**Алгоритм Particle Swarm Optimization.**

Автори реалізації алгоритму, що розглядається в описі: MUSHEER AHMAD, ISHFAQ AHMAD KHAJA, ABDULLAH BAZ, HOSAM ALHAKAMI, AND WAJDI ALHAKAMI.

Метод рою часток - це евристичний метод глобальної оптимізації, який реалізовано на основі популяцій. Запропоновано даний метод Кеннеді та Еберхартом у 1995 році. PSO базується на уявленні про «ройовий» інтелект (поведінка зграї птахів, риб, комах, тощо). Птахи або розлітаються, або навпаки летять разом під час пошуку їжі, перш ніж вони знайдуть гарне місце, де ця їжа знаходитиметься. Під час пошуку їжі завжди є один птах, який відчуває та шукає їжу краще за інших, тобто такий птах з більшою ймовірністю знайде місце, де може знаходитися їжа, а це в свою чергу означає, що він матиме кращу інформацію про джерело їжі ніж інші. Оскільки птахи в процесі пошуку постійно діляться такою «гарною» інформацію, зграя зрештою переміститься до «кращого» місця, де з більшою ймовірністю можна знайти їжу. У PSO пересування «птахів», починається з одного місцезнаходження на інше, еквівалентно зграї, хороша інформація – еквівалентна найбільш оптимістичному рішенню, а передбачувана їжа еквівалентна найбільш оптимістичному рішенню за весь час роботи алгоритму.

Даний метод допомагає знайти найбільш оптимістичне рішення саме завдяки співпраці кожного елементу популяції, «птиці». Завдяки алгоритму можуть бути вирішені доволі складні проблеми оптимізації. Переваги дають змогу застосовувати його до багатьох областей оптимізації окремо та в поєднанні з іншими існуючими алгоритмами. Алгоритм використовується в таких галузях, як навчання нейронних мереж, оптимізація певних функцій, машинне навчання, обробка сигналів, тощо.

**Опис алгоритму Particle Swarm Optimization**

У алгоритмі Particle Swarm Optimization популяція складається з «N» часток, а розташування кожної з часток відповідає потенційному рішенню в d-розмірному просторі. На положення кожної частинки у рої впливає як найбільш оптимістична позиція під час її руху (індивідуальний досвід, що називається особистим кращим чи  частинки), так і положення найбільш оптимальної частинки в її сусідстві (близький досвід, що називається найкращим серед усіх або). Частинки зграї летять у пошуковий простір завдяки своїм можливостям розвідки та експлуатації та використовують найкращі особисті та найкращі позиції у світі, щоб досягти найкращого рішення в PSO. Крім того, кожна частинка характеризується швидкістю. Швидкість і положення кожної частинки переглядаються після кожної наступної ітерації алгоритму. Швидкість і положення кожної частинки визначаються на кожному кроці як:





де,  та  це швидкість та позиція частки «» на її «» кроці відповідно та -розмірне значення її позиції.  представляє собою -розмірне значення кожного «» елемента у його найбільш оптимістичній позиції.  це -розмірне значення найбільш оптимістичної позиції для всього «рою». Параметри  генеруються випадковим чином у межах .

Для вирішення задачі генерації S-Box’ів з гарними показниками надійності було обрано саме такий метод з наступних причин:

1. PSO може бути застосований як для наукового, так і для інженерного використання, оскільки він базується на певному інтелекті.
2. Обчислення, що виконуються в PSO є доволі нескладними.
3. Метод передбачає зменшення кількості параметрів для налаштування та прийняття обмежень у порівнянні з іншими методами оптимізації.

**Параметри захищеності S-Box’ів**

Алгоритм, описаний у статті, що розглядається, передбачає порівняння захищеності різних згенерованих S-Box’ів на основі розрахунку та порівняння їхньої нелінійності, а саме: чим більшою є нелінійність блоку, тим більш надійним та захищеним він є. Це не завжди є правильним твердженням, оскільки на захищеність та надійність S-Box’ів впливає безліч параметрів, таких як: нелінійність, -рівномірність, максимум DDT (таблиці диференціалів), максимум LAT (таблиці лінійних апроксимацій), циклічна структура, наявність фіксованих точок, алгебраїчний імунітет, лінійна збитковість, тощо. Але оскільки для нас в алгоритмі важлива саме швидкість, можна допустити пошук на основі нелінійності.

1. Нелінійність

У статті пропонується обчислювати нелінійність блоків за допомогою перетворення Уолдша-Адамара наступним чином:



У нашому варіанті зміненої реалізації нелінійність блоків обчислюється через максимум таблиці лінійних апроксимацій () наступним чином:

* Будується таблиця лінійних апроксимацій. Кожен елемент у таблиці представляє кількість збігів між лінійним рівнянням, представленим у шістнадцятковій формі як "Вхідна сума", та сумою вихідних бітів, представлених у шістнадцятковій системі у вигляді "Вихідної суми" – 8 (для S-Box’ів розмірності ).
* Знаходиться абсолютний максимум таблиці.
* Обчислюється нелінійність за формулою:



де  - розмірність S-Box’у.

