Univerzitet u Sarajevu
Elektrotehnički Fakultet
Odsjek za Računarstvo i Informatiku

Seminarski rad iz predmeta Optimizacija Resursa

Upotreba PSO Algoritma za Analizu Kriptovanog Teksta

Student:
Krupalija Ehlimana, 1431/17461

Predmetni nastavnik:
v. prof. dr. Samim Konjicija, dipl. ing. el.

Januar, 2019

Sažetak

U ovom radu predstavljeni su osnovni koncepti kriptografije, odnosno transformacije podataka zbog njihove zaštite, na kojima počiva i kriptoanaliza, koja ima zadatak da dekodira enkriptovane podatke bez poznavanja načina na koji su podaci transformisani. U tu svrhu korišten je PSO (*Particle Swarm Optimization*) algoritam, koji vrši pomijeranje čestica (koje predstavljaju potencijalna rješenja) u problemskom prostoru do ispunjenja uslova zaustavljanja algoritma. Nakon prilagođavanja problema kriptoanalize *cipher* transpozicijskih kriptografskih sistema u oblik pogodan za rješavanje ovim algoritmom, izvršena je njegova implementacija, testiranje performansi te analiza rezultata (s aspekta uspješnosti i efikasnosti).

Abstract

In this paper, main ideas on which criptography (the transformation of data in order to increase their security) is founded are presented. Cryptanalysis is based on the same ideas, but its goal is to decode the encrypted data without any knowledge of the way in which the data has been transformed to its final form. In order to achieve this, PSO (*Particle Swarm Optimization*) algorithm is used: it moves particles (potential solutions of the problem) in the problem area until one or more of the conditions for the algorithm's termination are fulfilled. The problem of cryptanalysing cipher transposition cryptography systems first needs to be changed so that it is suitable for being solved by this algorithm. Afterwards, the algorithm for solving this problem has been developed, its performances tested and, finally, an analysis of the results has been conducted (in terms of success in decoding the messages and efficiency in doing so).

Sadržaj

1 Uvod

1.1 Opis problema

Često postoji potreba da se podaci koje je potrebno poslati nekome zaštite od neovlaštenog pristupa. Postoji veliki broj **kriptografskih metoda** koje vrše transformaciju podataka prije njihovog slanja primaocu, kako bi se potencijalno presretanje poruke učinilo beskorisnim, odnosno kako bi se onemogućilo da osoba koja ne treba da primi poruku, ukoliko je uspije presresti, istu bude u stanju pročitati (odnosno dešifrovati). Sve ove metode mogu se podijeliti u dvije velike klase:

1. Metode koje koriste tajni ključ (**simetrična kriptografija**), u koje spadaju algoritmi poput *Cezarovog koda*;

2. Metode koje koriste javni ključ (**asimetrična kriptografija**), u koje spadaju kriptosistemi poput *RSA*.

Glavna razlika između ove dvije klase je u samom ključu, odnosno funkciji koja se koristi za transformaciju podataka (tj. njihovo skrivanje - enkripciju). Metode koje koriste tajni ključ oslanjaju se na činjenicu da bi za dešifrovanje poruke bez poznavanja ključa bilo potrebno provjeriti **veoma veliki broj kombinacija** kako bi se otkrio šablon (odnosno funkcija) kojim se može izvršiti dešifrovanje, te da takvo nešto nije izvodivo u realnom vremenu. S druge strane, metode koje koriste javni ključ ne oslanjaju se na tajnost ključa, već na **kompleksnost izračuna vrijednosti privatnih ključeva**, na osnovu kojih je dobivena vrijednost jvnih ključeva. Na ovaj način uklanja se jedan veliki sigurnosni propust metoda simetrične kriptografije: potreba za slanjem informacije o funkciji koja se koristi za enkripciju, koja se može presresti i na taj način omogućiti dekripciju originalnih podataka. [1]

Analiza enkriptovanog teksta te pokušaj rekonstrukcije funkcije za dekripciju istog problem je kojim se bavi **kriptoanaliza**. No, osim javnosti ključa i težine pronalaska pravila kojim je moguće obuhvatiti sve ulazne podatke, u analizi šifrovanih podataka potrebno je obratiti pažnju na još jednu mogućnost: da se pravilo enkripcije ne vrši nad pojedinim znakovima, već nad riječima ili blokovima podataka. Iz tog razloga, kriptografski sistemi dijele se u dvije kategorije [2]:

1. **Šifre** (*ciphers*), pri čemu se enkripcija vrši nad pojedinim karakterima (zamjena drugim karakterima, promjena pozicije u poruci i sl.);

2. **Kodovi** (*codes*), pri čemu se enkripcija vrši nad više karaktera (podaci se dijele u blokove koristeći neko pravilo, te se oni zatim tretiraju kao pojedinačni karakteri).

Cipher sistemi najčešće koriste vrlo jednostavne funkcije za transformaciju podataka, a te funkcije se zasnivaju ili na principu **transpozicije** (zamjene mjesta karaktera u poruci), ili na principu **substitucije** (zamjene karaktera drugim karakterima). [2] Vršenje kriptoanalize ovakvih sistema veoma je jednostavno, jer ukoliko se enkripcija vrši nad pojedinačnim karakterima, frekvencija njihovog pojavljivanja biti će jednaka frekvenciji pojavljivanja odgovarajućim karakterima jezika na kojem je poruka poslana. Na taj način moguće je veoma brzo otkriti ključ prema kojem se vrši transpozicija (ili substitucija), zbog čega se ovakvi sistemi danas najčešće ne koriste. [3]

Code sisteme mnogo je teže dekriptovati, budući da isti broj karaktera u šifrovanoj poruci mogže imati različit broj karaktera u riječi koja opisuje taj blok. I sami blokovi u okviru šifrovane poruke mogu imati varijabilnu dužinu, no u tom slučaju kompleksnost izračuna se drastično povećava, kao i narušenost sigurnosti zbog potrebe za slanjem ključa. Iz tog razloga varijabilna dužina se rijetko

koristi u kodnim sistemima. [2]

Veliki broj različitih pristupa primjenjen je kako bi se kreirali jedinstveni algoritmi za enkripciju podataka. Rotori, *shift* registri, *hash* funkcije, te veliki broj protokola (poput *Diffie-Hellman*-a) samo su neki od najpoznatijih. Zbog širine oblasti, u ovom radu biti će implementirano rješenje samo za transpozicijske *cipher* sisteme.

