**777**	**791,8**	**811**	**19,4**	**99**	**28**
	11	728	837	822,2	856	48,3	22	30	682	826	811,5	856	52,4	51	30
	12	732	842	818,5	856	44,7	16	30	710	816	795,8	856	55,8	17	30
	13	683	794	791,1	856	51,5	7	30	699	842	817,8	856	49,7	16	30
	14	673	844	816,3	856	67,3	10	30	726	805	804,9	856	49,9	15	30
	15	725	839	817,9	856	48,0	13	30	714	839	818,1	856	47,6	16	30
	16	765	821	823,8	856	32,1	62	30	701	826	802,1	856	56,0	9	30
	17	765	814	811,0	856	35,4	8	30	735	817	820,0	856	36,3	39	30
	18	683	751	776,6	856	66,6	4	30	691	798	807,0	856	52,1	28	30
	19	720	844	827,3	856	40,6	8	30	638	806	789,9	856	70,5	18	30
	20	713	837	821,4	856	42,7	27	30	718	835	816,9	856	49,6	31	30
	21	728	816	818,3	856	40,3	38	30	805	840	832,7	856	23,2	45	30
	22	717	826	804,1	856	48,4	19	30	770	856	833,6	856	31,6	15	30
	23	738	801	797,7	856	38,6	8	30	776	844	839,6	856	24,4	24	30
	24	730	814	810,0	856	43,2	42	30	670	751	771,4	856	59,9	32	30
	25	713	826	811,1	856	46,4	11	30	778	805	822,8	856	34,2	20	30
	26	776	826	832,5	856	26,5	8	30	719	851	829,6	856	44,7	71	30
	27	686	822	802,1	856	63,6	30	30	**739**	**800**	**805,0**	**851**	**38,2**	**99**	**28**
	28	819	837	840,1	856	13,5	23	30	**719**	**814**	**809,2**	**832**	**33,2**	**99**	**27**
	29	681	842	799,8	856	65,9	40	30	781	856	834,1	856	28,5	70	30
	30	688	826	800,9	856	56,3	37	30	668	838	805,0	856	63,2	11	30

Tabela A.4: Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_2 - klucz #2

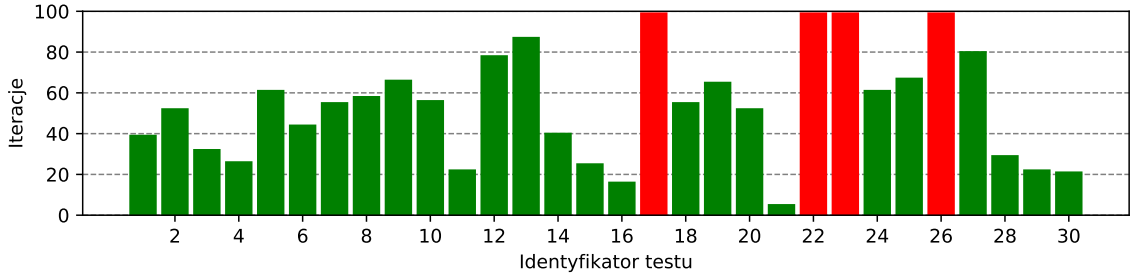
Klucz Szyfrujący	Id	*MASA*							*NEA*						
		Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.	Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.
BF8980377505B539	6	862	975	968,6	1056	76,3	35	30	873	956	956,5	1056	59,3	21	30
	7	871	957	961,0	1056	60,7	66	30	945	1002	1002,2	1056	30,8	90	30
	8	864	938	952,3	1038	60,4	80	30	863	935	959,7	1056	64,9	72	30
	9	884	1009	983,9	1056	60,1	51	30	888	1027	999,3	1056	63,5	59	30
	10	932	956	975,7	1056	44,0	31	30	868	915	944,2	1056	67,4	63	30
	11	895	993	992,1	1056	56,4	34	30	870	903	941,1	1056	69,3	68	30
	12	878	994	996,3	1056	56,0	26	30	860	925	956,2	1056	73,1	13	30
	13	886	1009	989,1	1056	58,6	59	30	876	969	981,9	1056	65,9	58	30
	14	870	932	952,4	1056	64,7	9	30	**878**	**960**	**964,9**	**1003**	**42,0**	**99**	**28**
	15	876	1033	993,8	1056	64,2	5	30	894	988	992,4	1056	58,2	73	30
	16	866	983	991,5	1056	63,3	42	30	927	971	985,4	1056	52,6	59	30
	17	924	982	994,6	1056	49,0	51	30	909	1012	1007,9	1056	41,0	65	30
	18	877	908	948,3	1056	70,0	15	30	868	958	962,6	1056	72,6	50	30
	19	936	976	1001,8	1056	47,7	53	30	882	992	979,3	1056	60,0	42	30
	20	948	1026	1011,9	1056	37,1	45	30	926	1007	1004,4	1056	44,6	42	30
	21	986	1041	1033,3	1056	24,3	24	30	899	968	981,5	1056	45,4	38	30
	22	901	1015	999,8	1056	52,4	54	30	883	1041	1011,5	1056	51,1	82	30
	23	875	938	958,4	1056	65,5	33	30	853	940	937,9	1006	51,9	97	30
	24	**879**	**962**	**975,2**	**1041**	**61,0**	**99**	**28**	**913**	**982**	**983,5**	**1041**	**43,5**	**99**	**29**
	25	1011	1026	1027,4	1056	10,7	78	30	**874**	**962**	**966,0**	**1026**	**55,1**	**99**	**27**
	26	876	972	970,8	1056	53,2	32	30	**863**	**934**	**953,6**	**1042**	**59,6**	**99**	**27**
	27	892	945	973,5	1056	53,3	24	30	891	927	961,8	1056	59,2	25	30
	28	891	965	969,8	1056	53,8	24	30	874	989	980,6	1056	53,8	40	30
	29	858	959	963,7	1056	74,2	13	30	920	1015	1004,1	1056	47,7	62	30
	30	875	977	981,0	1056	51,0	66	30	885	991	981,7	1056	56,3	33	30

Tabela A.5: Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_1 - klucz #3

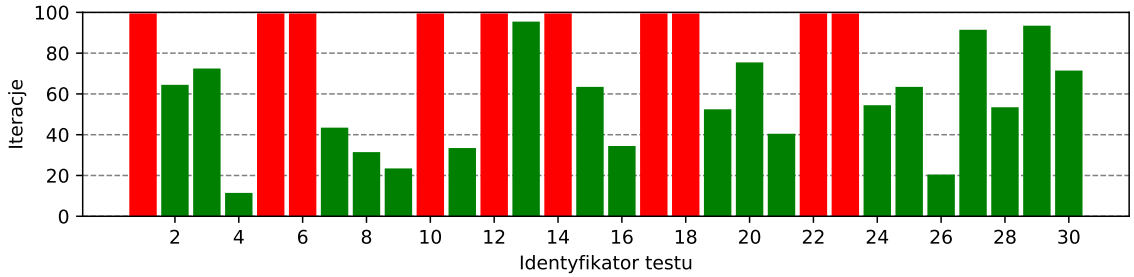
Klucz Szyfrujący	Id	*MASA*							*NEA*						
		Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.	Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.
0D8E27040A46F6C2	6	**866**	**1008**	**1008,8**	**1101**	**84,3**	**99**	**28**	913	1040	1040,4	1106	74,5	24	30
	7	885	1089	1056,5	1106	68,5	27	30	842	1016	1026,2	1106	78,9	30	30
	8	891	1101	1058,0	1106	71,5	20	30	1002	1062	1064,2	1101	28,3	39	30
	9	995	1074	1071,1	1106	37,8	17	30	963	1085	1062,3	1106	50,3	77	30
	10	910	1064	1048,3	1106	65,7	25	30	**929**	**1058**	**1043,1**	**1101**	**62,6**	**99**	**28**
	11	884	1061	1016,2	1106	89,9	21	30	940	1093	1055,7	1106	61,1	43	30
	12	1017	1101	1070,7	1106	39,3	49	30	**855**	**967**	**986,4**	**1043**	**64,2**	**99**	**24**
	13	906	1060	1058,5	1106	58,3	10	30	852	1031	1015,5	1106	95,4	24	30
	14	934	1101	1056,3	1106	66,1	33	30	**836**	**925**	**976,4**	**1101**	**89,4**	**99**	**28**
	15	1026	1095	1088,4	1106	23,4	33	30	**870**	**1010**	**1004,8**	**1101**	**88,3**	**99**	**28**
	16	1000	1093	1075,0	1106	36,2	30	30	**887**	**963**	**979,2**	**1032**	**43,0**	**99**	**25**
	17	873	1049	1033,7	1106	79,6	43	30	969	1095	1065,7	1106	48,3	77	30
	18	978	1089	1073,4	1106	47,1	34	30	**872**	**954**	**996,7**	**1101**	**82,0**	**99**	**28**
	19	889	1074	1036,1	1106	76,1	11	30	**924**	**982**	**1016,2**	**1095**	**56,0**	**99**	**28**
	20	**933**	**995**	**1012,8**	**1069**	**47,3**	**99**	**29**	938	1101	1057,4	1106	62,7	37	30
	21	910	1054	1041,5	1106	68,1	27	30	**940**	**999**	**1022,5**	**1101**	**59,5**	**99**	**29**
	22	1089	1093	1095,3	1106	6,1	23	30	**918**	**1101**	**1074,2**	**1101**	**55,7**	**99**	**28**
	23	1089	1101	1098,4	1106	6,4	29	30	865	1020	1020,7	1106	71,5	93	30
	24	902	1035	1051,9	1106	61,9	41	30	859	1042	1011,7	1106	87,2	74	30
	25	902	1093	1054,8	1106	68,9	19	30	**850**	**1020**	**1005,9**	**1101**	**86,1**	**99**	**28**
	26	**864**	**953**	**979,6**	**1101**	**75,4**	**99**	**27**	**1023**	**1054**	**1055,8**	**1069**	**13,5**	**99**	**28**
	27	961	1091	1067,1	1106	52,2	79	30	927	996	1024,0	1106	62,8	24	30
	28	871	1048	1038,1	1106	62,9	32	30	881	1003	1007,1	1106	80,0	66	30
	29	914	1101	1049,0	1106	71,2	52	30	864	957	990,2	1106	84,9	54	30
	30	951	1068	1060,3	1106	54,0	66	30	**858**	**1029**	**1009,7**	**1101**	**97,4**	**99**	**28**

Tabela A.6: Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_2 - klucz #3

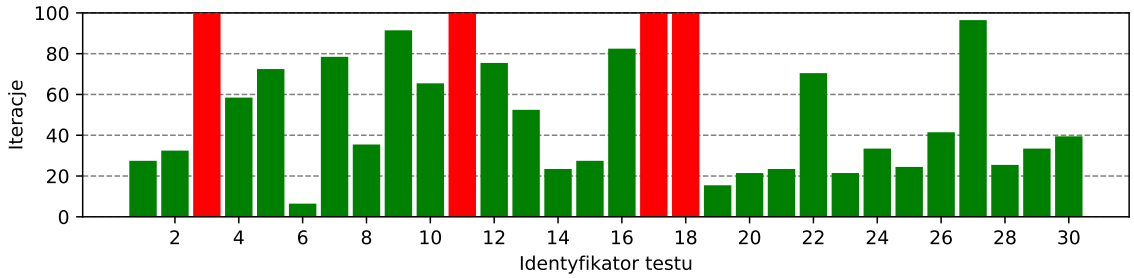
Klucz Szyfrujący	Id	*MASA*							*NEA*						
		Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.	Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.
0D8E27040A46F6C2	6	940	1015	1004,4	1051	36,8	35	30	**891**	**971**	**974,8**	**1037**	**47,7**	**99**	**28**
	7	**873**	**983**	**978,2**	**1037**	**46,1**	**99**	**28**	**916**	**970**	**984,1**	**1037**	**35,0**	**99**	**26**
	8	950	1037	1023,5	1051	29,4	30	30	926	981	982,3	1015	30,4	14	30
	9	957	1025	1018,1	1051	27,2	8	30	**827**	**954**	**945,7**	**1037**	**66,6**	**99**	**28**
	10	936	1015	1003,8	1051	34,0	92	30	982	1015	1014,4	1051	19,1	66	30
	11	967	1015	1018,7	1051	20,5	20	30	981	1012	1009,6	1051	21,9	75	30
	12	942	1016	1007,6	1051	32,0	4	30	847	1024	996,3	1051	67,7	12	30
	13	923	1015	1007,3	1051	36,0	17	30	847	1015	995,5	1051	57,0	24	30
	14	904	958	970,9	1051	54,6	81	30	891	1015	998,6	1051	49,8	39	30
	15	924	1015	1012,9	1051	37,3	5	30	**951**	**991**	**996,6**	**1037**	**30,7**	**99**	**28**
	16	971	1015	1016,6	1051	19,0	29	30	840	959	960,8	1051	72,9	34	30
	17	936	993	1004,7	1051	34,1	13	30	809	965	970,8	1051	70,2	12	30
	18	885	983	985,9	1051	61,0	9	30	907	991	989,4	1051	50,2	43	30
	19	**930**	**991**	**995,4**	**1037**	**38,3**	**99**	**28**	**844**	**961**	**964,9**	**1037**	**63,9**	**99**	**28**
	20	948	1015	1006,1	1051	34,3	10	30	920	997	996,9	1051	43,6	30	30
	21	908	1020	1005,9	1051	43,6	18	30	965	1016	1017,2	1051	29,0	26	30
	22	952	999	1000,1	1037	25,1	57	30	**827**	**955**	**945,3**	**1020**	**73,4**	**99**	**28**
	23	901	1020	1008,4	1051	43,8	8	30	1012	1027	1028,7	1051	10,6	31	30
	24	888	980	994,0	1051	46,9	4	30	918	1010	1003,5	1051	38,3	23	30
	25	864	985	991,1	1051	56,2	30	30	**980**	**994**	**1002,2**	**1027**	**15,9**	**99**	**28**
	26	985	1012	1018,1	1051	19,1	29	30	901	988	988,0	1051	54,6	20	30
	27	**916**	**943**	**973,5**	**1037**	**44,5**	**99**	**28**	837	964	977,0	1051	70,1	20	30
	28	1012	1027	1026,8	1051	11,6	29	30	903	1023	993,3	1051	58,2	20	30
	29	906	1020	1004,9	1051	46,2	9	30	913	950	979,5	1051	53,9	15	30
	30	979	1015	1016,2	1051	16,5	53	30	903	988	988,4	1051	47,5	11	30