**Алгоритм**

У даному розділі описується запропонований у статті алгоритм генерації вузлів заміни за допомогою методу Particle Swarm Optimization, а також пропонується та описується модифікований алгоритм, який дає значно кращі результати, ніж ті, що надаються в статті.

Опис алгоритму зі статті:

* Ініціалізація популяції

У випадку задачі оптимізації S-box’ів, кожен блок  приймається за частку. Популяція S-box’ів генерується випадковим методом, таким чином, щоб S-Box’и залишалися бієктивними. Генерація повторюється доти, доки популяція розміру  не буде заповнено.

* Обчислення нелінійності

Далі для кожного блоку обчислюється нелінійність. Згідно з цією нелінійністю сортується сама популяція блоків (за спаданням нелінійності).

* Ініціалізація вектору PSO

Вектор швидкості заповнюється нулями і оновлюється на кожній вдалій ітерації. Ініціалізація кожного вектору розташування відбувається з використанням відповідного S-Box’у в популяції. Вектор швидкості оновлюється за формулою (1.1), а вектор розташування оновлюється за формулою (1.2). Вектор найбільш оптимістичних позицій під час виконання ітерації () оновлюється для кожної згенерованої популяції, якщо значення нелінійності нових блоків є кращими за попередні. Найбільш оптимальною з усіх є частинка з найбільшою нелінійністю у популяції.

* Ініціалізація параметрів PSO

Параметри PSO, такі як ,, та  обираються випадковим чином за допомогою вибірки Рені. Протягом оптимізаційної фази роботи алгоритму ці параметри випадково змінюються на кожній ітерації. Ще одним параметром є коефіцієнт інерційності (інерційна вага), який задається за формулою:



де  та  є початковим та кінцевим значенням коефіцієнту відповідно.

* Покращення та регулювання

Вектори швидкості та розташування оновлюються згідно з законами в формулах (1.1) та (1.2) відповідно. Процес такого покращення генерує певні значення, які повторюються або ж від’ємні значення. Щоб цьому зарадити, автори використовують певні методи та процеси обробки, замінюючи повторні значення, значеннями, яких не вистачає для збереження бієктивності. Алгоритм таких покращень у статті не надано.

* Фінальний крок ітерації

Далі для всіх нових згенерованих блоків також обчислюється значення нелінійності, всі S-Box’и, включаючи ті, що вже знаходилися в популяції, знову сортуються за спаданням нелінійності. В популяції залишаються  кращих за нелінійністю S-Box’ів, всі інші блоки відкидаються. Вектори  та  оновлюються як вказано вище.

Псевдокод алгоритму, представленого у статті, показано далі:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вхідні значення:

N ← number of populations

max\_itr ← maximum number of iterations

xr ← initial value of Renyi chaotic map

c ← parameter of Renyi map

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Generate initial population of S-boxes as:

xr ← renyi\_map(xr, c, 100)

population ← zeros(2 × N, 256) // 256 для 8 × 8 S-box’ів

for i ← 1 to N

[sboxi, xr] ← gen\_sbox(xr, c)

population[i] ← sboxi

end for

population[1] ← aes\_sbox

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Обчислення нелінійності для кожної з часток:

for i ← 1 to N do:

NL[i] ← nonlinearity(population[i])

end for

NL\_sorted ← sort(NL) // сортування у порядку спадання

gBest ← population[1]

pBesti ← population[i]

Vel ← zeros(N, 256)

Встановлення інерційної ваги w

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Початок фази покращення:

While (max\_itr > 0) do:

xr ← renyi\_map(xr, c); c1 ← 2∗xr

xr ← renyi\_map(xr, c); c2 ← 2∗xr

xr ← renyi\_map(xr, c); r1 ← xr

xr ← renyi\_map(xr, c); r2 ← xr

NL ← NL\_sorted

for i ← 1 to N do:

for j = 1 to 256 do:

Vel[i][j] ← ceil(w∗Vel[i][j] + c1∗r1∗(pBest[i][j] - population[i][j]) + c2∗r2∗(gBest[j]-population[i][j]))

if (Vel[i][j] < 0)

Vel[i][j] ← (Vel[i][j] + 256)mod(256)

end if

X[i, j] ← int(population[i][j] + Vel[i][j])mod(256)

temp\_sbox[j] ← X[i, j]

end for

Застосування певного алгоритму регулювання для збереження бієктивності temp\_sbox’у

population[N + i] ← temp\_sbox

end for

for i ← 1 to N do:

NL\_sorted[N + i] ← nonlinearity(population[N + i])

end for

NL\_sorted ← sort(NL\_sorted)

Відкидання зайвих у популяції елементів згідно з нелінійністю

for i ← 1 to N do:

if (NL[i] < NL\_sorted[i])

pBest[i] ← population[i]

end if

end for

Зміна gBest, якщо такий знайдено

max\_itr ← max\_itr - 1

end while

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_