1.2 Pregled literature

Primarna literatura na osnovu koje je obrađen problem kriptoanalize, te algoritam *Particle Swarm Optimization* koji će se koristiti za rješavanje problema, sastoji se iz sljedećih stručnih knjiga:

1. *Basic Cryptanalysis*, priručnik za kriptoanalizu sastavljen od strane američke vojske, u kojem su objašnjeni osnovni kriptografski principi, te u kojem je moguće naći praktične primjere kriptoanalize; [2]

2. *Applied Cryptanalysis: Breaking Ciphers in the Real World*, knjiga u kojoj se nalaze detaljni opisi velikog broja postojećih kriptografskih sistema te način vršenja dekripcije poruka kodiranih korištenjem istih; [4]

3. *Particle Swarm Optimization and Intelligence: Advances and Applications*, knjiga koja detaljno opisuje način rada osnovnog PSO algoritma i njegovih brojnih modifikacija; [5]

4. *Particle Swarm Optimization*, knjiga u kojoj se nalaze detaljni opisi razloga, načina poboljšavanja osnovnog PSO algoritma te njihovih performansi; [6]

5. *Cryptanalysis of Block Ciphers via Improvised Particle Swarm Optimization and Extended Simulated Annealing Techniques*, članak koji opisuje pretvaranje kriptoanalitičkih problema u oblik pogodan za optimizaciju korištenjem PSO algoritma, uključujući formiranje funkcije cilja te primjenu PSO algoritma za optimizaciju iste. [7]

1.3 Moguće aplikacije u praksi

Kriptografija je oblast koja ima široku primjenu u praksi, jer je sigurnost neizostavan aspekt koji ni u jednom trenutku ne smije biti narušen. Kriptoanaliza je imala veliku primjenu tokom svjetskih ratova, kada je bilo neophodno presretati i dešifrovati neprijateljske poruke (koje su bile kodirane, upravo kako bi se to spriječilo), te se i danas **aktivno izučava u vojsci**. Osim mogućnosti dešifrovanja neprijateljskih poruka, kriptoanaliza je koristan alat i za poboljšavanje vlastitih kriptografskih sistema, te američka vojska koristi ovakav pristup kako bi konstantno vršila unaprjeđenje šifrovanja poruka putem traženja rješenja koje bi iste dešifrovalo. [2]

Kriptoanaliza ima široku primjenu i u **oblasti računarskih mreža**, gdje se koristi kako bi se testirala sigurnost *hash* protokola koji se koriste kako bi se zaštitili podaci korisnika koji koriste Internet. Funkcije koje su se dugo vremena smatrale sigurnim (poput MD5 funkcije) lako su dešifrovane koristeći kriptoanalizu, te je ova nauka omogućila pronalazak slabosti takvih funkcija kako bi se iste mogle poboljšati, a samim tim i sigurnost svih korisnika. [8] Kriptoanaliza je oblast koja se razvija, jer se svakodnevno pronalaze novi načini za vršenje enkripcije podataka, te se u skladu s tim konstantno razvijaju i novi načini za dekripciju istih.

2 Korišteni Algoritam

2.1 Opis rada korištenog algoritma

Algoritam koji će biti korišten za rješavanje problema kriptoanalize naziva se ***Particle Swarm Optimization***, te je napravljen po uzoru na ponašanje jata ptica i drugih grupa životinja. Ovaj algoritam pripada klasi populacijskih algoritama, odnosno u svakoj iteraciji postoji populacija - *swarm* potencijalnih rješenja koja se zatim poboljšavaju dok se ne ispuni neki od uslova zaustavljanja algoritma. [5]

Populacija se definiše kao set od n čestica: $S = \{x_1, x_2, \ldots, x_n\}$, pri čemu svaka čestica predstavlja vektor od m elemenata: $x_i = \{x_{i1}, x_{i2}, \ldots, x_{im}\}$, $i = 1, \ldots, n$.

Za svaku česticu računa se vrijednost funkcije cilja: $f_i = f(x_i)$, čiji optimum je potrebno naći.

Za razliku od genetičkih algoritama, u kojem se stvaraju generacije populacija te vrši uništavanje čestica i njihove zamjene novim česticama, u PSO algoritmu vrši se **pomijeranje** čestica u problemskom prostoru. U tu svrhu definiše se **brzina čestice**: $v_i = \{v_{i1}, v_{i2}, \ldots, v_{im}\}^T$, $i = 1, \ldots, n$ koja omogućava da potencijalna rješenja izađu iz lokalnih ekstrema, kako bi se na taj način nastavilo pretraživanje prostora i pronalazak globalnog optimuma. [5]

Poput tabu pretraživanja, i u PSO algoritmu postoji mogućnost da potencijalna rješenja dođu do globalnog optimuma te zatim nastave pretraživanje prostora i pređu u nepovoljnije tačke. Iz tog razloga potrebno je imati i memoriju najboljih rješenja: $P = \{p_1, p_2, \ldots, p_n\}$, koja se sastoji od svih najboljih rješenja pojedinačnih čestica.