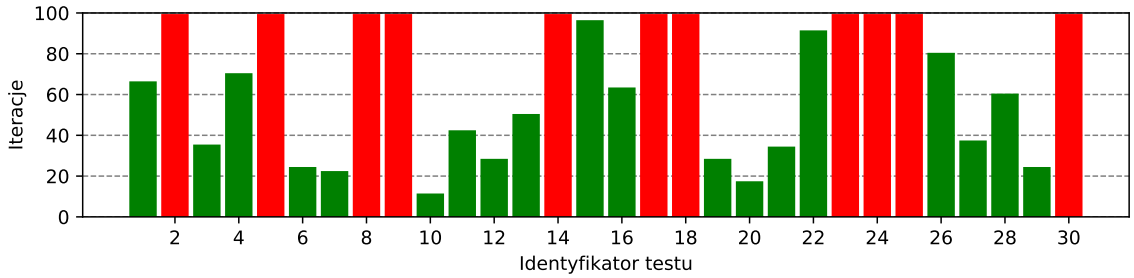
Rysunek A.7: Zestawienie poprawnie odgadniętych bitów alg. *MASA* dla Ω_2 - klucz #4



Rysunek A.8: Zestawienie poprawnie odgadniętych bitów alg. *NEA* dla Ω_2 - klucz #4



Rysunek A.9: Zestawienie poprawnie odgadniętych bitów alg. *MASA* dla Ω_2 - klucz #5



Rysunek A.10: Zestawienie poprawnie odgadniętych bitów alg. *NEA* dla Ω_2 - klucz #5

Tabela A.7: Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_1 - klucz #4

Klucz Szyfrujący	Id	*MASA*							*NEA*						
		Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.	Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.
A49ECC6471E38D65	6	892	972	973,7	1038	56,2	68	30	915	968	983,6	1038	40,0	31	30
	7	892	978	969,4	1038	48,7	8	30	**914**	**995**	**993,6**	**1025**	**32,1**	**99**	**29**
	8	894	976	979,0	1038	48,1	9	30	916	974	989,0	1038	44,0	41	30
	9	922	1025	1004,1	1038	40,9	25	30	895	982	982,2	1038	55,5	28	30
	10	918	966	980,2	1038	36,7	13	30	914	973	987,2	1038	47,4	32	30
	11	893	968	973,1	1038	47,6	17	30	900	938	953,8	1038	51,1	71	30
	12	915	1015	1002,0	1038	36,1	39	30	926	999	991,9	1038	43,7	45	30
	13	923	1015	998,7	1038	39,5	20	30	**883**	**975**	**967,9**	**1025**	**52,5**	**99**	**28**
	14	**911**	**982**	**978,9**	**1023**	**40,5**	**99**	**27**	**889**	**941**	**951,4**	**1025**	**45,4**	**99**	**28**
	15	949	1013	1000,6	1038	31,7	32	30	**957**	**990**	**991,2**	**1025**	**22,5**	**99**	**27**
	16	928	1020	1008,0	1038	34,0	45	30	903	992	983,4	1038	51,1	80	30
	17	937	1006	1001,9	1038	32,4	50	30	960	989	1002,9	1038	29,7	68	30
	18	908	991	984,0	1038	46,7	17	30	**899**	**950**	**974,0**	**1028**	**49,2**	**99**	**29**
	19	905	964	974,7	1038	51,4	39	30	906	984	992,2	1038	43,3	41	30
	20	912	984	989,0	1038	43,3	33	30	912	975	985,3	1038	42,8	34	30
	21	903	1000	988,3	1038	48,1	45	30	**884**	**983**	**977,2**	**1013**	**41,4**	**99**	**27**
	22	902	961	975,4	1038	45,6	69	30	900	969	971,9	1038	40,0	54	30
	23	928	965	985,7	1038	42,5	8	30	894	950	967,2	1038	48,5	32	30
	24	887	966	968,4	1038	47,9	42	30	973	1015	1016,4	1038	20,7	59	30
	25	937	977	993,0	1038	35,4	78	30	911	988	986,4	1038	41,8	28	30
	26	929	990	993,3	1038	35,5	50	30	**918**	**985**	**986,9**	**1028**	**37,7**	**99**	**28**
	27	**899**	**946**	**962,1**	**1008**	**38,5**	**99**	**27**	**896**	**959**	**949,4**	**995**	**35,8**	**99**	**27**
	28	904	997	995,5	1038	44,2	23	30	939	977	990,4	1038	38,9	31	30
	29	941	987	1000,1	1038	37,0	56	30	889	976	973,1	1038	53,1	15	30
	30	910	994	995,9	1038	39,4	11	30	885	976	975,9	1038	49,7	50	30

Tabela A.8: Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_2 - klucz #4

Klucz Szyfrujący	Id	*MASA*							*NEA*						
		Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.	Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.
A49ECC6471E38D65	6	984	1042	1051,2	1101	32,6	44	30	**970**	**1054**	**1056,0**	**1095**	**36,1**	**99**	**28**
	7	975	1040	1056,6	1101	40,4	55	30	1049	1095	1079,9	1101	21,1	43	30
	8	975	1042	1048,4	1101	34,3	58	30	986	1067	1062,1	1101	32,9	31	30
	9	1005	1056	1055,7	1101	31,1	66	30	956	1047	1038,3	1101	48,6	23	30
	10	1023	1068	1064,6	1101	26,3	56	30	**913**	**1035**	**1024,3**	**1086**	**47,3**	**99**	**28**
	11	1030	1046	1057,3	1101	26,5	22	30	959	1005	1017,7	1101	48,7	33	30
	12	1041	1052	1066,3	1101	21,0	78	30	**953**	**997**	**1007,3**	**1071**	**44,9**	**99**	**27**
	13	961	1037	1030,2	1101	49,6	87	30	930	1039	1031,1	1101	54,0	95	30
	14	977	1041	1045,3	1101	37,6	40	30	**894**	**1052**	**1023,6**	**1095**	**64,6**	**99**	**28**
	15	967	1036	1046,3	1101	42,8	25	30	956	999	1018,5	1101	51,1	63	30
	16	986	1057	1066,4	1101	32,1	16	30	894	1022	1003,3	1101	69,4	34	30
	17	**967**	**1037**	**1043,7**	**1095**	**34,8**	**99**	**28**	**901**	**1040**	**1042,6**	**1095**	**53,4**	**99**	**28**
	18	1026	1063	1059,2	1101	21,9	55	30	**911**	**1037**	**1009,8**	**1095**	**59,5**	**99**	**28**
	19	1020	1063	1064,5	1101	24,5	65	30	971	1034	1036,3	1101	41,2	52	30
	20	1017	1052	1056,0	1101	24,9	52	30	1006	1063	1063,3	1101	28,0	75	30
	21	922	1063	1055,8	1101	47,6	5	30	976	1046	1050,5	1101	36,3	40	30
	22	**978**	**1046**	**1045,8**	**1095**	**38,8**	**99**	**28**	**896**	**976**	**1000,9**	**1089**	**69,1**	**99**	**26**
	23	**898**	**1034**	**1018,1**	**1071**	**52,0**	**99**	**26**	**887**	**1041**	**1018,6**	**1086**	**65,6**	**99**	**27**
	24	977	1052	1059,7	1101	37,8	61	30	1025	1064	1065,7	1101	22,2	54	30
	25	990	1059	1055,8	1101	31,5	67	30	949	1034	1026,5	1101	52,4	63	30
	26	**1027**	**1067**	**1069,8**	**1095**	**21,5**	**99**	**28**	890	1055	1035,6	1101	62,1	20	30
	27	963	1056	1051,4	1101	44,6	80	30	955	1035	1033,3	1101	53,0	91	30
	28	981	1046	1046,1	1101	43,0	29	30	954	1034	1034,2	1101	51,9	53	30
	29	966	1057	1045,9	1101	45,9	22	30	1013	1054	1061,3	1101	29,3	93	30
	30	955	1034	1032,2	1101	42,8	21	30	994	1064	1060,5	1101	33,2	71	30

Tabela A.9: Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_1 - klucz #5

Klucz Szyfrujący	Id	*MASA*							*NEA*						
		Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.	Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.
EB2275E5FFB1F2C4	6	898	923	944,9	1014	42,8	60	30	**851**	**970**	**941,1**	**998**	**50,0**	**99**	**27**
	7	893	1014	969,9	1014	54,4	6	30	**911**	**963**	**960,3**	**988**	**25,6**	**99**	**27**
	8	891	950	956,9	1014	47,1	59	30	**931**	**998**	**994,4**	**1014**	**23,9**	**99**	**28**
	9	943	997	997,2	1014	20,2	62	30	**873**	**950**	**959,6**	**1014**	**45,9**	**99**	**27**
	10	942	980	980,0	1014	25,9	87	30	919	984	984,1	1014	25,3	26	30
	11	899	953	957,1	1014	42,3	16	30	942	976	976,2	1014	24,8	98	30
	12	884	939	944,4	1014	51,4	27	30	**863**	**930**	**936,6**	**1014**	**47,8**	**99**	**27**
	13	895	965	966,7	1014	45,5	39	30	934	1014	995,3	1014	31,0	23	30
	14	953	997	999,4	1014	17,4	45	30	**894**	**918**	**942,3**	**997**	**44,8**	**99**	**29**
	15	940	986	984,1	1014	21,6	53	30	**886**	**946**	**949,3**	**997**	**36,2**	**99**	**28**
	16	949	984	988,5	1014	20,9	32	30	**947**	**984**	**981,1**	**1014**	**17,4**	**99**	**26**
	17	909	1000	986,2	1014	37,6	26	30	895	967	965,2	1014	33,2	92	30
	18	905	984	983,3	1014	33,8	23	30	**879**	**974**	**958,9**	**988**	**39,4**	**99**	**25**
	19	**967**	**998**	**993,6**	**1010**	**13,7**	99	**28**	**883**	**931**	**949,1**	**998**	**39,3**	**99**	**29**
	20	897	919	942,9	1014	43,4	93	30	905	989	973,5	1014	42,0	23	30
	21	908	966	976,8	1014	34,4	18	30	861	925	937,7	1014	57,6	81	30
	22	879	996	966,7	1014	53,9	10	30	885	958	955,0	1014	50,8	25	30
	23	972	984	992,0	1014	13,7	82	30	927	982	976,0	1014	31,7	16	30
	24	943	974	979,5	1014	22,0	67	30	917	984	980,2	1014	34,8	75	30
	25	898	983	974,9	1014	39,9	11	30	843	919	938,8	1014	57,5	40	30
	26	940	989	986,0	1014	26,6	48	30	863	912	936,8	1014	51,9	77	30
	27	**901**	**929**	**945,0**	**1010**	**36,9**	99	**28**	867	943	957,2	1014	49,3	13	30
	28	930	998	996,9	1014	24,4	48	30	868	909	945,1	1014	55,4	11	30
	29	885	992	970,6	1014	44,5	30	30	897	983	971,8	1014	43,4	56	30
	30	981	996	999,2	1014	14,6	76	30	919	994	985,0	1014	35,0	26	30

Tabela A.10: Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_2 - klucz #5

Klucz Szyfrujący	Id	*MASA*							*NEA*						
		Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.	Min.	Med.	Śr.	Max.	Odch. Std.	t	Bit.
EB2275E5FFB1F2C4	6	944	990	1012,0	1095	58,4	6	30	957	1026	1029,5	1095	47,8	24	30
	7	953	1058	1039,8	1095	50,4	78	30	956	1012	1018,6	1095	40,1	22	30
	8	949	1027	1027,7	1095	53,1	35	30	**890**	**998**	**1007,9**	**1075**	**70,3**	**99**	**27**
	9	959	975	1009,1	1068	43,6	91	30	**979**	**1037**	**1031,2**	**1074**	**28,2**	**99**	**26**
	10	957	1007	1025,0	1095	48,8	65	30	942	1029	1013,3	1095	50,5	11	30
	11	**943**	**968**	**1000,4**	**1074**	**49,6**	**99**	**28**	916	1038	1038,5	1095	47,5	42	30
	12	1014	1074	1063,4	1095	25,3	75	30	947	1010	1021,6	1095	55,6	28	30
	13	941	1041	1039,9	1095	47,1	52	30	1018	1061	1059,4	1095	24,6	50	30
	14	946	1041	1044,6	1095	44,0	23	30	**975**	**1015**	**1027,4**	**1068**	**26,8**	**99**	**27**
	15	950	1029	1020,1	1095	55,6	27	30	984	1017	1025,9	1095	33,0	96	30
	16	934	1029	1038,1	1095	57,0	82	30	992	1041	1046,3	1095	35,5	63	30
	17	**964**	**1014**	**1022,2**	**1074**	**32,2**	**99**	**27**	**946**	**1009**	**1013,5**	**1075**	**47,7**	**99**	**27**
	18	**945**	**1017**	**1030,6**	**1074**	**36,4**	**99**	**29**	**906**	**954**	**989,5**	**1074**	**65,1**	**99**	**28**
	19	954	1020	1025,9	1095	52,0	15	30	887	994	987,4	1095	66,6	28	30
	20	961	1027	1033,0	1095	47,9	21	30	934	1011	1016,6	1095	60,3	17	30
	21	952	1018	1028,4	1095	50,4	23	30	895	1029	1017,1	1095	57,9	34	30
	22	1010	1047	1052,2	1095	26,1	70	30	944	1012	1011,6	1095	48,0	91	30
	23	967	1023	1036,3	1095	42,8	21	30	**960**	**1027**	**1026,3**	**1074**	**41,0**	**99**	**25**
	24	962	1002	1015,0	1095	40,6	33	30	**888**	**1027**	**993,9**	**1054**	**58,7**	**99**	**25**
	25	942	961	1002,4	1095	59,0	24	30	**895**	**1015**	**1008,3**	**1074**	**56,2**	**99**	**26**
	26	943	1007	1024,1	1095	51,6	41	30	970	1048	1031,8	1095	36,0	80	30
	27	953	992	1017,0	1095	54,7	96	30	967	1006	1039,7	1095	49,6	37	30
	28	952	1015	1026,0	1095	60,0	25	30	908	1033	1028,2	1095	52,4	60	30
	29	962	1023	1025,8	1095	49,6	33	30	919	997	1022,7	1095	62,3	24	30
	30	956	1021	1024,1	1095	46,8	39	30	**969**	**1018**	**1025,5**	**1062**	**30,0**	**99**	**29**

