Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп'ютерних наук

Спеціальність 125 «Кібербезпека»

Освітня програма «Безпека інформаційних та комунікаційних систем»

«Допущено до захисту»

Зав.кафедрою БІСТ

Сергій РАССОМАХІН

_____

« » 2022 р.

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Моделі та методи генерації нелінійних підстановок на основі ройового інтелекту»

оцінка « »

Голова ЕК

Доценко С.І. _____

Керівник проф. Горбенко І.Д. _____

Рецензент к.т.н. Бобух В.А. _____

Виконавець : студент групи КБ-61

_____ Дерев'янко Я.А. _____

Харків – 2022

РЕФЕРАТ

Пояснювальна записка містить: 74 сторінки, 50 рисунків, 14 таблиць, 31 джерело посилань, 3 додатки.

Об’єкт дослідження – процеси та алгоритми генерування нелінійних таблиць підстановки (блоків заміни) для симетричних криптографічних перетворень.

Предмет досліджень – методи генерації нелінійних вузлів заміни (S-блоків) для сучасних симетричних криптографічних перетворень, засновані на концепції ройового інтелекту, а саме метод Particle Swarm Optimization.

Метою кваліфікаційної роботи магістра є:

- аналіз різноманітних показників ефективності нелінійних вузлів замін;
- дослідження концепції ройового інтелекту та можливості його застосування в області генерування і покращення нелінійних підстановок;
- реалізація можливості представлення простору перестановок у вигляді одновимірного числового простору та можливості руху у такому просторі, а також дослідження таким чином показників ефективності вузлів замін що межують з нелінійним перетворенням (блоком заміни) шифру AES;
- аналіз та реалізація різних «функцій вартості» (cost-functions) для нелінійних вузлів замін з метою покращення процесу фільтрації блоків;
- розробка та реалізація ефективних методів генерації нелінійних вузлів замін з заданими показниками на основі ройового інтелекту;
- експериментальні дослідження розроблених методів, аналіз їх ефективності та обґрунтування отриманих результатів.

Методи дослідження: аналіз існуючих наукових досліджень у даній галузі, аналіз математики методів ройового інтелекту, розробка програмних застосунків на мовах C та C++, аналіз, систематизація та обґрунтування отриманих результатів.

Ключові слова: НЕЛІНІЙНІ ПІДСТАНОВКИ, S-БЛОКИ, ЕВРИСТИЧНІ МЕТОДИ, PARTICLE SWARM OPTIMIZATION, ФАКТОРІАЛЬНА СИСТЕМА ЧИСЛЕННЯ, ЦІНОВІ ФУНКЦІЇ, СИМЕТРИЧНА КРИПТОГРАФІЯ.

ABSTRACT

The explanatory note contains: 74 pages, 50 figures, 14 tables, 31 sources, 3 appendixes.

The object of research – processes and algorithms of generating non-linear substitutions (S-boxes) for symmetric cryptographic transformations.

The subject of research – methods of generating non-linear substitutions (S-boxes) for modern symmetric cryptographic transformations based on the concept of swarm intelligence, and the Particle Swarm Optimization method in particular.

The purposes of the qualifying work of the master are:

- analysis of various efficiency indicators of non-linear substitutions;
- research of the concept of swarm intelligence and the possibility of its application in the field of generating and improvement non-linear substitutions;
- implementation of the possibility of representing the space of permutations in the form of a one-dimensional numerical space and possibility of moving in such space and also researching the efficiency indicators of substitutions next to the AES substitution box in such way;
- analysis and implementation of various cost-functions for non-linear substitutions in order to improve the process of the block filtering;
- development and implementation of effective methods of generating non-linear substitutions with given parameters based on swarm intelligence;
- experimental researches of the developed methods, analysis of their efficiency and substantiation of the obtained results.

Research methods: analysis of existing scientific research in this field, analysis of the mathematics of swarm intelligence methods, development of software applications in C and C++ languages, analysis, systematization and substantiation of the obtained results.