Svaka čestica ima svoje informante, odnosno čestice na osnovu čijih informacija se odlučuje u kojem smjeru će se čestica kretati. U svakoj iteraciji algoritma vrši se nasumični odabir K čestica koje postaju informanti čestice koja se trenutno razmatra. Zatim se čestica pomijera u onom smjeru u kojem se nalazi čestica koja ima najbolju poziciju (najbolje potencijalno rješenje).

U okviru algoritma vrše se dvije elementarne operacije [5]:

1. **Promjena brzine svih pojedinačnih čestica**, koja se računa prema sljedećoj formuli:

$$v_{ij}(t+1) = c_1 \cdot v_{ij}(t) + c_2 \cdot (p_{ij}(t) - x_{ij}(t)) + c_3 \cdot (p_{gj}(t) - x_{ij}(t)) \tag{1}$$

$$i = 1, ..., n, j = 1, ..., m$$

   c_1, c_2 i c_3 predstavljaju tzv. **težinske faktore**. p_{ij} predstavlja najbolje memorisano rješenje od strane date čestice, dok p_{gj} predstavlja najbolje trenutno rješenje informanata čestice.

2. **Pomijeranje svih varijabli u problemskom prostoru**, koje se vrši prema sljedećoj formuli:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{2}$$

$$i = 1, ..., n, j = 1, ..., m$$

Preporučene vrijednosti svih prethodno navedenih parametara prikazane su u Tabeli 2.1: [9]

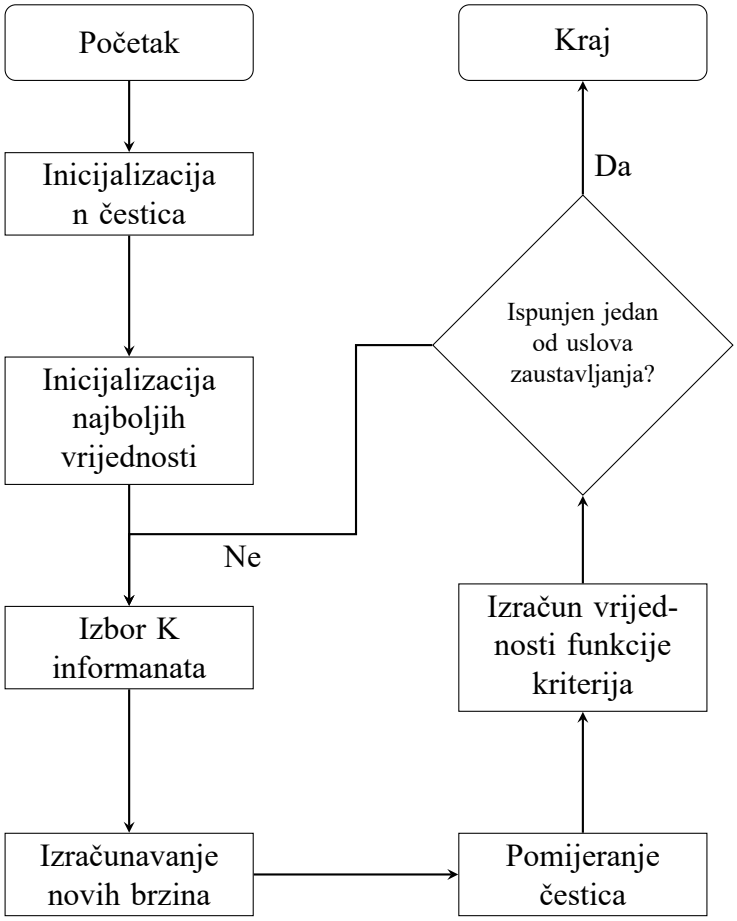
Parametar	**Preporučena vrijednost**
m	20 - 40
K	3 - 5
c_1	0.7 ili 0.8
c_2, c_3	`random(0,` c_{max}`)`
c_{max}	1.47 ili 1.62

Tabela 2.1: Preporučene vrijednosti parametara PSO algoritma

Algoritam 1 i Slika 2.1 prikazuju pseudokod osnovne verzije PSO i način na koji on funkcioniše. Nakon inicijalizacije roja čestica i njihovih najboljih vrijednosti, iterativno se vrši izbor informanata za sve pojedinačne čestice, ažuriranje brzina svih čestica, njihovih pozicija te izračunavanje novih rješenja za sve pojedinačne čestice. Nakon toga provjerava se da li je ispunjen neki od uslova za zaustavljanje algoritma, u kojem slučaju se kao najbolje uzima najbolje od svih najboljih rješenja pojedinačnih čestica koji se nalaze u memoriji.

```
PSO(N, n, K)
t ← 0
S ← INITIALIZESWARM(K, n)
P ← ARGMIN(S)
while t < N do
    S.CHOOSEINFORMANTS()
    S.CALCULATEVELOCITIES()
    S.MOVEPARTICLES()
    P.UPDATEVALUES(S)
end
return MIN(P)
```

Algoritam 1: Pseudokod PSO algoritma



Slika 2.1: Blok-dijagram PSO algoritma

Postoji mnogo načina da se izvrši poboljšanje osnovnog algoritma, a neki od njjih su: [5]

1. **Uvođenje inercijske težine**:

   Brzina je veličina koja direktno utječe na promjenu pozicije svih čestica, odnosno na pomijeranje potencijalnih rješenja u problemskom prostoru. Osnovna formula za promjenu brzine pokazala se neadekvatnom, zbog pojave tzv. ***swarm explosion*** efekta, pri čemu se brzina za veoma kratko vrijeme dostigne tako velike vrijednosti da sve čestice divergiraju, te je za posljedicu pronalazak najboljeg rješenja teško moguće (budući da se čestice ne teže k pretraživanju prostora koji nudi bolja potencijalna rješenja).