Tabela A.11: Wartości funkcji przystosowania algorytmu *DPSO* dla szyfru *DES* - klucze #1 i #2

Id	34E9F71A20756231						BF8980377505B539					
	Min.	Med.	Śr.	Max.	Odch. Std.	Bit.	Min.	Med.	Śr.	Max.	Odch. Std.	Bit.
6	52582,8	52736,2	52740,3	52874,0	68,6	44	52578,2	52742,6	52747,7	52900,0	67,6	42
7	52617,4	52743,4	52746,2	52998,4	67,4	40	52603,0	52738,0	52740,6	52921,6	66,6	42
8	52571,0	52740,0	52730,9	52823,0	53,2	42	52547,4	52739,8	52729,2	52885,8	64,2	41
9	52609,0	52768,6	52747,0	52854,2	64,3	42	52616,2	52739,6	52744,8	52915,2	62,8	41
10	52601,2	52740,2	52744,9	52915,8	65,2	42	52567,4	52743,6	52742,3	52856,4	57,8	42
11	52603,0	52730,0	52729,1	52968,6	72,8	41	52538,8	52729,8	52727,8	52870,6	63,7	42
12	52619,8	52742,2	52745,9	52894,4	53,9	43	52632,0	52739,4	52737,5	52884,0	61,8	43
13	52609,4	52735,4	52735,4	52876,2	66,1	42	52629,6	52739,6	52738,7	52884,0	54,6	40
14	52590,0	52755,6	52743,0	52916,8	63,4	42	52623,4	52734,6	52747,5	52932,8	72,4	45
15	52545,2	52743,6	52745,5	52915,2	90,4	40	52642,0	52734,8	52739,4	52914,4	58,1	41
16	**52575,2**	**52747,0**	**52733,5**	**52855,4**	**62,2**	**39**	52614,8	52745,4	52750,0	52917,2	63,2	42
17	**52596,8**	**52728,2**	**52727,7**	**52879,2**	**63,1**	**39**	52590,4	52754,0	52748,7	52925,6	76,8	43
18	52596,6	52729,8	52730,8	52867,2	58,4	40	52626,8	52741,2	52745,1	52877,2	56,8	41
19	52566,2	52708,6	52723,8	52874,0	72,8	44	52588,6	52706,4	52711,8	52887,0	68,3	41
20	52560,2	52731,0	52728,6	52949,2	74,0	40	52563,8	52733,0	52733,5	52883,2	68,5	40
21	**52590,0**	**52742,6**	**52746,5**	**52946,2**	**66,3**	**39**	52621,2	52726,0	52738,2	53007,4	72,0	43
22	52618,4	52734,4	52731,4	52832,8	57,2	44	52592,4	52751,4	52744,1	52861,6	60,7	42
23	52510,4	52733,0	52728,8	52832,0	69,1	40	52603,0	52752,8	52739,5	52928,0	73,5	41
24	**52598,4**	**52756,8**	**52744,4**	**52892,8**	**75,9**	**39**	52523,4	52751,4	52735,1	52886,2	70,9	42
25	52572,6	52729,8	52731,6	52890,6	70,3	40	52617,4	52723,6	52730,0	52877,8	57,1	42
26	52624,2	52734,2	52741,9	52905,0	68,6	41	52552,0	52760,4	52743,8	52857,0	69,7	42
27	**52547,0**	**52741,8**	**52741,8**	**52877,2**	**69,1**	**39**	52625,4	52739,6	52730,8	52821,2	48,6	41
28	52580,6	52715,4	52718,9	52837,6	59,6	40	52584,6	52744,2	52738,4	52893,8	65,4	43
29	52546,6	52731,6	52732,3	52871,8	74,7	40	52626,6	52740,0	52735,7	52843,6	57,8	43
30	52588,0	52742,0	52734,9	52855,0	66,0	41	52595,0	52754,0	52755,3	52916,6	67,6	41

Tabela A.12: Wartości funkcji przystosowania algorytmu *DPSO* dla szyfru *DES* - klucze #3 i #4

Id	0D8E27040A46F6C2						A49ECC6471E38D65					
	Min.	Med.	Śr.	Max.	Odch. Std.	Bit.	Min.	Med.	Śr.	Max.	Odch. Std.	Bit.
6	52637,4	52749,0	52745,6	52909,0	57,3	43	**52616,2**	**52747,6**	**52745,9**	**52895,4**	**67,9**	**39**
7	52629,6	52731,8	52738,0	52854,8	56,7	40	52587,8	52727,8	52733,8	52948,8	73,7	43
8	52590,0	52724,2	52731,5	52877,8	67,8	41	52594,4	52741,0	52740,7	52877,8	68,0	43
9	52587,6	52730,0	52734,1	52879,2	56,0	41	52588,6	52745,4	52736,9	52852,4	65,1	40
10	52562,8	52742,8	52729,3	52940,0	81,5	41	52528,2	52727,8	52723,2	52859,8	76,3	41
11	52597,8	52715,8	52730,5	52890,4	67,2	41	52572,8	52716,2	52722,9	52891,2	66,2	44
12	52587,4	52757,8	52748,8	52846,8	55,2	40	52635,4	52725,4	52729,7	52894,6	55,3	41
13	52609,6	52741,0	52742,3	52888,4	66,0	40	**52577,2**	**52738,8**	**52729,6**	**52829,0**	**60,6**	**39**
14	52615,2	52736,4	52741,3	52859,6	54,0	41	52581,8	52729,6	52726,2	52921,2	72,1	41
15	52597,4	52723,8	52739,4	52924,0	70,7	43	52531,0	52750,8	52745,8	52904,4	70,1	44
16	52563,6	52747,0	52740,2	52885,2	69,4	46	52604,8	52763,0	52754,0	52922,6	61,5	43
17	52585,2	52728,4	52727,0	52914,2	77,5	40	52552,2	52737,4	52717,5	52834,4	67,6	42
18	52583,4	52740,8	52738,2	52911,8	66,7	43	52610,6	52733,8	52732,0	52927,6	66,0	42
19	52554,8	52742,2	52733,3	52845,6	59,1	40	52596,6	52743,0	52729,8	52869,4	64,3	43
20	**52529,8**	**52728,6**	**52729,7**	**52860,2**	**68,6**	**39**	52550,4	52740,6	52736,4	52862,6	71,2	41
21	52600,8	52744,2	52735,9	52866,8	63,9	44	52587,4	52728,2	52729,2	52896,0	61,8	44
22	52611,0	52721,8	52731,0	52883,8	65,3	41	52587,2	52721,6	52718,1	52882,8	59,8	42
23	52602,0	52763,0	52752,2	52872,6	62,8	41	52614,2	52763,2	52768,8	52911,4	73,9	42
24	52596,6	52728,6	52734,2	52911,8	69,9	42	52602,0	52719,2	52717,5	52828,8	56,4	40
25	52529,8	52750,4	52744,9	52878,2	68,1	41	52526,6	52744,2	52733,4	52892,4	62,3	42
26	52542,8	52742,6	52741,3	52873,6	65,4	42	52614,4	52718,4	52725,7	52867,0	54,9	42
27	52577,4	52732,0	52732,4	52874,2	65,2	41	52573,6	52728,0	52724,4	52877,8	67,8	43
28	52623,4	52755,0	52745,8	52885,2	67,1	40	52602,0	52727,6	52717,0	52870,8	71,6	44
29	52591,8	52747,4	52751,4	52923,2	76,3	46	52542,4	52738,4	52729,9	52862,4	69,4	41
30	**52557,8**	**52729,6**	**52723,2**	**52823,0**	**61,7**	**39**	52544,2	52731,2	52726,6	52884,4	79,1	40

Tabela A.13: Wartości funkcji przystosowania algorytmu *DPSO* dla szyfru *DES* - klucz # 5

Klucz Szyfrujący	Id	*DPSO*					
		Min.	Med.	Śr.	Max.	Odch. Std.	Bit.
EB2275E5FFB1F2C4	6	52584,8	52732,0	52731,1	52877,6	66,1	45
	7	52573,4	52747,0	52751,5	52888,4	65,2	42
	8	52543,0	52751,2	52740,6	52874,0	69,0	41
	9	52594,6	52742,6	52741,8	52848,0	53,5	40
	10	52609,8	52741,4	52737,6	52836,2	57,6	40
	11	52522,2	52730,6	52725,5	52942,4	81,5	43
	12	**52575,8**	**52738,6**	**52739,2**	**52886,8**	**70,4**	**39**
	13	52509,0	52748,0	52743,9	52865,8	65,5	40
	14	52541,6	52717,8	52715,2	52837,6	69,0	40
	15	52524,8	52736,6	52721,9	52884,0	74,7	43
	16	52580,0	52763,0	52743,8	52861,6	64,7	41
	17	52548,8	52737,8	52744,9	52895,4	64,3	41
	18	52596,4	52739,0	52744,2	52837,2	56,6	41
	19	52627,2	52737,8	52742,4	52867,4	53,6	41
	20	52599,2	52725,6	52736,3	52837,2	56,9	44
	21	52612,6	52756,8	52755,1	52903,2	65,6	47
	22	52551,6	52734,6	52736,4	52898,8	60,5	40
	23	52608,2	52730,8	52737,1	52872,6	61,8	40
	24	52613,6	52705,2	52721,1	52876,6	60,2	42
	25	52544,4	52729,8	52728,9	52870,8	70,4	43
	26	52483,0	52724,8	52719,4	52826,2	67,5	41
	27	52611,4	52731,8	52733,2	52886,8	59,1	42
	28	52611,4	52731,6	52734,6	52902,8	56,8	43
	29	52609,6	52726,4	52736,2	52951,0	76,5	43
	30	52527,2	52731,6	52731,8	52871,2	69,7	42

B

Dodatek

Id	Pary tekstów jawnych		Pary szyfrogramów	
Klucz 1: 34E9F71A20756231				
1	F002C4E225E460B7	B00AC4E221E460B7	7AC053F39D03BB53	99049D09155F452D
2	4797BDB0431DCD65	079FBDB0471DCD65	F28BCE0FCC3F1CD3	4D69F8D1728102E4
3	E075F20F2792068B	A07DF20F2392068B	160FCDD3041040D8	66F7164A6190AFFE
4	99A658886D272452	D9AE588869272452	AAD7DCFF4C5B07AD	34B549C5A40A953E
5	263C64B1E9C3D749	663464B1EDC3D749	C3F780786DFD5E8C	6B3F8DB49E78AC05
6	4DEDA240D829B580	0DE5A240DC29B580	8C218B1A2946667D	B250E72386D88896
7	7BF087FB32C217B6	3BF887FB36C217B6	7C607E908F40FC20	4F03B5DB53F863A7
8	1A9DF22DEE58119B	5A95F22DEA58119B	FE0AB71460231F59	90D1E4689AB21D36
9	A7146A0A37FA6AAA	E71C6A0A33FA6AAA	D4FCA411636C1990	F71E324863F5E259
10	4B5941BF24A29973	0B5141BF20A29973	697789F829A29037	CAC5FF3B08F4C0D1
11	7BAD3ABAE6D01CA1	3BA53ABAE2D01CA1	270E84648AF8A0AC	F2E1A28FB5E62D07
12	7607E909FAAE4AE6	360FE909FEAE4AE6	6FE48409371D40CF	E28116C6EC70B24D
13	996B6F90BB0BE0D8	D9636F90BF0BE0D8	4D6158BE832B34FC	F651333BC396A35B
14	961401B7C0C285BE	D61C01B7C4C285BE	387B2C6DA9D884E4	B3F83B29BC333A76
15	D24813A652EC8A3A	924013A656EC8A3A	FDC464E8B0DC2E61	76A5D69577BF97E0
16	E819C116CAED2EF2	A811C116CEED2EF2	1ED0A8F4BAFD9699	C3156F192DBB9F48
17	994C009242E78990	D944009246E78990	2E06890E7F770EDC	E62FF8D4E143422D
18	55F4CB382DA92086	15FCCB3829A92086	CAB26966CEF38D21	BC29D990936825DC
19	E3F684B473FB98AD	A3FE84B477FB98AD	831189BB7D866707	1B2DF2AC86611EF3
20	E9D5EFC48271D4CC	A9DDEFC48671D4CC	B0A6EEA9E252D7B3	DD5A4377AAC3DAE9
21	260E2091246EEEBC	66062091206EEEBC	19AA4B626811B062	FF1ADD18BCD2A5E7
22	E2742C544D663EBE	A27C2C5449663EBE	6D380A2C8B1A298B	591980C6D73E2A88
23	A997ED9295C45416	E99FED9291C45416	1658C3835F26D09F	E2F650273EB7E399
24	3BA6171185BCF1FF	7BAE171181BCF1FF	F98DF52656135E52	902B80FE6C50C6AD
25	E81AAB5E5D31BDF4	A812AB5E5931BDF4	E7A2DFB01D52453A	779580D65B85AD9C
26	CA53F8437FF5238C	8A5BF8437BF5238C	D019E22595A65BF5	FEFAB18FD4F295BA
27	81F31FD9B6F9ECB8	C1FB1FD9B2F9ECB8	43F22DE3CF6C425D	13A8BF8A678B22A4
28	7F267FE7ED5FD7CB	3F2E7FE7E95FD7CB	AC5E08D09F893EE7	FABC5FE451D2BE58
29	82E02C7A1571B906	C2E82C7A1171B906	6774D9A60B609C0B	3CA8831B5C0FC0C6
30	C737632A8722175C	873F632A8322175C	32134EC45AFD003C	E387AC56B8A19E6E
31	0B5FC4B7C8FDD02B	4B57C4B7CCFDD02B	73665D69912C31E8	DD0909BDEBBE12D9
32	C94BD8B282AFBAB4	8943D8B286AFBAB4	28B4F79D54521A03	0FD9971D400AACC7
33	31D9D4171016A10C	71D1D4171416A10C	DB0CF104CBD15A28	6904EC628DDFEA0A
34	3F6625F43904FA27	7F6E25F43D04FA27	54B2D43154E3EDC0	F5E469961BBE4770
35	EFBAD278E0676F1A	AFB2D278E4676F1A	0C2E63283DDA88C0	FA33ABD3FE0D2B14
36	709C7542DB7C56FA	30947542DF7C56FA	FC2F1B70DB288C7E	35285045A7FD7921