Key words: NON-LINEAR SUBSTITUTIONS, S-BOXES, HEURISTIC METHODS, PARTICLE SWARM OPTIMIZATION, FACTORIAL NUMBER SYSTEM, COST FUNCTIONS, SYMMETRIC CRYPTOGRAPHY.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ УМОВНИХ ПОЗНАК, ОДИНИЦЬ І ТЕРМІНІВ

БСШ - Блоковий симетричний шифр
S-box - Substitutional Box (table)
ФСЧ - Факторіальна система числення
PSO - Partical Swarm Optimization
NL - Non-linearity
LAT - Linear Approximation Table
DDT - Differential Distribution Table
WH - Walsh–Hadamard
pBest - Population’s best
gBest - Global best
AES - Advanced Encryption Standard
NTL - Number Theory Library
WHS - Walsh-Hadamard spectrum
WCF - Walsh-Hadamard cost function
PCF - Picek cost function

ВСТУП

Сучасний стан розвитку інформаційних та інформаційно-комунікаційних систем викликає необхідність забезпечення найвищого рівня захищеності інформації. Основним і найбільш ефективним з точки зору універсальності механізмом захисту на сьогодні є методи блокового симетричного шифрування. При цьому одним із найважливіших компонентів сучасних блокових симетричних криптографічних перетворень є нелінійні вузли заміни або нелінійні підстановки. Їхня захищеність, стійкість та надійність має прямо пропорційний вплив на ефективність механізмів захисту, в яких вони застосовуються.

Дослідження бієктивних вузлів заміни є найбільш актуальним напрямом, оскільки такі блоки застосовуються у більшості БСШ, як застарілих, так і сучасних. У роботі буде детально розглянуто основні криптографічні показники, що впливають на захищеність нелінійних вузлів заміни, а також продемонстровано розроблений застосунок для їх швидкого знаходження.

З огляду на важливість використання у шифрах S-блоків з найбільш хорошими показниками, набуває значної актуальності задача генерації (синтезу) таких блоків. Для генерації криптографічно стійких підстановок використовуються різні методи: методи випадкової генерації, евристичні методи та алгебраїчні методи. Кожен з типів методів заснований на певних математичних принципах, випадковості, евристиці, тощо. При цьому кожен з них має як певні переваги, так і недоліки, які будуть властивими тільки для цього конкретного типу методів.

У кваліфікаційній роботі магістра досліджується обчислювальний метод оптимізації рою часток. Такий метод є евристичним методом глобальної оптимізації на основі популяцій. Particle Swarm Optimization допомагає знайти найбільш оптимістичне рішення саме завдяки співпраці кожного елементу популяції. У роботі надається детальне пояснення принципів застосування даного методу при роботі з нелінійними підстановками, а також при задачі синтезу таких підстановок.

Особливістю методу рою часток є те, що при виконанні задачі пошуку або синтезу S-боксів з високими показниками, необхідно швидко переміщатися між підстановками та мати можливість отримання значення необхідної підстановки, знаючи лише її номер у просторі пошуку. Для вирішення цієї задачі було обрано застосування факторіальної системи числення (ФСЧ). У роботі буде розглянуто деталі її застосування, а також надано можливий спосіб її використання разом із методом PSO і при дослідженні простору нелінійних підстановок. Також для оптимізації роботи пошукових алгоритмів у роботі пропонується використання цінових функцій.

Отже, метою кваліфікаційної роботи є дослідження моделей та методів ройового інтелекту, аналіз можливості їх застосування для генерації нелінійних підстановок, реалізація моделей та методів генерації нелінійних вузлів замін, які були б здатні забезпечувати необхідний рівень стійкості сучасних симетричних криптоперетворень.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ГЕНЕРАЦІЇ НЕЛІНІЙНИХ ВУЗЛІВ ЗАМІН

З метою забезпечення необхідного рівня захищеності інформації в сучасних інформаційних та інформаційно-комунікаційних системах застосовують різні криптографічні засоби. Методи блокового шифрування, а саме блокового симетричного криптографічного перетворення, на сьогодні є основним і найбільш ефективним механізмом криптографічного захисту інформації. Зумовлено це тим, що такі криптоперетворення спроможні забезпечити досить високу стійкість до різноманітних методів криптоаналізу, а також відрізняються швидкістю виконання перетворень та простотою практичної реалізації [1].