   Prvi pokušaj da se riješi problem divergencije čestica bio je uvođenje maksimalne brzine, no u tom slučaju, iako nema potpune divergencije čestica, nema ni njihove konvergencije k najboljem rješenju. Umjesto da se koncentrišu u okolini najboljih rješenja, čestice napuštaju taj dio problemskog prostora i osciliraju oko ekstrema, bez mogućnosti da ga pronađu. [5]

   Iz ovog razloga uvedena je nova veličina koja se naziva **inercijskom težinom**, čija je funkcija stabilizacija brzine, odnosno smanjenje perturbacija koje dovode do oscilacije čestica od povoljnih potencijalnih rješenja. Inercijska težina `w` mijenja vrijednost prethodne brzine, kako bi ona imala što manji utjecaj na njenu novu vrijednost, prema sljedećoj formuli: [5]

$$v_{ij}(t+1) = w \cdot c_1 \cdot v_{ij}(t) + c_2 \cdot (p_{ij}(t) - x_{ij}(t)) + c_3 \cdot (p_{gj}(t) - x_{ij}(t)) \qquad (3)$$

$$i = 1, ..., n, j = 1, ..., m$$

2. **Uvođenje susjedstva**:

   U osnovnoj verziji PSO algoritma, sve čestice imaju memoriju u kojoj se nalazi najpovoljnija tačka koja je posjećena do datog trenutka. To znači da je za memoriju potrebno izdvojiti istu količinu resursa kao i za same čestice, odnosno da za `m` čestica treba postojati i `m` varijabli koje čuvaju informaciju o najboljim rješenjima.

   Kako bi se smanjila količina resursa potrebnih za čuvanje informacija o najboljim rješenjima, uveden je koncept **susjedstva**, koji se zasniva na udaljenosti čestica jednih od drugih, no ne u prostornom smislu (bez obzira na moguću razliku u prostornoj udaljenosti, susjedima se smatraju one čestice koje su najbliže jedne drugima, što se naziva socijalnim susjedstvom, u odnosu na klasično geografsko susjedstvo). Za `k` susjeda formira se ciklus koji povezuje jednu memorijsku ćeliju sa svim odabranim susjedima. Ažuriranje memorije se vrši samo ukoliko se pronađe rješenje koje je bolje od najboljeg dotad pronađenog rješenja (od svih susjednih čestica). Na ovaj način štede se resursi, smanjuje se broj upisa u memoriju, te broj neophodnih poređenja svih ćelija memorije na kraju rada programa. [6]

3. **Uvođenje većeg broja *swarm*-ova**:

   Kako bi pretraživanje prostora bilo još efikasnije, odnosno kako bi se u što kraćem vremenu pretražio što veći prostor, uvedena je modifikacija PSO algoritma koja se naziva **grupni PSO**. U okviru ovog algoritma kreira se više *swarm*-ova, pri čemu su svi neovisni jedni o drugima i imaju vlastite funkcije za pomijeranje i brzinu.

   U idealnom slučaju, svi *swarm*-ovi nalaze se u različitim dijelovima problemskog prostora, te čuvaju informacije o svojim globalnim ekstremima, od kojih se na kraju rada programa bira najbolji. Iako je preklapanje nekih čestica veoma teško izbjeći (posebno u slučaju kada se koristi veliki broj *swarm*-ova), ova modifikacija pokazala se kao nešto brža u odnosu na osnovnu verziju PSO algoritma. Budući da je pogodan i za paralelno izvršavanje, grupni PSO može postići još bolje performanse ukoliko se različiti dijelovi algoritma (pretraživanje problemskog prostora koristeći različite *swarm*-ove) budu izvršavali u isto vrijeme. [7]

2.2 Svođenje opisanog problema u formu korištenog algoritma

PSO je algoritam koji vrši traženje optimuma date funkcije cilja. U problemu kriptoanalize potrebno je izvršiti dekodiranje enkriptovane poruke sa što manje grešaka, te iz tog razloga cilj optimizacije predstavlja **tačnost dekodirane poruke**, odnosno što manje odstupanje dekodirane poruke od prirodnog jezika.

Potencijalna rješenja predstavljaju vrijednosti dekodiranih poruka, odnosno permutacije karaktera engleskog jezika koje se sastoje od predefinisanog broja karaktera. Budući da je i za relativno mali broj karaktera broj njihovih permutacija veoma veliki (`n!`), problemski prostor u kojem se vrši pretraživanje biti će veoma veliki, te će se na ovaj način moći izvršiti adekvatno testiranje PSO algoritma pri rješavanju problema kriptoanalize.

Funkcija kriterija je sličnost frekvencija karaktera u poruci predefinisanim frekvencijama karaktera engleskog jezika. Što je veća sličnost, to je veća mogućnost da potencijalno rješenje ima smisla (odnosno, da predstavlja neku smislenu poruku), s tim da bi vrijednost funkcije kriterija bila ista za sve permutacije dekodirane poruke (od kojih je samo jedna tačna), te se iz tog razloga u funkciju kriterija uvodi i sličnost frekvencija parova karaktera [7] (kako bi bolju vrijednost imala npr. permutacija `SENDHELP` od `ENDESHLP`, bez obzira što su frekvencije karaktera u oba slučaja iste).