37	C490ECF489613C04	8498ECF48D613C04	D13168B738911FED	E05EC3C17764DD47
38	A03C23492D52AA39	E03423492952AA39	950E6061F3D78181	686E1BABDE1B7B61
39	0830CE50F2C54AF0	4838CE50F6C54AF0	81F5CA14219433C9	B3C083B8F42A50FD
40	B790FC8B16367CB2	F798FC8B12367CB2	92BD2026299F763F	A71E50672567C552
41	9DAC0E058C3CDAD9	DDA40E05883CDAD9	BD07B824BCE8FDF7	7915F0A4F79C1681
42	CE16D37DD1BD572B	8E1ED37DD5BD572B	C514C0879CA336EF	2A4027D8934DABA4
43	670B5C0D8BD5F901	27035C0D8FD5F901	26A9CC7016F779EA	6FE919C922358DE6
44	477E766B91AE6706	0776766B95AE6706	91AFAE48EF542C8B	85606CE0B5D45227
45	5FA7E036B79369B6	1FAFE036B39369B6	9E6EAB38B0CBB265	49504F52B8663C8C
46	1F1372234020C25F	5F1B72234420C25F	BBD831F98FF9C2C9	FF18D5700AA7FE84
47	64C9693357FCD31F	24C1693353FCD31F	8DFC58184DC3A7D7	834DB9264170BC4E
48	4F2687EFAD220974	0F2E87EFA9220974	E1995EDAF85CF8BC	0708A0033834DE79
49	C390F947CDD02660	8398F947C9D02660	31A3652822967A03	D45DA5D02A1A5249
50	E3BD0F7E5DE9613C	A3B50F7E59E9613C	CA09CF62ECCE98DC	42FFF7D120F51C3B
51	A1075D6C4F265C57	E10F5D6C4B265C57	7EB635CD98C8A016	501A73A2698CC72C
52	533F8CD429020757	13378CD42D020757	5075204E9FEB9C4D	9B0A2513EF7D860B
53	D1D6CD99C996D0F8	91DECD99CD96D0F8	EB8D80B91D04E546	7CA2AC5C3C8B4E67
54	154BF979829EEA89	5543F979869EEA89	6BEDC80850A6FEF2	E8C52C9F94E27AB5
55	8C616CA26FEF2738	CC696CA26BEF2738	C585D3936AD9EFC6	6D180CF1E620A672
56	A6211BE5954DC75B	E6291BE5914DC75B	9400E7F8616AD9E1	079A677CFAAFED4E
57	555126A6A8D2A9B0	155926A6ACD2A9B0	53D28A57D6F25630	1A53C13D05CB8FF6
58	9A5C41747392C41D	DA5441747792C41D	B795CBC14AD60D25	6E78C70717C027C4
59	110E76203D817C9E	5106762039817C9E	3710A39BE1C3BB04	A3A38C577AE5929F
60	AD7301584F3561C8	ED7B01584B3561C8	85FCE99DCC69166F	D3F6CDB07C528CCB
61	D44C933FC8A82CA1	9444933FCCA82CA1	DA4DCDE9AAB885EC	30D09220101841C0
62	1079410385C8C22A	5071410381C8C22A	AE6ED005E5AF8AB6	3C1CB6F97C30FD94
63	6062494211EC4A5C	206A494215EC4A5C	F33C3A526E9F081F	80B44A105CA8F968
64	42FD5C53FE416279	02F55C53FA416279	0ED568E68832EE5B	0361A4AD262F87FE
65	2D72B86BE893856E	6D7AB86BEC93856E	BA16DDDBC17DA57A	0977E5A0186A5F6A
66	4305FB19B64BBD66	030DFB19B24BBD66	C860ADF03FA3D5B2	D954B6F2AD01FE07
67	42387F66CD744BF0	02307F66C9744BF0	52D85F4F33244455	EE662A6A5B66DCCA
68	4FE34774311C141C	0FEB4774351C141C	33B82C07F62F45DF	1AEC71DE95431179
69	D0C17AEF1096A4E0	90C97AEF1496A4E0	43CDD2EDF4E2BB6C	DB208516F10C0AE8
70	8B9E9B1AD0B6415A	CB969B1AD4B6415A	1F7DFC6BD1F062AE	C9D6E6A0153E5569
71	26CABE9A0F23E09E	66C2BE9A0B23E09E	7ED34573FC2464B9	A5549B9186E318CE
72	6C33DFFF3AB8A1B8	2C3BDFFF3EB8A1B8	33B2933BDBA06BEA	DAFC544AFDE290A9
73	34575CF15363C29A	745F5CF15763C29A	DCCCD539783D2147	C8D95E14F783D92E
74	C68A897FE6A2C3C0	8682897FE2A2C3C0	EDE8CE4AA3CC7903	0C882E1EEE43E774
75	D3CDD84E2FFB92E0	93C5D84E2BFB92E0	7CC2156782F715C4	62C7B89B2609F950
76	003A6331EB041CDC	40326331EF041CDC	61B017FEF3C8CAB5	74954389E65BBCAD
77	B7C8B1F41A899C6C	F7C0B1F41E899C6C	F2060068575ED47C	0AC7B3BB1198F7DB
78	12F1555F1F8BBA82	52F9555F1B8BBA82	F9F531BDF640B2EA	94D6AECF986D9D5D
79	53B6A2D9E2803DB1	13BEA2D9E6803DB1	47AACFA2FA616643	3972E3ED2CF464AC
80	E177C46252A09ED5	A17FC46256A09ED5	ECBD783F1A427FCF	1D8515C7AE523A6B
81	2BB445BF5633D4C3	6BBC45BF5233D4C3	379D22E494819F0E	E27460D2443BE71B
82	63DA751CB43AF0FA	23D2751CB03AF0FA	A9AC20C1DFD5DC56	B831A40386711381
Klucz 2: BF8980377505B539				
1	A118FA8135CE42DF	E110FA8131CE42DF	F120D8BB8BB651B0	D444570675E6F19D
2	216596D16433D020	616D96D16033D020	0FCAFACE2C8223C7	FA0DB10BEF9FC3D4
3	3B8C2366F539043C	7B842366F139043C	913B39A05EEF8F0D	78E60ACE900DAC78

4	10BB4E4DD47DD5B5	50B34E4DD07DD5B5	4D4A5F00F939A863	B55C6A27CB5383AA
5	B4511604B5364C8F	F4591604B1364C8F	48AE71551A28C526	CA4D9B7240EA22D1
6	B7ABA74280A66181	F7A3A74284A66181	0738AF834BA74861	D8DFDF523E0C5D0B
7	EE32132E5A5B1A9B	AE3A132E5E5B1A9B	B8608E12FA425244	AA5A5C09149D9020
8	6148E32E8AB2A380	2140E32E8EB2A380	BFEB259DAC71FC40	DAE3871F2245F4BB
9	A7AA887EB4E4740F	E7A2887EB0E4740F	314208EB02E10FDD	616DEDA535DCB3EC
10	141A0F3D96D94D64	54120F3D92D94D64	8419814F7991A64A	CBE6A757B5B255B6
11	E5CBEF6F43211FF8	A5C3EF6F47211FF8	32FB2513E4FAB45A	DFDE56E113827C3B
12	A4ACDD3A38EEE48F	E4A4DD3A3CEEE48F	F0C145F58D4587F1	38964337DC89ABE2
13	9FF2D632AD6F0796	DFFAD632A96F0796	620D6790889426FD	AF36277D9D4A19A6
14	D59C4C81F7B0F197	95944C81F3B0F197	0B4A1D3201F3263C	ECDFF44155A38D92
15	E8B89817B370180E	A8B09817B770180E	D5AEF687513080F2	CA1C6C5EFB016E03
16	47CC1F24410157F7	07C41F24450157F7	493AC21AFFF57C0C	96B56D971488653D
17	86323335E35FDC21	C63A3335E75FDC21	374E74C5C194756B	DFA7BF0598E43756
18	D7EDE50E174DEA55	97E5E50E134DEA55	0CD3D0B9128F38FB	B2D01075ACAEA143
19	B64E6693ACD79394	F6466693A8D79394	9B35F88E712A8D16	57AF76C410EE1F7E
20	3B54E02A9BCC7172	7B5CE02A9FCC7172	05C7C3C1565DCBC0	BDF344EF8AE03256
21	0D7C5DAAA3719102	4D745DAAA7719102	9ABDAC4D417DB76C	A3C50F783ABE9933
22	D39317076CD5D644	939B170768D5D644	3F8017B14FEBAA9C	A0B9421ED089BA76
23	2BDABA3A6ED3B743	6BD2BA3A6AD3B743	4A535A0EA2FBA230	316BD004B57F3BEF
24	A10F5AB30EC29CCF	E1075AB30AC29CCF	F1162723C13ECCF3	644757BAEA43F6CA
25	318E64E9B25EA44E	718664E9B65EA44E	F191A96491F17EE1	680B17DF4311C56E
26	0C19686016B4A4A0	4C11686012B4A4A0	6A52C341A41E652B	BA8CCD98DB2A228F
27	CE963888A3376091	8E9E3888A7376091	32DE45AF3146B758	F376EF8C754B6267
28	6BDB4303CECC6069	2BD34303CACC6069	94875DCE8ED87E91	33CA084A878DB8B5
29	FC04D54F1BDF79ED	BC0CD54F1FDF79ED	26419DFD807DED20	A6DF3057F0D5557B
30	059F1CA3E539D7D3	45971CA3E139D7D3	1AA7E67B3251351F	696E1D9C48BAE530
31	F1F1BC23EECAC6A1	B1F9BC23EACAC6A1	910F4258C48FC370	815379BD4FF1B2A9
32	37FE7A08B54EF71E	77F67A08B14EF71E	3133B71102FA1C43	AA3B3A8F11FA925A
33	DE711FA37F55E58F	9E791FA37B55E58F	2A93125AC6AC54F3	07FF67A12FB03B5C
34	828985CE550163D6	C28185CE510163D6	0809236D135A2EE3	6BD25114B94710F6
35	8810DF79A8AF44AF	C818DF79ACAF44AF	06D3FB2C169A5D47	1EA561BD2EFD5A9D
36	AADB05FB7151D486	EAD305FB7551D486	C9B53162B84A2E8B	A7D0260AED1007B7
37	10FCF11CC9E2503E	50F4F11CCDE2503E	5228915AA1B92FC4	4E3223ECC6D9BF69
38	46F770DEA88658B9	06FF70DEAC8658B9	6D93AC2F3990401D	EA6292D900FAC9A0
39	E4D45974F15CDE96	A4DC5974F55CDE96	065FB362884BD587	2FAACBA181122E2C
40	6B5E1953914A227F	2B561953954A227F	64749F5AC47E0000	EB5A0C21AE89A235
41	CE8EE9FF37D79C7C	8E86E9FF33D79C7C	5084F1B272C4F18F	E82D39334772B609
42	6A6370D727894C78	2A6B70D723894C78	41AA9E583813E320	C58A6F5AE5CA2D16
43	793E5AAFFF3483BE	39365AAFFB3483BE	D3FD0E0978E6CBA3	49CB2C58C2A3DCDB
44	69ACE16575FA6506	29A4E16571FA6506	61D2EC7A89CF62B5	292A1FC53E553ACF
45	ED37105D6562378A	AD3F105D6162378A	50A0C2C783A43102	52B1CEFED48467FA
46	CD9B23366B7B1296	8D9323366F7B1296	EF31392E706E6689	9F8E7E198753819B
47	5C3D5D7A9D169AC4	1C355D7A99169AC4	912A46170C144143	07519902F7F6E578
48	CF0E77FBD95ED72E	8F0677FBDD5ED72E	4C48F27C7B6CCB17	876F6903499981E6
49	8B54EC14D57CF96B	CB5CEC14D17CF96B	B87BC887CC5AD183	22039EA677F1F566
50	751CC77FD06B21CA	3514C77FD46B21CA	1A688CC9CB05F029	8D02C069B999DC1A
51	BCA0CD2493130BFE	FCA8CD2497130BFE	090A0B78348852B8	FDF094D0261A0F56
52	DDD5719B4CEB557F	9DDD719B48EB557F	D86AD210998585A0	1619C11DCAFB1B29
53	700332C1B52DF3A5	300B32C1B12DF3A5	92C68F4B46E62E92	E9A83D887AC85661