Одним із основних компонентів сучасних блокових симетричних криптографічних перетворень є нелінійні вузли заміни. Стійкість та надійність цих вузлів має безпосередній вплив на ефективність механізмів захисту, в яких вони застосовуються.

Вузол заміни (S-box [2-4], S-блок, Substitution Box чи Блок заміни) є відображенням n вхідних бітів у m вихідних бітів: $S-box:\{0,1\}^n \to \{0,1\}^m$. Отже, S-box представляється у вигляді множини окремих m вихідних булевих функцій, що об’єднані в заданому порядку. Вузол заміни має 2^n, а також 2^m можливих виходів. Зазвичай S-блоки представляються у вигляді таблиці, де входу відповідає певний вихід. Коли для S-блоку всі вихідні значення є різними, такий блок є ін’єктивним. Коли у S-блоці представлені всі можливі виходи, такий блок є сюр’єктивним. Якщо ж S-box ін’єктивний та сюр’єктивний одночасно, такий S-box називається бієктивним або ж збалансованим. Для бієктивного S-блоку завжди виконується умова $n=m$.

Дослідження саме бієктивних вузлів заміни є найбільш актуальним напрямом, оскільки такі блоки застосовуються у більшості БСШ, як застарілих, так і сучасних.

1.1 Критерії та показники стійкості нелінійних вузлів замін

До основних актуальних криптографічних показників нелінійних вузлів заміни можна віднести [2-4]:

- Нелінійність;
- Збалансованість;
- Максимум таблиці лінійних апроксимацій ($LAT_{\max}$);
- Дельта-рівномірність;
- Алгебраїчний ступінь;
- Алгебраїчний імунітет;
- Лінійна надмірність, тощо.

Існують і інші показники, проте при відборі вузлів, що будуть використовуватися у блокових симетричних шифрах, ці показники не мають значного впливу на ефективність.

Розглянемо більш детально деякі з представлених показників:

Нелінійність [2,4] булевої функції (функції від n змінних) є мінімальною відстанню Хеммінга від цієї функції до множини всіх афінних функцій від n змінних. Тобто, нелінійність булевої функції буде обчислюватися наступним чином:

$$NL(f) = \frac{1}{2}(2^n - WHT_{\max}) \qquad (1.1)$$

Щодо S-блоку, його нелінійність буде визначатися як мінімальна нелінійність серед множини нелінійностей координатних булевих функцій цього блоку, а також серед всіх лінійних комбінацій цих функцій. Показник нелінійності блоку вважається тим краще, чим більшим він є.

S-box буде збалансованим [2-4] (бієктивним), якщо всі координатні булеві функції цього блоку та їхні лінійні комбінації будуть збалансованими. Тобто, якщо число нулів в таблиці істинності даних функцій буде дорівнювати числу одиниць.

Максимум таблиці лінійних апроксимацій [3] ($LAT_{\max}$), для S-блоку буде обчислюватися наступним чином:

$$\mathrm{LAT}(\alpha,\beta)=\left\{X\in Z_2^n \mid \alpha\cdot X=\beta\cdot S(X)\right\}-2^{n-1} \quad (1.2)$$

Знаючи цей показник, можна легко вирахувати нелінійність блоку із співвідношення:

$$NL=2^{n-1}-LAT_{\max} \quad (1.3)$$

Алгебраїчний ступінь [2-4] S-блоку буде визначатися як мінімальний ступінь серед множини координатних булевих функцій цього блоку, а також серед всіх лінійних комбінацій цих функцій.

Алгебраїчний імунітет S-блоку – це показник захищеності вузла щодо алгебраїчного аналізу, добре захищеними вважаються блоки з імунітетом >2.

Щодо лінійної надмірності, вважається, що блок є не має лінійної надмірності, якщо всі його координатні булеві функції не є еквівалентними афінно. В іншому разі блок є лінійно надмірним.

1.2 Програмна реалізація визначення показників ефективності нелінійних вузлів заміни

При виконанні поставлених задач, було розроблено застосунок для визначення показників ефективності та захищеності нелінійних вузлів заміни на мові програмування С.

Даний застосунок дозволяє отримати значення наступних параметрів S-блоку:

- Дельта-рівномірність;
- Максимум таблиці лінійних апроксимацій ($LAT_{\max}$);
- Нелінійність;
- Циклічну структуру;
- Кількість фіксованих точок;
- Алгебраїчний ступінь;
- Алгебраїчний імунітет.