Frekvencije pojedinačnih karaktera (*monograma*), kombinacija od po dva karaktera (*bigrama*), kombinacija od po tri karaktera (*trigrama*) i kombinacija od po četiri karaktera (*quadgrama*) engleskog jezika koje će biti korištene pri kriptoanalizi izračunate su na osnovu analize teksta koji se sastoji od nekoliko milijardi karaktera, iz kojeg razloga imaju dovoljnu vjerodostojnost da bi se koristile u analizi poruka. [10]

Postoje tri funkcije kriterija, koje se koriste u različitim dijelovima algoritma:

1. *Bigram* funkcija, koja vrši usporedbu da li se *bigrami* u poruci razlikuju od standardne frekvencije *bigrama*:

$$f(x) = \sum_{i=1}^{26} \sum_{j=1}^{26} SDF[i,j] - DDF[i,j] \tag{4}$$

2. *Trigram* funkcija, koja vrši usporedbu da li se *trigrami* u poruci razlikuju od standardne frekvencije *trigrama*:

$$f(x) = \sum_{i=1}^{26} \sum_{j=1}^{26} \sum_{k=1}^{26} STF[i,j,k] - DTF[i,j,k] \tag{5}$$

3. *Quadgram* funkcija, koja vrši usporedbu da li se *quadgrami* u poruci razlikuju od standardne frekvencije *quadgrama*:

$$f(x) = \sum_{i=1}^{26} \sum_{j=1}^{26} \sum_{k=1}^{26} \sum_{l=1}^{26} SQF[i,j,k,l] - DQF[i,j,k,l] \tag{6}$$

Veličine u funkcijama kriterija imaju sljedeća značenja:

– `SDF[i, j]`: standardna frekvencija *bigrama* (niza od dva karaktera) koja se izračunava na osnovu podataka iz [10];

– `DDF[i, j]`: frekvencija *bigrama* izmjerena u poruci x koja predstavlja potencijalno rješenje;

– `STF[i, j, k]`: standardna frekvencija *trigrama* (niza od tri karaktera) koja se izračunava na osnovu podataka iz [10];

– `DTF[i, j, k]`: frekvencija *trigrama* izmjerena u poruci x koja predstavlja potencijalno rješenje;

– `SQF[i, j, k, l]`: standardna frekvencija *quadgrama* (niza od četiri karaktera) koja se izračunava na osnovu podataka iz [10];

– `DQF[i, j, k, l]`: frekvencija *quadgrama* izmjerena u poruci x koja predstavlja potencijalno rješenje.

Cilj algoritma je **minimizacija** ove funkcije - optimalno rješenje je ono koje je najprirodnije, tj. ono koje najmanje odstupa od prirodnog jezika (što je ispunjeno kad su frekvencije pojedinačnih karaktera i bigrama najsličnije standardnim frekvencijama).

Algoritam ima sljedeće uslove zaustavljanja:

1. **Dostizanje maksimalnog broja iteracija** (parametra koji je moguće mijenjati), koji za cilj ima zaustavljanje rada algoritma nakon pomijeranja čestica dovoljan broj puta. Veličina ovog parametra određuje se testiranjem, te može varirati ovisno o problemu koji se rješava. Broj iteracija ne smije biti premali, jer u tom slučaju čestice ne istraže prostor dovoljno i ne konvergiraju ka optimumu, no broj iteracija ne smije biti ni prevelik, jer u tom slučaju vrijeme izvršavanja algoritma može postati neprihvatljivo dugo.

2. **Izlazak svih čestica iz dopuštenog prostora**, u kom slučaju više nema čestica koje generišu nova rješenja (jer su pozicije svih čestica nedozvoljene). Čestice dolaze u nedozvoljenu poziciju u trenutku kad više nema mogućnosti za poboljšavanjem rješenja, a čestice nisu dostigle optimalno rješenje (u tom slučaju, svako novo rješenje bi dostizalo gore vrijednosti, tako da se takve čestice prestaju razmatrati).

3. **Dostizanje optimalne vrijednosti**, odnosno trenutak u kojem se pronađe čestica s vrijednošću greške jednakom 0 (a samim tim i vrijednošću funkcije cilja jednakom minimalnoj vrijednosti, koja također iznosi 0).

Konvergencija čestica (odnosno nepronalazak boljeg rješenja nakon jedne iteracije) ne predstavlja uslov zaustavljanja algoritma, jer se brzina čestica mijenja u ovisnosti o boljim pozicijama drugih čestica (informanata), koji se određuju nasumično. Ukoliko se u nekoliko iteracija ne pronađe nijedno bolje rješenje, brzine čestica će se promijeniti na takav način da se približavaju česticama koje imaju povoljnije pozicije, te će se nakon nekog vremena početi generisati bolje pozicije (ukoliko takve postoje).

Brzine čestica su `Boolean` vrijednosti, odnosno mogu imati vrijednost `True` (to znači da se slovo nalazi na pogrešnoj poziciji) ili `False` (to znači da slovo čini *bigram, trigram ili quadgram*, u kom slučaju se ne treba kretati).

Postoji više vrsta pomijeranja čestica u problemskom prostoru:

– **Permutiranje karaktera** koji se ne nalaze na ispravnim pozicijama, koje se vrši na dva načina (svaki način ima 50 % šanse odabira):

  1. Generisanjem nasumične konfiguracije kojom će se svi neispravno pozicionirani karakteri repozicionirati na nove pozicije;
  2. Pronalaskom mjesta na kojem se neki od karaktera koje je potrebno permutirati nalaze u poziciji najboljeg informanta, nakon čega se na to mjesto umeće traženi karakter.

– **Umetanje karaktera** između ispravno pozicioniranih blokova, što se izvršava u posebnom slučaju kada je broj neispravno pozicioniranih karaktera manji od dužine bloka, nakon čega nema mogućnosti da se prethodno opisanim mehanizmom uopće promijeni (a kamoli poboljša) vrijednost funkcije cilja;

– **Promjena funkcije kriterija** (i svih pripadajućih funkcija za izračunavanje frekvencija blokova karaktera), nakon što ne postoje karakteri koje je potrebno permutirati, a nije dostignuta optimalna vrijednost funkcije cilja. U ovom slučaju svi karakteri čine dio nekog bloka, te ih je kao takve nemoguće modificirati koristeći bilo koji od prethodno navedenih mehanizama.