54	85B0BE87E80F7DA4	C5B8BE87EC0F7DA4	92257CCBED18C9D3	99EF06881127E71F
55	84AB2AB525789743	C4A32AB521789743	733B50F659FC69D8	B93930628B8BB264
56	5055D9439517FB28	105DD9439117FB28	81F8ACF587B33B7B	E6C18110C243465C
57	DC4C80E4C822D71A	9C4480E4CC22D71A	803B9A990ACAA0B4	076D25AB4BF194B5
58	969A831E7B39157F	D692831E7F39157F	DB5FBA13C3755EA0	AF7AF00003001C01
59	32B08B906AFF942C	72B88B906EFF942C	99BCE12137E804DA	03F8A48CC83F5742
60	8C6E976A2E927120	CC66976A2A927120	A3D479919AF8982E	D95C771B1D3C5565
Klucz 3: 0D8E27040A46F6C2				
1	623AA524CB149023	2232A524CF149023	82972AD214507E3B	F0F69A783C0969D9
2	F60170BBABDDE2AA	B60970BBAFDDE2AA	44DF1F8BDBC77EE8	767435C1B88B0376
3	6D51209510C02E2F	2D59209514C02E2F	1729EBEFFD98A5CC	34966436DDC616DB
4	F36A487C79CB76D1	B362487C7DCB76D1	D8B996DF63BE0952	7ABC91B5988A5C02
5	E2C0AD9194BF9007	A2C8AD9190BF9007	464A8AAC7633B843	877834C19F2248D1
6	DCAD4271B238A94B	9CA54271B638A94B	0FEC765EEE46FF2E	324AD29A87B15A14
7	4810A1229410D837	0818A1229010D837	D419CE93DF9DBFBC	6BB376A0862A40F5
8	B17A296BCCA5CFD9	F172296BC8A5CFD9	B9C3B034992E21E6	7B029B72D08BF8C4
9	880071CD6D6E4498	C80871CD696E4498	30C160ED35245775	49492C2C3E832D7C
10	64F4C58F95D3D4BD	24FCC58F91D3D4BD	711682D62DB262BF	38FA021520DAA446
11	4553CADA6AF3917A	055BCADA6EF3917A	D016E3E426BA1630	52E10B0ACF8BBC5B
12	388F44CAA8E70E8C	788744CAACE70E8C	B1C5860AD0160556	5220B50E211B997B
13	12AFA73C7C805A34	52A7A73C78805A34	B39FEBF666773139	F143D7D4F9101CE4
14	7415CC9C2ED3F7F9	341DCC9C2AD3F7F9	BE7586BC507DD9BC	6B4B597C32D9B0D8
15	B653FD521DE7DAEC	F65BFD5219E7DAEC	29F12114C16198E6	C1EF8F1F58BDED72
16	6022B677F2727025	202AB677F6727025	F5D30EF4FB690EBA	78F788898EB89154
17	F41E5DC0FEB61344	B4165DC0FAB61344	F004E371C75F19B9	E69CC96929F6956C
18	1481996DC35E909C	5489996DC75E909C	4D4A9E7EBD87CCBA	DF12FF7ADE1EE614
19	05E007A77EB69C52	45E807A77AB69C52	8CB1731A577AF2AD	10996C27CC7EC5BC
20	F83355B83C4F05FA	B83B55B8384F05FA	5136D1AA801E2BF4	79AEE683AFD73CC2
21	12AB08000E66CFA0	52A308000A66CFA0	19BF267229D87C1A	C7E7888F104F160B
22	5C09D0AF4F976FF3	1C01D0AF4B976FF3	499ECB229CCE34F1	6D69674112F61C4F
23	14E3E872B17A78FE	54EBE872B57A78FE	4BDA650557828E61	7B0CD07FF9F9F3D4
24	F84BA32DE17CA314	B843A32DE57CA314	C354CFC9504709B4	97470DCB2CE7B390
25	421B70156B791AC7	021370156F791AC7	B81AA75B25D70313	B59D83537D50647C
26	E6C5821A0709CE80	A6CD821A0309CE80	9D951AEC53CF4E08	312DB2C10E3F2B14
27	BF55EDFC3CD8A168	FF5DEDFC38D8A168	D306248737852366	A21564FC3E40B175
28	6E7B22066F9314E0	2E7322066B9314E0	39802ADB8E54286E	241D16A0C9E54E35
29	A25C56A3BA97B22B	E25456A3BE97B22B	40FE449F264C2823	0A4769AC07BD3475
30	7D2B2EBF78AFC5B1	3D232EBF7CAFC5B1	3D418C22EE7ABBF4	E1FAFE47675D348F
31	48592B2887017083	08512B2883017083	7A10D4B3FC176F1C	69FD5520115E5E59
32	A1E3D330FB81C298	E1EBD330FF81C298	D87597CB991CAF3B	86F459D9EFF74E3C
33	304F4D81069140DC	70474D81029140DC	C3615F6350F25BE7	7EEE763AC6DE2D0B
34	A6751554051AA72D	E67D1554011AA72D	0D8AB23D4A6245BE	1752FC66F33A1C61
35	8EE98D5D8D3692BF	CEE18D5D893692BF	9FD6A254853AB279	9678529F9E456535
36	DC65A36EBFEEBE78	9C6DA36EBBEEBE78	ADC2A24387FAE2FE	45CB8FCF66161A71
37	4C85143D1AC41EBE	0C8D143D1EC41EBE	3A491FCEBB8E4F1B	5ABE956997B703AB
38	A2CD1C248FE9E719	E2C51C248BE9E719	E3C5ED5822D45791	98323CE2A893C2DE
39	5A2C96D8884F61DC	1A2496D88C4F61DC	B1EA17C4E991B23C	30F11AE8211C7756
40	92E6CE53545883D6	D2EECE53505883D6	E5A47F820D23B97D	7F0B93F7867B99B9
41	5ACC009C3207D3C0	1AC4009C3607D3C0	D3DA8C0B938892FE	050F27780F07158E
42	88626BBCCADB62B0	C86A6BBCCEDB62B0	2AF676D40EDEE61D	7739DD10C4578C95

43	8A45532FCCDCA42F	CA4D532FC8DCA42F	5B7938DE45E8A010	BEF41A2809C2C9CB
44	A9F7C7F8C1C3E83F	E9FFC7F8C5C3E83F	7FB1DA6A327891F8	4DA19B4E1E8EE582
45	8D0D2341BB846E14	CD052341BF846E14	999DD82142C4533B	1AA70E4AA6A5CF31
46	8D3C4E33EB0515AC	CD344E33EF0515AC	B86EC12C718BBFFD	5E8E0C28FA4F1062
47	E4B076D49F90D6E8	A4B876D49B90D6E8	D5A3044178549527	3FB65B9001414873
48	510FB63A9CE72FF5	1107B63A98E72FF5	4165A51829F376D0	808B9BD361C89E35
49	F3E11C454EFC2F56	B3E91C454AFC2F56	E44D0E08D34E5AFD	465612397E918171
50	D950456F39B6B552	9958456F3DB6B552	FEF1F10F02FDD11B	B2B1E9444F9F843E
51	7E69EAD5F5DDE231	3E61EAD5F1DDE231	82AAC09FD012ABB0	E2ABAA384AE2A3F7
52	D9256C87E3F114B8	992D6C87E7F114B8	0991BB0C8D360453	10DE3DABC2148AEC
53	C1CEB3297894642F	81C6B3297C94642F	B7BEE881691F0649	EDF3C5E5821448E5
54	A50D555A68ED330A	E505555A6CED330A	839EE66C0F67EE88	1C798FFB02B07A88
55	4FA13492FC4F7172	0FA93492F84F7172	643093F6C681CB82	D7822230545EAD46
56	FFBA37DA5AB3D920	BFB237DA5EB3D920	E7DE4EFC946D558A	92A26611458DC327
57	A5D9B37C1FC89646	E5D1B37C1BC89646	9C68D5CE1DEED344	E869FB753C8A31F2
58	A35483908F4BADE4	E35C83908B4BADE4	C8E957C6081A4894	B8D2CBBD6481370A
59	9E7B1536C38A0B5C	DE731536C78A0B5C	39E5932CC50E97E6	A95B0B18DE94C904
60	F5FB3FBBD99BFB97	B5F33FBBDD9BFB97	6DEB3FEC32D121D8	F897DC2BC0E8D66F
61	A92371870165DF28	E92B71870565DF28	38F270071C53352B	2D93AAB0DEF8DFB5
62	CFE28ABA2CD8B861	8FEA8ABA28D8B861	91B4F07516C86871	4238E23E11CAD281
63	B4BCFB0AA061FA0C	F4B4FB0AA461FA0C	5676DD0D26800B79	AD4BFE553639FE17
64	73B501F4F88E3081	33BD01F4FC8E3081	41244C24413C29DD	848568315A0FE9D8
65	058ED4BA21528080	4586D4BA25528080	A82036C0AFD134D5	C7474A8425A0CCCC
66	52DD7932EBB27F5F	12D57932EFB27F5F	B0314B38E89C4B14	90CE2BFF42E12E0F
67	D98D4CEF6CC0F128	99854CEF68C0F128	74EB92A40826FE81	0C3D5EEE650356F5
68	EED52A92117C6C42	AEDD2A92157C6C42	C1B17B1C790EC611	8205DD82AB096821
69	EA348C0779EA9E31	AA3C8C077DEA9E31	26811C8EFF31BECB	BD3BC0ED8E69F1B6
70	6D23A984ABADE9F1	2D2BA984AFADE9F1	D14526555F39D64C	A685D12D82B9A696
71	9C8B9832C6997CFD	DC839832C2997CFD	37DF9C8466BB1344	203689CA7C5AC709
72	BBBEB950AF8EA3A6	FBB6B950AB8EA3A6	F565822E5E41845B	9B3332F995372FC5
73	31A57ACA29149107	71AD7ACA2D149107	BAC45248236487D4	268BE626092A3150
74	89DD529764B72A3F	C9D5529760B72A3F	FFE83A3731CA7A1B	90FADD8CACE53CE0
75	0427BB9E86259F66	442FBB9E82259F66	115CE9AC0E57678F	1B023376F0F5A2EA
76	79586B25353FAB0A	39506B25313FAB0A	86611AEA632801CD	82B9FD4C754F82FA
77	0A31D5F1E2C97E57	4A39D5F1E6C97E57	D8825B50985B8334	39EE33ED78D73202
Klucz 4: A49ECC6471E38D65				
1	F19E47C586415775	B19647C582415775	4E74CA1842AE209E	944688962C83E0F4
2	9C02E929A44DB365	DC0AE929A04DB365	94D7ABFD3931310D	6C4B2753A74249D4
3	215B9A7605BF7823	61539A7601BF7823	7777AC9E3BA34092	9D44407DD12A7503
4	B608AE1218DC8B8C	F600AE121CDC8B8C	1239BD2C136EE9C1	A2B7F31ED9499B43
5	43D07236DD8D03B2	03D87236D98D03B2	29FFEA790A78D7F6	0D48EBA6FE994C3F
6	CCB6371976AED2E8	8CBE371972AED2E8	9483A4C68B37E8D4	746BB6CC14683636
7	5C4B1D36EE8711FD	1C431D36EA8711FD	99C6616B1DD93FF8	FADC9A0D984B06AB
8	3602BFDA59CA1226	760ABFDA5DCA1226	67331DF31B99D11E	32286E8F41411929
9	34DC81421EB46D4D	74D481421AB46D4D	6577EEC0CFA7B346	C41651FF2AFD0315
10	B8C000E56A75AEE5	F8C800E56E75AEE5	7A3158A99FF9086E	2272B3CFF845F0C6
11	2573FF773381C681	657BFF773781C681	2D1687CF315D2319	29AD52F1872F6B04
12	AFDB7F6EF4F20E50	EFD37F6EF0F20E50	1A8B9F778C71220F	D8505B50B74FD23C
13	FCEC51A23AE7DB0A	BCE451A23EE7DB0A	594AB13C75033C40	6052CA2FA473A4A3
14	130141D2E83B0474	530941D2EC3B0474	E29775682D0E7E6F	B0DA8A27F0F2EB4B

15	3985AF2577B9802E	798DAF2573B9802E	6EA1CF3D23CD2386	6EA96014115E1FCF
16	A9A66B933C47825D	E9AE6B933847825D	46ACA2520414F18B	433582E3D38C818A
17	9F6E86B83F87D669	DF6686B83B87D669	695CFA3D0A95E56E	4AED9E5C3A061D61
18	C5EF6B7B53765E0A	85E76B7B57765E0A	A0CD51B9828B2FA0	E952769AEC9D252A
19	D41E7CD3E127326C	94167CD3E527326C	30D799DD3D7AF4C9	23674629F866CF45
20	DFAF0F63A9E626B4	9FA70F63ADE626B4	45BBF5D59AC0DAF3	86B609305D71FCAB
21	2B0DAA1FA92E9DC7	6B05AA1FAD2E9DC7	F5BEACBBA5150FCC	A7ABCBEB7B2AAB68
22	D65D938C003057B8	9655938C043057B8	856C31CC980DC80B	719DE88953B859A3
23	840CEA8E2BC6789A	C404EA8E2FC6789A	15136EE28C45887F	C32F000C43478CB0
24	EE953D39E8126824	AE9D3D39EC126824	B94E67E8A8BBC33A	5DE8BC4EA248F0F9
25	6927D55B07E5868A	292FD55B03E5868A	C4F8AEB895C97487	67CE1B79B307184E
26	38BD6B78E24C03A8	78B56B78E64C03A8	340C6BDA2F296989	B1F9D1AF8C40022D
27	D04091E22117D899	904891E22517D899	21B7B674DF2DBAAF	7A8F1930C65389D0
28	521FBB3DF047FAF1	1217BB3DF447FAF1	228E9CC091996460	BBA33A461BA310C3
29	10BE9774FF520F4F	50B69774FB520F4F	ED7BA924CE0487D0	DDC22EFE7036CC2D
30	102F8D401ACD1B36	50278D401ECD1B36	C2CDB7AD9C70C32C	0D7256EF1D61E3A4
31	B1BECBBA1C003CB6	F1B6CBBA18003CB6	1F8A7C9E470DB15C	A0CA63CACB39E84B
32	D72B577783AD7FBA	9723577787AD7FBA	5F1EAC517C1597DD	5941F575B5E77149
33	51B9B68A70EC3623	11B1B68A74EC3623	85862A6700CF85E8	7521E4991B9D6DDB
34	85426277F7348B31	C54A6277F3348B31	0B3D005CE6B50006	F0E10B3E9FE68BAA
35	184D1FDB376AEBF3	58451FDB336AEBF3	58437DF3337F78F3	F18B8A96C98CF120
36	36740E4F31F1EA3B	767C0E4F35F1EA3B	A254646E046C6B48	A8A6D66AAD832222
37	ACF1FD3794B57D3E	ECF9FD3790B57D3E	7BD6E44B80D6D781	F14C30420F433107
38	1E5C95C396474769	5E5495C392474769	AEAEB7C5BCC5BF1E	AA32BA2C531E6DC6
39	5BAA56640C799B32	1BA2566408799B32	86530233A62B1235	DAC4703CCC5161C8
40	9561F2C27A778978	D569F2C27E778978	C76778FC3F95BDED	6B1DC521592E78DE
41	EFCC5F9D1232425B	AFC45F9D1632425B	52DA0E384767128B	2A3A78A70A29F75D
42	FD2C2CC527F9F884	BD242CC523F9F884	F9C1E36400E42D4A	70F8A6040D1C99A1
43	2239454399B49748	623145439DB49748	F4AB2F7287B93AC2	88D69B541A0964DB
44	FA3B50B86293548C	BA3350B86693548C	16693C43DA2E0404	99AFA8A38517F926
45	0E72340194D72BCD	4E7A340190D72BCD	EF8CDE72D5F7DEA9	FB17FD339F331A00
46	993643B6118CAE3D	D93E43B6158CAE3D	77BD79217B4AA65F	6D469245A16BD3B1
47	80C977FE9356FB44	C0C177FE9756FB44	202D3472404E58B4	C99005C448D467AF
48	4E0F623F0FFA6DF3	0E07623F0BFA6DF3	92E9F1D4BD13A4D8	5C930A0781B068B6
49	41D0FC1045377037	01D8FC1041377037	E88B71832F641B09	12A3A1C431D56DC9
50	3BF819BF3DF884D0	7BF019BF39F884D0	D8B97A7C6996DB0A	EC6B411DF8D5A733
51	6000B69F502A6014	2008B69F542A6014	12B46903ADAF2C2C	C9C87329A5B5BC91
52	35DDE909FE825BE6	75D5E909FA825BE6	89F368EDEF4A28D2	A2A00BD9CDCA86AA
53	52F37E79B7F77577	12FB7E79B3F77577	E9D052BA24588A69	31B303AEF4511A6B
54	EE0AB4891A562D55	AE02B4891E562D55	41409A0919480545	420602E8BC039A5D
55	46C330EFAA242985	06CB30EFAE242985	0558120A83F9DBB2	63908C47983A52C6
56	6B8FFE6CB02499A0	2B87FE6CB42499A0	20ECEBB82AC24AC4	05AEA4037B6C389A
57	54C0CE3AC6DEA079	14C8CE3AC2DEA079	4C4901B78941E290	407C45344EB6ECB2
58	01FED7988E39081B	41F6D7988A39081B	FB69E21E99C5B221	326A9BD2C6C10D57
59	D18EE91ACCD7E2C8	9186E91AC8D7E2C8	E39B6D547DAD78A4	2D5D9C8D6728218D
60	124160BDB96A01D7	524960BDBD6A01D7	0D22E445DEA4072B	08855100E2E69CD9
61	E0A0CA3942ED4560	A0A8CA3946ED4560	80A107D9C1BEF3A7	56C4BE993D7C5A07
62	83DFC9922BE07F98	C3D7C9922FE07F98	94A5E9BDEA65438B	85700C7E94AAB505
63	2D1C98BF8E181AE0	6D1498BF8A181AE0	987DD358F9317C23	02BC36292F89B7E3
64	3DBF0899C9F30C3C	7DB70899CDF30C3C	6C8D964A270620B9	C524BE355709D47A