3 Simulacijski rezultati

3.1 Implementacija rješenja

Prvi korak u implementaciji rješenja je izračunavanje prirodnih frekvencija. Neophodne frekvencije učitavaju se iz *file*-ova `monograms.txt`, `bigrams.txt`, `trigrams.txt` i `quadgrams.txt` (za testne svrhe, zbog kratkoće poruka njihove frekvencije uvijek previše odstupaju od prirodnih, pa je u tom slučaju izvršeno računanje prirodne frekvencije samo za karaktere poruke).

Izračunavanje prirodnih frekvencija karaktera implementirano je u okviru funkcija: `naturalMonograms`, `naturalBigrams`, `naturalTrigrams` i `naturalQuadgrams`. Izračunavanje frekvencija za datu poruku implementirano je u okviru funkcije `calculateMessageFrequencies`, koja zatim poziva funkcije `findFrequenciesCharacters`, `findFrequenciesBigrams`, `findFrequenciesTrigrams` i `findFrequenciesQuadgrams` koje vrše pojedinačne izračune.

Funkcija `toString` vrši pretvaranje niza cjelobrojnih vrijednosti (što je način čuvanja koordinata rješenja) u niz karaktera, odnosno poruku.

Funkcija `transpozition` vrši kreiranje kodirane poruke, odnosno pravi nasumičnu konfiguraciju poruke koja se sastoji od istih karaktera, no koji se nalaze na različitim mjestima.

Funkcije `CaesarCode` i `CaesarDecode` vrše kodiranje i dekodiranje poruke respektivno korištenjem Cezarovog koda s nasumičnom vrijednošću ključa.

Funkcije kriterija, odnosno funkcije koje vrše izračun greške - odstupanja od prirodnih frekvencija su funkcije `criterionFunctionBigrams`, `criterionFunctionTrigrams` i `criterionFunctionQuadgrams`.

Funkcije koje vrše ažuriranje brzine čestice ovisno o tome da li se neki karakter ili blok karaktera nalaze na ispravnoj poziciji su funkcije `findCorrectBigrams`, `findCorrectTrigrams` i `findCorrectQuadgrams`.

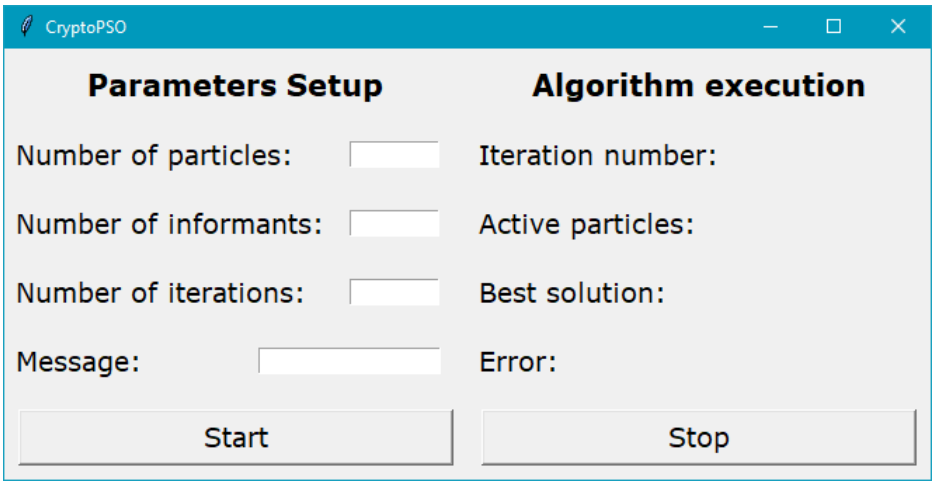
Funkcije koje vrše pronalazak prvog ili neispravnog bloka karakteru sa specificiranim rednim brojem su funkcije `findIncorrectBigram`, `findIncorrectTrigram` i `findIncorrectQuadgram`.

Klasa `Particle`, osim konstruktora, posjeduje i metodu `move` koja, uz korištenje prethodno navedenih funkcija i poštivanje koncepata koji su prethodno opisani u radu, vrši pomijeranje čestice, promjenu režima rada (da li se posmatraju *bigram*-i ili veći blokovi karaktera) i ažuriranje svoje brzine.

Klasa `SwarmOfParticles`, osim inicijalizacije, posjeduje i funkciju `moveAllParticles`, u okviru koje se vrši jedna iteracija algoritma - pomijeranje svih čestica i ažuriranje svih neophodnih parametara.

3.2 Postavka simulacija

Prethodno opisani algoritam implementiran je u programskom jeziku *Python*. Cijela implementacija rješavanja problema kriptoanalize koristeći PSO algoritam dostupna je na:
https://github.com/ehlymana/PSOCryptanalysis. Izgled simulacijske aplikacije prikazan je na Slici 3.1.



Slika 3.1: Simulacijska aplikacija

Aplikacija prihvata samo riječi (ili više riječi bez razmaka) koje se sastoje od velikih slova engleskog alfabeta. Nakon unošenja početnih parametara (broj čestica u roju, broj informanata i broj iteracija) i pritiska na dugme **Start**, vrši se inicijalizacija početnog roja čestica te iterativno pomijeranje čestica u problemskom prostoru.

Na ekranu se prikazuju rezultati izvršavanja algoritma u iterativnom postupku. Prikazuje se dekodirana trenutno najbolja poruka, vrijednost greške u odnosu na stvarnu vrijednost, kao i broj trenutno aktivnih čestica.

U svakom trenutku moguće je prekinuti izvršavanje algoritma pritiskom na dugme **Stop**, u kojem slučaju se najbolje rješenje iz trenutne iteracije proglašava za finalno rješenje.