65	51F9CA4F7D7D8007	11F1CA4F797D8007	233866EB35EDADF0	A1F33B892737B84C
66	CDD94989480098FB	8DD149894C0098FB	E0D720CD03506F96	B6640AEBAC86DCF1
67	9763FA8B8553945E	D76BFA8B8153945E	ECF8691FE59DA679	BD335133ACCCDB84
68	64622CB0B876F0F4	246A2CB0BC76F0F4	9E6C9C5AAFDB69D3	61FC638258FE1888
69	E4923972D0485905	A49A3972D4485905	31EBA09CAC6CB828	7DED03DF39B3F131
70	D1C187DD502F9753	91C987DD542F9753	04ABDADAA80D1F3C	8168FBAB914B1C70
71	69044970A99BEDE7	290C4970AD9BEDE7	1BB61CFCD1175F92	C3249416D9001AA2
72	BC1FBD04D231AC5D	FC17BD04D631AC5D	6BD78BC40C4D7B15	BD907DB8D2DB85BE
73	16595C91F0101C1A	56515C91F4101C1A	DE1CC80516E0BDFF	8D6C3D740721BC1D
74	195CB5D9AC0958CA	5954B5D9A80958CA	FA4D216653886736	B09C361AA0A7B299
75	A2C63926B2D091C8	E2CE3926B6D091C8	E812FA5780386138	1DDED3CAC3B35FB3
76	CD150B7E344EB987	8D1D0B7E304EB987	F3FE89C13D6AF1C4	05CF13FD4C19CE5A
77	7838B6BCE9A18F8F	3830B6BCEDA18F8F	883B38A91491226C	226306D0E2CFC8EA
78	2AA62C7CCD829726	6AAE2C7CC9829726	27F3B0E0B69A6BB5	377B6925D1FBB21E
79	EE57BF1FD3B84DE1	AE5FBF1FD7B84DE1	32BAE0B827A556D3	D7E8B8BDA2C98C01
80	70BF117A818BB44E	30B7117A858BB44E	50D871AB9F21C950	C8DB18EFAB39313A
81	1A1C09B541EE0EFC	5A1409B545EE0EFC	E6C93E8696CCB635	3B00AC2E844B90DA
Klucz 5: EB2275E5FFB1F2C4				
1	B3729F54AD43CA78	F37A9F54A943CA78	3FE77A734701C59A	E668C12F1DF31C72
2	96981A2021BB24FE	D6901A2025BB24FE	71FB65269840759E	A28E48E841624E57
3	47EED8F016F83925	07E6D8F012F83925	2AD4D0C70388B31E	DF44381107A3D915
4	D5A88BBE086FB375	95A08BBE0C6FB375	675F7B31E99F154A	0427E5DA46E65FA5
5	AD0123121E064BE5	ED0923121A064BE5	330C5464DDDFD677	69E6F03CADC7B018
6	E7F3405D33F8DE70	A7FB405D37F8DE70	DC4CBC40C34A9606	0BB553BEB5603F39
7	35EC8EB74B7E9DA2	75E48EB74F7E9DA2	A49F1DAAE7DD8AE3	E2ED58F08B3F7C18
8	507A92270B47FAB8	107292270F47FAB8	DB0E27C26357AFD8	F1E44160CD13639C
9	4146E76B87110DD2	014EE76B83110DD2	A5E4D8A8F2218877	971788F3E395F5C5
10	53050E080BD45ECA	130D0E080FD45ECA	3067DA8653379A20	3E0087E53CDAE976
11	1251675EAB110163	5259675EAF110163	7193F2908983FF36	B36DF8FAD67EE304
12	E22CA6761C53916A	A224A6761853916A	23BDAFE8BCBAD5BF	71C335CA53FF0BC5
13	566909B164E7B291	166109B160E7B291	85106210C0BE10CC	5EF99AF905D24BE5
14	45EFA2FA6528C1CB	05E7A2FA6128C1CB	AA8578D070BB10A2	4E20710495A73654
15	D2F68EEBF4C618F8	92FE8EEBF0C618F8	C1D195C03A410D20	8AD24E779CD04789
16	266DF7A6C1472BB3	6665F7A6C5472BB3	BE707D2F3F722774	0CB90E0D62CA2536
17	E27E9BEE4E8BC62C	A2769BEE4A8BC62C	B19D7D4D94A84C4A	615F25901BAC9882
18	D094C7D7D81D2D19	909CC7D7DC1D2D19	6DFF0BA950814437	D70B8D58AA6E07F6
19	F79127664F11A1B7	B79927664B11A1B7	6307A6C1B44EEB48	36434CB7E15EC767
20	633E304D7A08FB07	2336304D7E08FB07	C9A661A8DB553FCF	E0240B2BA95AD2EC
21	567EA96E9DF79728	1676A96E99F79728	537ECD487551E704	4425801DDB540FF5
22	874605562B2508F1	C74E05562F2508F1	B1F41F9235A2B1D2	D29B5A513C527AC9
23	910E011C790E770D	D106011C7D0E770D	BDCDF5A88CFCC4C9	38FCF0164992D32E
24	7EA77B49519BA5EB	3EAF7B49559BA5EB	0AAF41E19E0987FD	826B0C7D21821678
25	A606784FE44CB8EE	E60E784FE04CB8EE	8BC80D04128DF804	112EB2BD7AB2B0B0
26	F63CC63FAAB09993	B634C63FAEB09993	BCDD79EEE85E9685	63AAAF1D3DB43D3C
27	7776C1593C05C70B	377EC1593805C70B	AA01056C1B788377	AD1824FC1317A7AA
28	4855CE89B47BA3AA	085DCE89B07BA3AA	2E14E6B307ADE4A9	4A508481FD04947A
29	526309AEC9C6BC06	126B09AECDC6BC06	50F9F98EB7E9CA8D	D147AEE97479DF0E
30	8DDD6E6913EC0B97	CDD56E6917EC0B97	7C16E3E90F6FF5B0	9B9552D6B7DA1F7C
31	F2EC15ABAA34BE29	B2E415ABAE34BE29	1A2E6437256DE555	AFA9939AE9DA36EB
32	A61095B1948C242A	E61895B1908C242A	CE3DDE1FEF150D26	23157AB2D6632671

33	44ACA90B64F3B0AA	04A4A90B60F3B0AA	F5FE91EE1122B9E0	394E473BB6B7BF91
34	DB6245B84EE82C34	9B6A45B84AE82C34	759351E581E3ECC2	EBE59744BA794187
35	8A1CDEA2A05576FF	CA14DEA2A45576FF	3F556A7C77746215	7DF57E0DB24AD73E
36	4A3311C94ECF278B	0A3B11C94ACF278B	0BC1FBA5271250D8	E4B916B43DF4FD03
37	CC90468C5C15FF03	8C98468C5815FF03	0DA806D906A4454A	550736EBFFC5C93C
38	04A6208D17D506A5	44AE208D13D506A5	F6C8CFE379DD1489	A672C9C9F72C55D6
39	E91399FC406D21D0	A91B99FC446D21D0	FFFA36A0F14EE4D3	563A474E52EEEDDE
40	F45B590D60E52293	B453590D64E52293	41B90E73196276FF	4E61731714ADF6D7
41	DE9439364D129AB0	9E9C393649129AB0	338F4543DA2316F4	67676607C3CA5936
42	103CE4691152DE85	5034E4691552DE85	9C9FFDCCA4DC4E64	02D53F4448965947
43	80DC7FCF82592532	C0D47FCF86592532	41C50B5027A32391	1F001A7D2627EF1D
44	CFAC0BD3F4FF76BF	8FA40BD3F0FF76BF	4722D8CEBA9B272E	F5EB9A285EBDE86E
45	8229F80469D289FD	C221F8046DD289FD	4F0427CCF9CBB24B	7C3A5E29339AF91B
46	68A92A52CE3F5E08	28A12A52CA3F5E08	1C1CC899AAC01BAE	0BC0940640080A9B
47	F4F8E6A72A645FA4	B4F0E6A72E645FA4	658A36CB9F183E88	AF12E288AE1FCACD
48	83EBE8B9D649CFAB	C3E3E8B9D249CFAB	D1314BA9A4C8FBB7	006CC6434C805C3F
49	FBF30850AF15A227	BBFB0850AB15A227	4D19992F0A279A6F	85D3FBA7E8B42D97
50	FAAC6C464D93CDC5	BAA46C464993CDC5	A59E536457009889	19105DDBB081827F
51	332257282937EB39	732A57282D37EB39	CC600517C62196CD	72C5196348A64EB1
52	195F860AA4F3D492	5957860AA0F3D492	B6EEA8AA4A781F67	2132D1C9FA3AFD41
53	B358EB38819D9C78	F350EB38859D9C78	0C6864DADF59CC4D	1984270078FD48EB
54	79416EBA4D4E5424	39496EBA494E5424	5DF02AB8A46D8609	4871B368690E1763
55	DEF257DF4977818F	9EFA57DF4D77818F	82F390F666395843	CF3EAA50B731B0AC
56	24934949978877D3	649B4949938877D3	0E372E1F74744130	5B0A2E4E7D7F42A5
57	B4BD48EC52FD90E3	F4B548EC56FD90E3	B727DBBF70AC2796	E38F00FB1102EE62
58	461D3B5979739D07	06153B597D739D07	7DAEA871758893C4	97FCECBDDE372DC1
59	90111FCAE968E3DE	D0191FCAED68E3DE	B07D231D47296C80	D8944A92FB3EEB33
60	D52F67519303D1C6	952767519703D1C6	78B8A74CDC6C25EE	ED6CF7B83F600069
61	B250AF09FF9AA36E	F258AF09FB9AA36E	8FF74DE9D6265EB6	A8E87ECC9056E59A
62	7B0BD21628C9642F	3B03D2162CC9642F	3C53E56E99325C40	9DD051859E7655B3
63	B1F1AB891E51E9D2	F1F9AB891A51E9D2	1A7BE6BFCF8039F1	629F9E6DF8E6522B
64	4A6C7266E898BC38	0A647266EC98BC38	A48E984D30398B99	91B4A7A0727EB342
65	A105B7E004A2D5C7	E10DB7E000A2D5C7	E92411D14CDB3B35	236580AE60575728
66	1E6B26BF287B7C6D	5E6326BF2C7B7C6D	AA04A85E9D06CE02	C9EE9A75DC391CD8
67	4672F86AB05A5186	067AF86AB45A5186	EF6732904DD6D248	F509D369E63D8297
68	0C4F1F84B8074751	4C471F84BC074751	3CB8A3824D51BCAC	59878F8AC62E9C5D
69	441A8F409D956B65	04128F4099956B65	BC1C2136FE1D8B58	E3C8EBB899F6A413
70	5228279505CB35B7	1220279501CB35B7	DA09AC80D8BE5823	E571F1594919014B
71	0D305A9F0AEA1586	4D385A9F0EEA1586	D8B9AB8C71F56C1F	53102982DF52B212
72	125986FA93879E3B	525186FA97879E3B	92D148E070C90124	9809AA80F069A7A3
73	C6CC8561B64916B6	86C48561B24916B6	D64A90B54FB8B953	E4E786E8954A3253
74	38030EACED3C2B72	780B0EACE93C2B72	84CA4525A9F642EC	95A734B1D5AC4366
75	9964F3752858C633	D96CF3752C58C633	42DBE7B67B3C6BE6	D700D8FA9B024593
76	C2BF172F1A089107	82B7172F1E089107	C48D38A621A1E657	FC2578D401A3AB93
77	D8BAA4208F7A5AE6	98B2A4208B7A5AE6	0309E67ED34F650D	C35958C8CCB85F07
78	C9EF94FE5873A0C2	89E794FE5C73A0C2	E56ADF4F4B888478	5D4058C63419024B
79	5CA056C5E4EC62EC	1CA856C5E0EC62EC	5C13AE2D45938FBB	87A5B4FE487F6827

Tabela B.1: Lista par tekstów jawnych i szyfrogramów wykorzystana podczas kryptoanalizy różnicowej Ω_1

Spis symboli i skrótów

Symbol	Opis
$\oplus$	(ang. *Exclusie-or*) - operacja XOR, znana również jako różnica symetryczna lub alternatywa wykluczająca
$\lll$	(ang. *Cyclic Shifting to the Left*) - cykliczne przesunięcie w lewo
$\|\|$	(ang. *Concatenation*) - konkatenacja
1-PX	(ang. *1-Point Crossover*) - krzyżowanie jednopunktowe - wariant krzyżowania wymieniającego
2-PX	(ang. *2-Point Crossover*) - krzyżowanie dwupunktowe - wariant krzyżowania wymieniającego
3DES	(ang. *Triple Data Encryption Standard*) - algorytm szyfrowania symetrycznego polegający na trzykrotnym zaszyfrowaniu tesktu jawnego algorytmem *DES*
A5/1	jeden z najbardziej znanych algorytmów szyfrowania strumieniowego
A5/2	następca strumieniowego schematu szyfrowania *A5/1*
ACO	(ang. *Ant Colony Optimization*) - algorytm optymalizacji mrowiskowej, metoda inspirowana zachowaniem mrówek
AES	(ang. *Advanced Encryption Standard*) - zaawansowany standard szyfrowania danych, światowy lider w dziedzinie symetrycznych szyfrów blokowych
AS	(ang. *Ant System*) - system mrowiskowy, metaheurystyka naśladująca zachowanie mrówek
AVX	(ang. *Average Crossover*) - krzyżowanie uśredniające, jeden z najpopularniejszych wariantów krzyżowania
ARX	(ang. *Arithmetical Crossover*) - krzyżowanie arytmetyczne, wariant krzyżowania zmiennoprzecinkowego
BLAKE2	algorytm funkcji skrótu. Jeden z finalistów konkursu na *SHA-3*

Symbol	Opis
C	(ang. *Ciphertext*) - szyfrogram, kryptogram, wynik transformacji tekstu jawnego za pomocą algorytmu szyfrującego
CX	(ang. *Cycle Crossover*) - krzyżowanie cykliczne, wariant krzyżowania dla list uporządkowanych
DE	(ang. *Differential Evolution*) - algorytm ewolucji różnicowej
DEA	(ang. *Data Encryption Algorithm*) - właściwa nazwa algorytmu *DES* - standardu szyfrowania danych
DES	(ang. *Data Encryption Standard*) - standard szyfrowania danych, jeden z najpopularniejszych symetrycznych szyfrów blokowych
EA	(ang. *Evolutionary Algorithms*) - algorytmy ewolucyjne, zbiór rozwiązań inspirowanych naturą
EP	(ang. *Evolutionary Programming*) - programowanie ewolucyjne, metoda automatycznego generowania programów komputerowych w opraciu o podzbiór algorytmów ewolucyjnych
ES	(ang. *Evolutionary Strategies*) - strategie ewolucyjne, metoda opierająca się na wykorzystaniu mutacji i reprodukcji
FEAL	(ang. *Fast Data Encipherment Algorithm*) - symetryczny szyfr blokowy, mający bardzo duży wpływ na rozwój kryptoanalizy
F_f	(ang. *Fitness Function*) - funkcja przystosowania
GA	(ang. *Genetic Algorithms*) - algorytmy genetyczne, specjalny przypadek algorytmu ewolucyjnego naśladujący zjawisko ewolucji naturalnej
GP	(ang. *Genetic Programming*) - programowanie genetyczne, metoda automatycznego generowania programów komputerowych w oparciu o drzewa składniowe
GSM	(ang. *Global System for Mobile Communications*) - standard telefonii komórkowej
HC	(ang. *Hill Climbing*) - algorytm wspinaczki, prosta metoda przeszukiwania lokalnego
HX	(ang. *Heuristic Crossover*) - krzyżowanie heurystyczne, wariant krzyżowania zmiennoprzecinkowego
IDEA	(ang. *International Data Encryption Algorithm*) - jeden z algorytmów symetrycznego szyfrowania blokowego
k_B	(ang. *Boltzmann Constant*) - stała Boltzmana, $k_B \approx 1,38$
K_E	(ang. *Encryption Key*) - klucz szyfrujący
K_D	(ang. *Decryption Key*) - klucz deszyfrujący

Symbol	Opis
MA	(ang. *Memetic Algorithms*) - algorytmy memetyczne, połączenie algorytmu genetycznego wraz z metodami lokalnego przeszukiwania
MD6	(ang. *Message-Digest Algorithm*) - jedna z najpopularniejszych jednokierunkowych funkcji skrótu
$\mathcal{N}(K_i)$	(ang. *Neighborhood*) - zbiór rozwiązań sąsiednich dla osobnika K_i
O(t)	(ang. *Offspring population*) - populacja potomna
OX	(ang. *Ordered Crossover*) - krzyżowanie z porządkowaniem, wariant krzyżowania dla list uporządkowanych
P	(ang. *Plaintext*) - tekst jawny, czytelna dla każdego wiadomość
P(t)	(ang. *Population*) - populacja bazowa
p_c	(ang. *Crossover Probability*) - prawdopodobieństwo krzyżowania
p_m	(ang. *Mutation Probability*) - prawdopodobieństwo mutacji
PMX	(ang. *Partially Matched Crossover*) - krzyżowanie z częściowym odwzorowaniem, wariant krzyżowania dla list uporządkowanych
RC4	(ang. *Rivest Cipher 4*) - jeden z najpopularniejszych szyfrów strumieniowych. Wykorzystywany w m.in. w protokołach *SSL*
RC5	(ang. *Rivest Cipher 5*) - symetryczny szyfr blokowy. Szyfr ten nie ma nic wspólnego z szyfrem *RC4* poza nazwą i autorem
ROT13	(ang. *ROTate 13 encoding*) - nazwa jednego z najprostszych szyfrów podstawieniowych kryptografii klasycznej
RSA	(ang. *Rivest-Shamir-Adleman*) - najpopularniejszy algorytm szyfrowania asymetrycznego
SA	(ang. *Simulated Annealing*) - symulowane wyżarzania, metoda przeszukiwania lokalnego wzorowana na wybranych procesach fizycznych
SAHC	(ang. *Steepest Ascent Hill Climbing*) - algorytm wspinaczki z metodą najszybszego wzrostu
SEAL	(ang. *Software-Optimized Encryption Algorithm*) - super wydajny programowo szyfr strumieniowy. Algorytm zoptymalizowano pod kątem wykorzystania na procesorach 32-bitowych
SHA-3	(ang. *Secure Hash Algorithm 3*) - właściwie algorytm Keccak. Światowy lider w dziedzinie jednokierunkowych funkcji skrótu
SPN	(ang. *Substitution–Permutation Network*) - sieć podstawień i permutacji. Popularna technika transformacji w wielu współczesnych szyfrach blokowych
SSH	(ang. *Secure Shell*) - standard protokołów komunikacyjnych używanych w sieciach *TCP/IP*

Symbol	Opis
SSL	(ang. *Secure Socket Layer*) - protokół zapewniający poufność oraz integralność transmisji danych
T(t)	(ang. *Temporary population*) - populacja tymczasowa
TCP/IP	(ang. *Transmission Control Protocol/Internet Protocol*) - teoretyczny model warstwowej struktury protokołów komunikacyjnych
TEA	(ang. *Tiny Encryption Algorithm*) - bardzo prosty, a zarazem wydajny symetryczny szyfr blokowy
TS	(ang. *Tabu Search*) - algorytm przeszukiwania z listą Tabu
UX	(ang. *Uniform Crossover*) - krzyżowanie równomierne, wariant krzyżowania wymieniającego
VMPC	(ang. *Variably Modified Permutation Composition*) - funkcja skrótu, a zarazem algorytm szyfrowania strumieniowego polskiego pochodzenia. Stanowi modyfikację szyfru *RC4*
WAKE	(ang. *Word Auto Key Encryption*) - szybki algorytm szyfrowania strumieniowego. Pozwala na szyfrowanie całych słów - poprzednie słowo wykorzystywane jest do stworzenia następnego słowa klucza
XTEA	(ang. *eXtended TEA*) - rozszerzona wersja algorytmu szyfrującego *TEA*, eliminująca jego największe niedoskonałości

Słownik pojęć

Pojęcie	Opis
Atak siłowy	(ang. *Brute Force Attack*) - najprostsza metoda ataku kryptoanalitycznego. Polega na sprawdzeniu wszystkich możliwych podkluczy do momentu, aż uda się znaleźć właściwy klucz deszyfrujący. Atak ten teoretycznie gwarantuje odgadnięcie poprawnego klucza, aczkolwiek praktycznie może on trwać bardzo długo, nawet kilkadziesiąt lat.
Chromosom	(ang. *Chromosome*) - miejsce przechowywania genotypu osobnika. Zawiera kompletny opis cech rozwiązania zakodowany w określony sposób, np. za pomocą znaków binarnych.
Deszyfrowanie	(ang. *Decryption*) - proces zamiany szyfrogramu na tekst jawny przy pomocy poprawnego klucza i algorytmu deszyfrującego.
Efekt Baldwina	(ang. *Baldwin Effect*) - technika zapamiętywania osobników podczas procesu eksploatacji np. w algorytmach memetycznych. Metoda ta polega na przypisaniu osobnikowi samej wartości funkcji przystosowania, wyznaczonej wskutek procesu przeszukiwania lokalnego. Wszystkie wyuczone cechy pozostają zapomniane - nie mają one wpływu na postać osobnika, zapamiętywana jest tylko wartość funkcji przystosowania.

Pojęcie	Opis
Efekt lawinowy	(ang. *Avalanche Effect*) - zjawisko polegające na tym, że minimalna modyfikacja w danych wejściowych (np. w tekście jawnym lub kluczu) pociąga za sobą znaczne zmiany w danych wyjściowych (np. w szyfrogramie). Niezbędna cecha wszystkich współczesnych algorytmów szyfrujących.
Ewolucja Lamarkowska	(ang. *Lamarckian Evolution*) - technika zapamiętywania osobników podczas procesu eksploatacji np. w algorytmach memetycznych. Zapamiętywany jest nie tylko cały osobnik, wraz z nowo wyuczonymi cechami, ale również wartości funkcji przystosowania.
Funkcja haszująca	(ang. *Hash Function*) - patrz jednokierunkowa funkcja skrótu.
Funkcja przystosowania	(ang. *Fitness Function*) - funkcja określająca poziom radzenia sobie w środowisku. Za pomocą funkcji przystosowania można ocenić stopień przydatności osobnika w populacji.
Integralność	(ang. *Data Integrity*) - własność danych, zabezpieczająca przed nieautoryzowanym zmodyfikowaniem danych z wykorzystaniem metod, mających na celu ukrycie faktu dokonania zmiany.
Jednokierunkowa funkcja skrótu	(ang. *Hash Function*) - zwana również funkcją haszującą, funkcja przekształcająca tekst jawny w nieczytelny, nieodwracalny oraz o stałej długości ciąg znaków. Wygenerowany ciąg znaków nazywany jest haszem.
Klucz deszyfrujący	(ang. *Decryption Key*) - klucz wykorzystywany w procesie deszyfrowania, wykorzystywany do transformacji szyfrogramu na tekst jawny. Niezbędny element każdego algorytmu szyfrującego. W przypadku szyfrów symetrycznych klucz deszyfrujący jest taki sam jak klucz szyfrujący.
Klucz szyfrujący	(ang. *Encryption Key*) - klucz wykorzystywany w procesie szyfrowania, służy do zamiany tekstu jawnego na szyfrogram. Niezbędny element każdego algorytmu szyfrującego. W przypadku szyfrów symetrycznych klucz szyfrujący jest takiej samej postaci jak klucz deszyfrujący.

Pojęcie	Opis
Kryptoanaliza	(ang. *Cryptanalysis*) - proces mający na celu odszyfrowanie najczęściej przechwyconego szyfrogramu bez znajomości właściwego klucza deszyfrującego.
Kryptoanaliza liniowa	(ang. *Linear Cryptanalysis*) - metoda ataku polegająca na sprowadzeniu wszystkich procesów szyfrujących do postaci funkcji liniowej. Większość operacji opiera się zazwyczaj na wykorzystaniu operatorów różnicy symetrycznej i koniunkcji, dzięki czemu, za pomocą aproksymacji liniowej, możliwe jest zbudowanie rozwiązywalnego układu równań liniowych.
Kryptoanaliza różnicowa	(ang. *Differential Cryptanalysis*) - metoda ataku z wybranym tekstem jawnym. Polega na wygenerowaniu skończonego zbioru par tekstów jawnych i odpowiadających nim szyfrogramów różniących się w pewien charakterystyczny sposób. Wykorzystując różnicę pomiędzy tekstami w kolejnych rundach można przypisać pewne prawdopodobieństwa, sugerujące poprawność niektórych podkluczy.
Kryptografia	(ang. *Cryptography*) - dziedzina bezpieczeństwa informacji zajmująca się tworzeniem algorytmów szyfrujących, protokołów oraz ogólnopojętym tematem szyfrowania.
Kryptologia	(ang. *Cryptology*) - dyscyplina nauki obejmująca tematy związane zarówno z dziedziną kryptografii jak i z dziedziną kryptoanalizy.
Krzyżowanie	(ang. *Crossover*) - jeden z podstawowych operatorów algorytmu memetycznego. Proces krzyżowanie odpowiada za wymianę materiału genetycznego pomiędzy osobnikami, w efekcie czego otrzymuje się nowych osobników potomnych.
Mutacja	(ang. *Mutation*) - jeden z podstawowych operatorów genetycznych genetycznych algorytmu memetycznego. Zadaniem tego operatora jest wprowadzenie pewnego elementu pseudolosowości wewnątrz osobnika potomnego. Najczęściej jest to jakaś niewielka perturbacja umożliwiająca wyskoczenie z aktualnie eksploatowanej przestrzeni rozwiązań.