Aplikacija je testirana na više načina, koristeći različite kriterije za performanse, koje će biti opisane u nastavku.

3.3 Rezultati izvršavanja za različite dužine poruka

Kako bi se dobili što relevantniji i konzistentniji rezultati, za ovu vrstu testiranja korišten je **konstantan broj** čestica (40), informanata (4) i iteracija (50). Taj broj odabran je u skladu s preporukama za **optimalne vrijednosti** [9], kako bi vjerovatnoća za pronalaskom rješenja bila ista (i što veća) bez obzira na varijabilnu dužinu poruka.

Simulacije su za dužine znakova manje od 10 pokrenute po 10 puta, kako bi se smanjio utjecaj randomizma (koji je prisutan u ovom algoritmu) na konačne performanse. Zbog predugog trajanja izvršavanja, simulacije za dužine znakova veće od 9 pokrenute su **samo jednom**. Rezultati simulacije prikazani su u Tabeli 3.1:

- Sve poruke predstavljaju validne riječi, ili nizove riječi engleskog jezika;
- Dužina poruka varira između 1 i 15 (vrijeme izvršavanja za veće dužine poruka je predugo);
- Najbolja finalna rješenja u skladu su s vrijednošću greške, pri čemu je broj karaktera koji su pogrešno pozicionirani jednak vrijednosti greške;
- Broj iteracija za dolazak do rješenja povećava se s povećanjem dužine poruke, zbog drastičnog povećanja broja permutacija;

– Vrijeme izvršavanja mjereno je u sekundama, te je vidljivo njegovo drastično povećanje za veću dužinu poruka.

Poruka	**Dužina poruke**	**Finalno rješenje**	**Vrijednost greške**	**Broj iteracija**	**Vrijeme izvršavanja**
A	1	A	0	0	0.338
TO	2	TO	0	0	0.435
THE	3	THE	0	0	0.456
STAR	4	STAR	0	0	0.314
APRIL	5	APRIL	0	0	0.404
BIRDIE	6	BIRDIE	0	0.4	0.666
SOMEDAY	7	SOMEDAY	0	1.1	0.684
COMPUTER	8	COMPUTER	0	2.6	1.242
YESTERDAY	9	YESTERDAY	0	5.8	67.471
BEPEACEFUL	10	BEPEACEFUL	0	6	16.435
SHAKESPEARE	11	SHAKESPEARE	0	8	21.911
THEGOLDENSUN	12	THEGOLDENSUN	0	9	48.137
CONSTELLATION	13	CONSTELATIONL	1	50	1310.583
CHEYENNEANTHEM	14	MYTINDIMESPENG	1	50	2394.452
SEAGULLJONATHAN	15	AGULLJONHANATSE	2	50	3489.189

Tabela 3.1: Rezultati testiranja za različite dužine poruka

3.4 Rezultati izvršavanja za različite vrijednosti parametara

U svrhu testiranja, izvršena je modifikacija sljedećih parametara:

– **Broj iteracija**: U ovisnosti od broja iteracija, vrijednost pronađenog rješenja trebala bi se mijenjati na način da je tačnost za veći broj iteracija veća;

– **Broj čestica**: Za manji broj čestica, biti će generisan manji broj početnih konfiguracija, zbog čega će biti teže formirati ispravne konfiguracije (potreba za mijenjanjem *quadgram*-a dovesti će do povećanja trajanja izvršavanja programa);

– **Broj informanata**: Veći broj informanata trebao bi osigurati brže poboljšanje tačnosti rješenja, zbog veće mogućnosti da je neko od informanata (kojih ima više) u ispravnoj konfiguraciji.

Svi ostali parametri imaju standardne vrijednosti (broj čestica: `40`, broj informanata: `4`, dužina poruke: `8`, broj iteracija: `40`), kako bi njihov utjecaj na izvršavanje programa bio minimalan. Svaka simulacija pokrenuta je 10 puta, te rezultati predstavljaju srednju vrijednost rezultata tokom svih 10 pokretanja. Rezultati simulacije prikazani su u Tabelama 3.2 - 3.4:

Broj iteracija	Srednja vrijednost greške	Vrijeme izvršavanja
1	1.7	0.503
10	0	0.668
25	0	1.375
50	0	0.81
100	0	0.82
250	0	1.677
500	0	0.595
1000	0	0.625

Tabela 3.2: Rezultati testiranja za različit broj iteracija

Broj čestica	Srednja vrijednost greške	Vrijeme izvršavanja
1	1.3	38.213
2	0.6	103.673
5	0.1	14.487
10	0	0.477
25	0	0.65
50	0	0.666
100	0	0.94
125	0	0.914
150	0	0.979

Tabela 3.3: Rezultati testiranja za različit broj čestica

Broj informanata	Srednja vrijednost greške	Vrijeme izvršavanja
1	0	0.845
2	0	0.522
3	0	1.781
4	0	0.613
5	0	0.926
10	0	1.018
25	0	1.848

Tabela 3.4: Rezultati testiranja za različit broj informanata

3.5 Zaključci

Nakon provedenih testiranja, mogu se lako identificirati dobre i loše strane implementirane aplikacije. Korištenje *bigram*-a, *trigram*-a i *quadgram*-a omogućava **gotovo 100 %-nu tačnost** pri dekodiranju poruka, no iz tog razloga vremenske performanse algoritma su **veoma loše**, pri čemu za dekodiranje poruke od 15 karaktera treba **oko 60 minuta**.