Pojęcie	Opis
Metaheurystyka	(ang. *Metaheuristic*) - algorytm wykorzystywany w przypadku, gdy klasyczne metody nie są w stanie zwrócić wyniku w rozsądnym czasie. Wykorzystywane są do iteracyjnej optymalizacji problemu. Wiele z nich inspirowanych jest naturalnymi procesami pochodzącymi ze świata natury.
Nacisk Selektywny	(ang. *Selection Pressure*) - zdolność algorytmu np. memtycznego lub genetycznego do poprawiania średniej wartości funkcji przystosowania wewnątrz populacji.
Populacja bazowa	(ang. *Base Population*) - zbiór osobników aktualnie funkcjonujących w środowisku - w danej iteracji algorytmu.
Populacja początkowa	(ang. *Initial Population*) - zbiór osobników wygenerowanych w pierwszej iteracji algorytmu ewolucyjnego. Zbiór najczęściej generowany jest losowo.
Populacja potomna	(ang. *Offspring Population*) - zbiór osobników potomnych powstałych wskutek realizacji podstawowych operatorów genetycznych takich jak np. krzyżowanie lub mutacja.
Populacja tymczasowa	(ang. *Temporary Population*) - zespół osobników, uzyskanych na podstawie populacji bazowej, za pomocą procesu selekcji. Zdarza się, że populacja tymczasowa, zawiera dużą liczbę najbardziej przystosowanych rozwiązań. Populacja ta dopuszcza również duplikację osobników, zwłaszcza kiedy był realizowany proces selekcji ze zwracaniem.
Selekcja	(ang. *Selection*) - proces odpowiadający za wybór rodziców, mających następnie przystąpić do realizacji operatora krzyżowania. Proces selekcji uzależniony jest od wartości funkcji przystosowania osobników.
Sieć Feistela	(ang. *Feistel Network*) - struktura pozwalająca na szyfrowanie i deszyfrowanie danych tym samym algorytmem, mimo że funkcja rundy f jest nieodwracalna. Struktura ta znajduje zastosowanie w bufowie symetrycznych szyfrów blokowych.

Pojęcie	Opis
Sukcesja	(ang. *Succession*) - proces odpowiadający za kreowanie nowej populacji bazowej na podstawie osobników znajdujących się w populacji potomnej oraz poprzedniej populacji bazowej.
Szyfr asymetryczny	(ang. *Assymetric Cipher*) - rodzaj algorytmu szyfrującego, zwany również szyfrem z kluczem publicznym, w którym klucz szyfrujący jest różny od klucza deszyfrującego. Dzięki temu rozwiązaniu klucz szyfrujący może zostać udostępniony publicznie bez utraty bezpieczeństwa informacji.
Szyfr blokowy	(ang. *Block Cipher*) - rodzaj algorytmu w którym tekst jawny zostaje podzielony na kilka, równej długości, bloków z danymi. Następnie każdy blok tekstu jawnego zamieniany jest na odpowiadający mu blok szyfrogramu.
Szyfr homofoniczny	(ang. *Homophonic Cipher*) - rodzaj szyfru w którym większa liczba znaków odpowiada pojedynczej literze. Litera "a" może być reprezentowany np. przez znaki "b" lub "z.
Szyfr strumieniowy	(ang. *Stream Cipher*) - szyfr przetwarzający tekst jawny bit po bicie lub znak po znaku. Szyfry te znajdują największe zastosowanie w przypadku, gdy nie jest znany rozmiar danych wejściowych.
Szyfr symetryczny	(ang. *Symmetric Cipher*) - rodzaj algorytmu szyfrującego w którym klucz szyfrujący jest taki sam jak klucz deszyfrujący tj. $K_E = K_D$.
Szyfrogram	(ang. *Ciphertext*) - zaszyfrowana i całkowicie nieczytelna informacja. Może przypominać ciąg pseudo-losowo wygenerowanych znaków binarnych lub obraz imitujący szum. Dane wejściowe dla procesu deszyfrowania oraz wynik procesu szyfrowania.
Szyfrowanie	(ang. *Encryption*) - proces transformacji tekstu jawnego na szyfrogram przy pomocy odpowiedniego klucza i algorytmu szyfrującego.

Pojęcie	Opis
Tekst jawny	(ang. *Plaintext*) - czytelny i zrozumiały dla każdego tekst, obraz lub plik audio. Dane wejściowe dla procesu szyfrowania oraz wynik poprawnego procesu deszyfrowania.

Spis algorytmów

1 Podstawowy algorytm ewolucyjny 29

2 Podstawowy algorytm memetyczny 43
3 Algorytm wspinaczki 46
4 Algorytm symulowanego wyżarzania 48
5 Algorytm przeszukiwania z listą Tabu 50

6 Pseudokod procesu przygotowania zbioru tekstów dla ataku *MASA* 58
7 Pseudokod ataku *MASA* 59
8 Pseudokod algorytmu *DPSO* 64

Spis rysunków

2.1 Uproszczony schemat działania algorytmu szyfrującego . . . 7
2.2 Algorytmy kryptografii klasycznej . . . 9
2.3 Algorytmy symetryczne oraz asymetryczne kryptografii współczesnej . . . 11
2.4 Pojedynczy cykl sieci Feistela . . . 13
2.5 Algorytm szyfrujący *FEAL* . . . 16
2.6 Funkcja rundy f algorytm szyfrującego *FEAL* . . . 17
2.7 Sześcio-rundowy algorytm szyfrujący *DES* . . . 19
2.8 Funkcja rundy f algorytmu *DES* . . . 19

3.1 Schemat przykładowego algorytmu ewolucyjnego (genetycznego) . . . 30
3.2 Przykładowe warianty operatora krzyżowania . . . 34
3.3 Przykład krzyżowania z częściowym odwzorowaniem . . . 37
3.4 Przykład krzyżowania z porządkowaniem . . . 38
3.5 Przykład krzyżowania cyklicznego . . . 38
3.6 Przykładowe warianty operatora mutacji . . . 39

5.1 Analiza różnicowa szyfru *FEAL* . . . 53
5.2 Ostatnia runda algorytmu *FEAL* . . . 54
5.3 3-rundowe charakterystyki Ω_P^1 i Ω_P^2 algorytmu *DES* . . . 56
5.4 Diagram popularności metaheurystycznej kryptoanalizy na przestrzeni ostatnich lat . . . 60
5.5 Zasada działania operatora heurystycznej negacji [22] . . . 61
5.6 Schemat działania algorytmu *DPSO* . . . 65

6.1 Zestawienie poprawnie odgadniętych bitów alg. *MASA* - klucz #1 . . . 73
6.2 Zestawienie poprawnie odgadniętych bitów alg. *NEA* - klucz #1 . . . 73
6.3 Przebieg wartości F_f testu #1 dla alg. *MASA* i *NEA* - klucz #1 . . . 74
6.4 Rozkład wartości F_f dla alg. *MASA* - klucz #1 . . . 75
6.5 Rozkład wartości F_f dla alg. *NEA* - klucz #1 . . . 75
6.6 Zestawienie poprawnie odgadniętych bitów alg. *MASA* - klucz #2 . . . 76
6.7 Zestawienie poprawnie odgadniętych bitów alg. *NEA* - klucz #2 . . . 77
6.8 Przebieg wartości F_f testów #4 i #10 dla alg. *MASA* i *NEA* - klucz #2 . . . 77
6.9 Rozkład wartości F_f dla alg. *MASA* - klucz #2 . . . 78
6.10 Rozkład wartości F_f dla alg. *NEA* - klucz #2 . . . 78
6.11 Zestawienie poprawnie odgadniętych bitów alg. *MASA* - klucz #3 . . . 79
6.12 Zestawienie poprawnie odgadniętych bitów alg. *NEA* - klucz #3 . . . 79
6.13 Przebieg wartości F_f testów #6 i #5 dla alg. *MASA* i *NEA* - klucz #3 . . . 80
6.14 Rozkład wartości F_f dla alg. *MASA* - klucz #3 . . . 81
6.15 Rozkład wartości F_f dla alg. *NEA* - klucz #3 . . . 81

6.16 Zestawienie poprawnie odgadniętych bitów alg. *MASA* - klucz #4 82
6.17 Zestawienie poprawnie odgadniętych bitów alg. *NEA* - klucz #4 82
6.18 Przebieg wartości F_f testu #1 dla alg. *MASA* i *NEA* - klucz #4 83
6.21 Zestawienie poprawnie odgadniętych bitów alg. *MASA* - klucz #5 83
6.19 Rozkład wartości F_f dla alg. *MASA* - klucz #4 84
6.20 Rozkład wartości F_f dla alg. *NEA* - klucz #4 84
6.22 Zestawienie poprawnie odgadniętych bitów alg. *NEA* - klucz #5 85
6.23 Przebieg wartości F_f testu #11 dla alg. *MASA* i *NEA* - klucz #5 85
6.24 Rozkład wartości F_f dla alg. *MASA* - klucz #5 86
6.25 Rozkład wartości F_f dla alg. *NEA* - klucz #5 86
6.26 Zestawienie poprawnie odgadniętych bitów alg. *DPSO* - klucz #1 88
6.27 Zestawienie poprawnie odgadniętych bitów alg. *DPSO* - klucz #2 88
6.28 Przebieg wartości F_f testu #1 dla alg. *DPSO* - klucz #1 i #2 89
6.29 Rozkład wartości F_f dla alg. *DPSO* - klucz #1 90
6.30 Rozkład wartości F_f dla alg. *DPSO* - klucz #2 90
6.31 Zestawienie poprawnie odgadniętych bitów alg. *DPSO* - klucz #3 91
6.32 Zestawienie poprawnie odgadniętych bitów alg. *DPSO* - klucz #4 91
6.33 Zestawienie poprawnie odgadniętych bitów alg. *DPSO* - klucz #5 91
6.34 Przebieg wartości F_f testu #2 dla alg. *DPSO* - klucze #3, #4 i #5 92
6.35 Rozkład wartości F_f dla alg. *DPSO* - klucz #3 93
6.36 Rozkład wartości F_f dla alg. *DPSO* - klucz #4 93

A.1 Zestawienie poprawnie odgadniętych bitów alg. *MASA* dla Ω_2 - klucz #1 116
A.2 Zestawienie poprawnie odgadniętych bitów alg. *NEA* dla Ω_2 - klucz #1 116
A.3 Zestawienie poprawnie odgadniętych bitów alg. *MASA* dla Ω_2 - klucz #2 119
A.4 Zestawienie poprawnie odgadniętych bitów alg. *NEA* dla Ω_2 - klucz #2 119
A.5 Zestawienie poprawnie odgadniętych bitów alg. *MASA* dla Ω_2 - klucz #3 119
A.6 Zestawienie poprawnie odgadniętych bitów alg. *NEA* dla Ω_2 - klucz #3 119
A.7 Zestawienie poprawnie odgadniętych bitów alg. *MASA* dla Ω_2 - klucz #4 124
A.8 Zestawienie poprawnie odgadniętych bitów alg. *NEA* dla Ω_2 - klucz #4 124
A.9 Zestawienie poprawnie odgadniętych bitów alg. *MASA* dla Ω_2 - klucz #5 124
A.10 Zestawienie poprawnie odgadniętych bitów alg. *NEA* dla Ω_2 - klucz #5 124

Spis tabel

6.1 Lista wykorzystywanych podkluczy w kryptoanalizie szyfru *FEAL* i *DES* 67
6.2 Lista parametrów algorytmu *MASA* dla szyfrów *FEAL* i *DES* 69
6.3 Lista parametrów algorytmu *DPSO* dla szyfru *DES* 69
6.4 Lista parametrów algorytmu *NEA* dla szyfru *FEAL* oraz *DES* 70
6.5 Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* oraz Ω_P^1 71
6.6 Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* oraz Ω_P^2 72
6.7 Wartości funkcji przystosowania algorytmu *DPSO* dla szyfru *DES* 87

7.1 Zestawienie liczby sprawdzonych podkluczy, potrzebnych do odgadnięcia poprawnego klucza, alg. *MASA* i *NEA* dla szóstej rundy szyfru *DES* 95
7.2 Zestawienie liczby wszystkich sprawdzonych podkluczy alg. *MASA* i *NEA* dla szóstej rundy szyfru *DES* 96
7.3 Całkowita i średnia liczba wszystkich kluczy każdego z ataków 97
7.4 Zestawienie czasu działania, potrzebnego do odgadnięcia poprawnego podklucza, alg. *MASA* i *NEA* dla szóstej rundy szyfru *DES* 98
7.5 Zestawienie całkowitego czasu działania alg. *MASA* oraz *NEA* dla szóstej rundy szyfru *DES* . . . 99
7.6 Szacowany całkowity i średni czas kryptoanalizy każdego z ataków 100

A.1 Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_1 - klucz #1 . . . 117
A.2 Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_2 - klucz #1 . . . 118
A.3 Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_1 - klucz #2 . . . 120
A.4 Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_2 - klucz #2 . . . 121
A.5 Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_1 - klucz #3 . . . 122
A.6 Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_2 - klucz #3 . . . 123
A.7 Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_1 - klucz #4 . . . 125
A.8 Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_2 - klucz #4 . . . 126
A.9 Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_1 - klucz #5 . . . 127
A.10 Wartości funkcji przystosowania algorytmów *MASA* i *NEA* dla szyfru *DES* i Ω_2 - klucz #5 . . . 128
A.11 Wartości funkcji przystosowania algorytmu *DPSO* dla szyfru *DES* - klucze #1 i #2 129
A.12 Wartości funkcji przystosowania algorytmu *DPSO* dla szyfru *DES* - klucze #3 i #4 130
A.13 Wartości funkcji przystosowania algorytmu *DPSO* dla szyfru *DES* - klucz # 5 131

B.1 Lista par tekstów jawnych i szyfrogramów wykorzystana podczas kryptoanalizy różnicowej Ω_1 . . 139