Iako na tačnost dekodiranja utječu parametri poput broja čestica, iteracija i informanata, njihova uloga **nije presudna**, te je praćenje preporučenih vrijednosti dovoljno (iako su i mnogo manje veličine dovoljne) kako bi se poruke ovih veličina uspješno dekodirale. Povećanje broja iteracija neko vrijeme poboljšava performanse algoritma, nakon čega one ostaju performanse (razlika između 500 i 1000 iteracija je neznatna). S druge strane, povećanje broja čestica drastično smanjuje trajanje izvršavanja, budući da se vjerovatnoća da će se neka od čestica brzo naći u tačnoj konfiguraciji povećava s brojem čestica. No, za prevelik broj čestica performanse algoritma počinju da degradiraju, zbog čega ovaj broj ne smije biti ni preveliki (posebno s aspekta memorije). Povećanje broja informanata nema nekog značajnog utjecaja na izvršavanje programa.

4 Zaključak

PSO algoritam moguće je **uspješno iskoristiti** za dekodiranje šifrovanih poruka koristeći osobine prirodnih jezika (odnosno, prirodnu frekvenciju). Cezarov kod pokazao se kao **potpuno nevažan** za samo dekodiranje, jer je dovoljno pokrenuti funkciju za identifikaciju takvog kodiranja u poruci kako bi se dobili izvorni karakteri (koji su, naravno, transponirani, nakon čega kreće proces kriptoanalize koristeći prethodno opisane koncepte). No, kao ključni problem nameće se **vrijeme izvršavanja** algoritma, jer osobina PSO algoritma da se nakon nasumične incijalizacije vrijednosti čestica poboljšavaju ovisno o najboljim pronađenim **nije dovoljna** kako bi se prevazišli problemi proizvedeni kroz sam algoritam kriptoanalize.

Prvobitni problem predstavlja **veličina problemskog prostora**, odnosno broj permutacija karaktera koji čine poruku, koji je već za male veličine poruka izrazito veliki, što drastično smanjuje vjerovatnoću da će bilo koja od inicijalnih vrijednosti čestica posjedovati bilo kakvu karakteristiku tačnog rješenja: tačne *bigram*-e, *trigram*-e ili čak *quadgram*-e. PSO algoritam **nije namijenjen za rješavanje ovakvih problema**, posebno iz razloga što brzina ne predstavlja broj za koji se vrijednost rješenja povećava ili smanjuje, već je prilagođena tako da predstavlja vrijednost koja pokazuje da li je traženi karakter dio ispravne konfiguracije ili ga je potrebno ponovo permutirati. Informanti također pružaju **limitirane informacije** o boljim rješenjima, jer je sama veza između različitih rješenja nejasna: na nivou *bigram*-a ili drugih blokova karaktera, informanti ne mogu dati mnogo korisnih informacija, jer su i njihove informacije samo parcijalno tačne, što može dovesti do degradacije rješenja.

Drugi problem predstavlja **trajanje izvršavanja**, na koje uveliko utječe korištenje *quadgram*-a u algoritmu, pri čemu se u svakoj iteraciji algoritma prolazi kroz četverostruku petlju (26^4), što drastično pogoršava performanse algoritma. No, s povećavanjem broja karaktera u poruci, korištenje manjih blokova karaktera jednostavno **nije dovoljno** da bi se dobili tačni rezultati (prevelika je vjerovatnoća da se više karaktera može naći u tako malim nizovima, te algoritam tvori pogrešne kombinacije koje imaju malu grešku s aspekta razlike u prirodnoj frekvenciji), zbog čega je ovaj koncept ugrađen u algoritam na samom početku razvoja.

Za male dužine poruka, ovaj algoritam je sasvim dovoljan da se izvrši **egzaktna i brza kriptoanaliza**, no za veće dužine poruka, njegove performanse drastično degradiraju, iz kojeg razloga bi se problem *quadgram*-a trebao prevazići koristeći **manje matrice**, budući da je većina elemenata u blokovskoj matrici približno jednaka nuli (naprimjer, *quadgram* `YYBB` se nikako ne pojavljuje u engleskom jeziku). Iz tog razloga nema potrebe za vršenjem velikog broja poređenja s takvim vrijednostima, te bi se performanse algoritma mogle značajno poboljšati ukoliko se funkcija kriterija za *quadgram*-e promijeni, kao i sam način čuvanja matrice s ovim koeficijentima.

Reference

[1] R. A. Mollin, *RSA and Public-Key Cryptography*, ser. Discrete Mathematics and Its Applications. Boca Raton, USA: CRC Press LLC, 2003.

[2] C. E. Vuono and T. F. Sikora, *Basic Cryptanalysis*, Department of the Army, Washington, DC, USA, 1990.

[3] A. J. Bagnall, "The applications of genetic algorithms in cryptanalysis," Master's thesis, School of Information Systems, University of East Anglia, 1996.

[4] M. Stamp and R. M. Low, *Applied Cryptanalysis: Breaking Ciphers in the Real World*. New Jersey, USA: John Wiley & Sons, Inc., 2007.

[5] K. E. Parsopoulos and M. N. Vrahatis, *Particle Swarm Optimization and Intelligence: Advances and Applications*. Hershey, USA: Information Science Reference, 2010.

[6] M. Clerc, *Particle Swarm Optimization*. London, UK: Hermes Science/Lavoisier, 2006.

[7] N. N. and G. R. Rao, "Cryptanalysis of block ciphers via improvised particle swarm optimization and extended simulated annealing techniques," *International Journal of Network Security*, vol. 6, no. 3, pp. 342–353, 2008.

[8] M. Ekera, "Differential cryptanalysis of md5," Master's thesis, School of Computer Science and Communication, Royal Institute of Technology, KTH, 2009.

[9] S. Konjicija, *Heuristički Algoritmi*. Sarajevo, BiH: Elektrotehnički fakultet Sarajevo, 2013.

[10] (2018, dec) English letter frequencies. [Online]. Available: http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/english-letter-frequencies/