Застосунок дозволяє обчислити показники блоку представленого як у десятковому форматі, так і у шістнадцятирічному.

Для використання застосунку, у папці з ним іде інструкція із застосування, в якій описується алгоритм дій для користування застосунком:

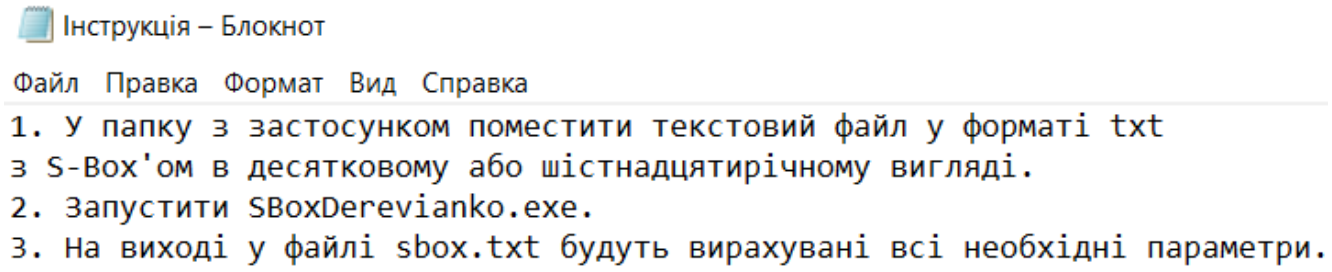

Рисунок 1.1 – Інструкція до застосунку

Для роботи із застосунком у папку необхідно помістити S-box у десятковому або шістнадцятирічному вигляді. Приклади таких блоків показано на рисунках 1.2 та 1.6:

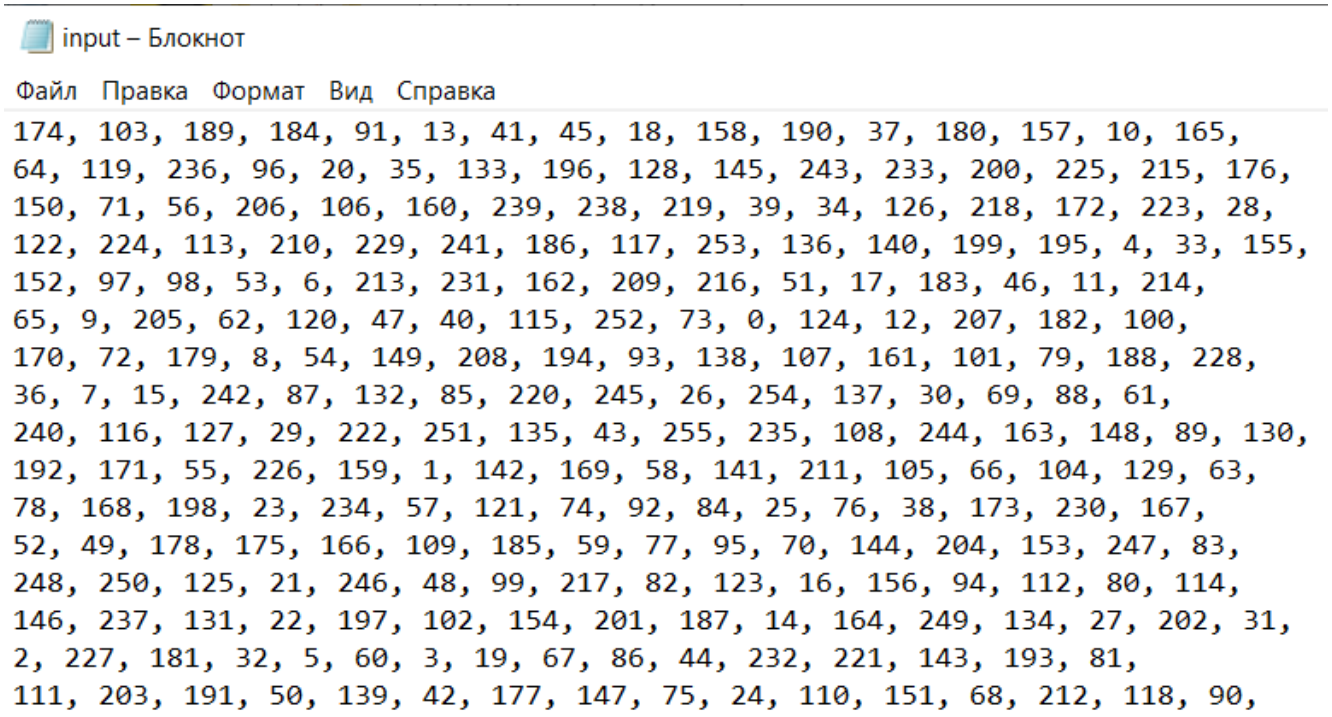

Рисунок 1.2 – Приклад певного блоку у десятковому форматі

Далі, після розміщення блоку у текстовому файлі до папки із застосунком маємо наступне:

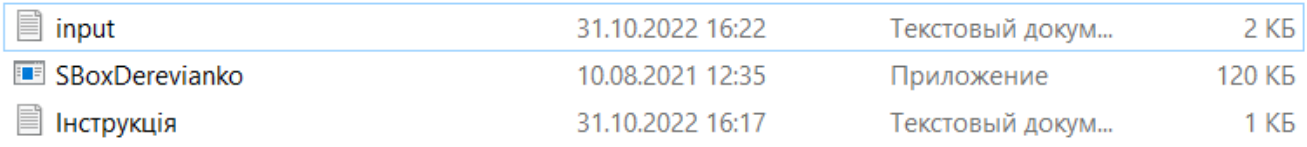

Рисунок 1.3 – Папка із застосунком до запуску

Тепер згідно інструкції запускаємо застосунок:

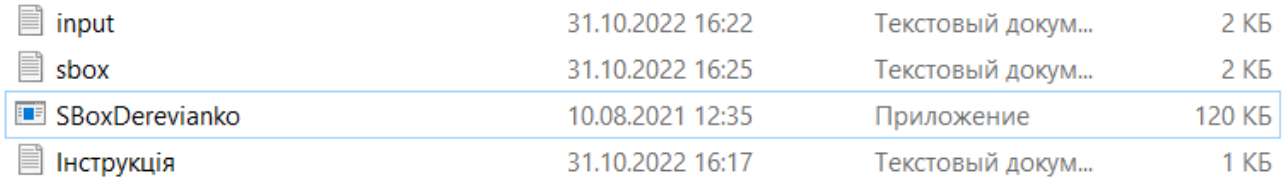

Рисунок 1.4 – Папка із застосунком після запуску

Бачимо, що після роботи застосунку у папці з’явився відповідний файл з обчисленими параметрами, необхідними нам. Подивимося вміст файлу:

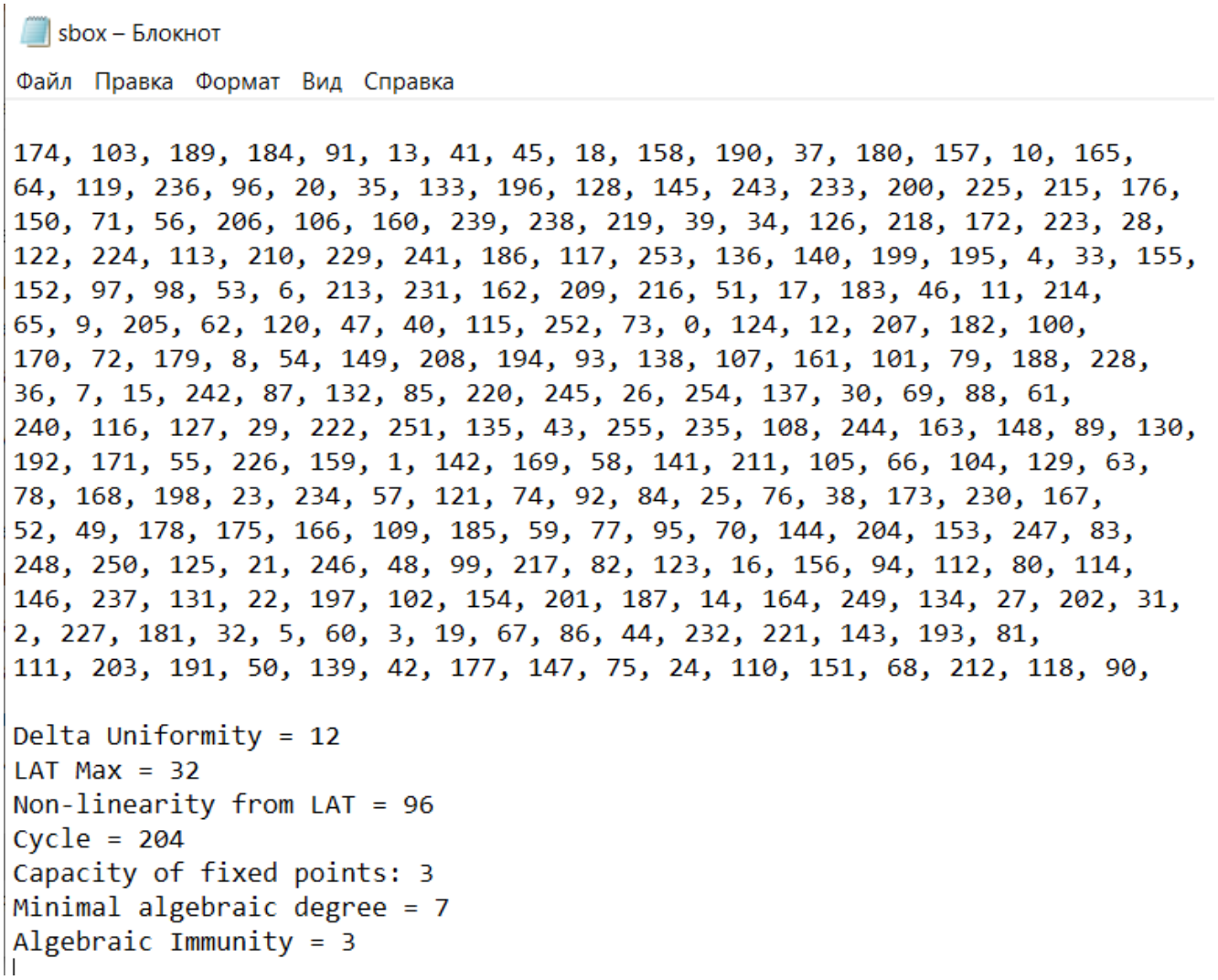

Рисунок 1.5 – Файл з порахованими параметрами блоку

Далі для демонстрації можливості використання застосунку із вузлами заміни у шістнадцятирічному форматі, надається приклад вхідного блоку у вигляді набору шістнадцятирічних байтів. Такий метод обчислення є більш зручним, оскільки найчастіше S-блоки представляються саме таблицями або набором таких байтів:

input – Блокнот

Файл Правка Формат Вид Справка

```
0x03, 0x6A, 0x57, 0xA4, 0xA9, 0xF3, 0x70, 0xF1, 0x6D, 0x00, 0x80, 0x87, 0x5A, 0x10, 0x81, 0x2C,
0x1C, 0x22, 0x9D, 0x67, 0x23, 0x71, 0x8F, 0x43, 0xAC, 0x21, 0xD2, 0x68, 0x18, 0xDE, 0x98, 0x41,
0x17, 0x69, 0x33, 0xC3, 0xCC, 0xA0, 0x4A, 0xE0, 0xB3, 0xEF, 0xDA, 0xD7, 0xC5, 0x55, 0x38, 0x29,
0x1B, 0x1D, 0xC6, 0x63, 0xBA, 0x8D, 0x9B, 0x2F, 0x8C, 0x7C, 0xAA, 0x0D, 0xCE, 0x06, 0xB1, 0xAD,
0x92, 0x9A, 0xD6, 0xB8, 0xBB, 0xC0, 0xE3, 0x32, 0xFF, 0xC2, 0xE9, 0x2D, 0xBC, 0xE8, 0x09, 0x5F,
0x0B, 0xF9, 0xDF, 0x36, 0x0E, 0x9C, 0xED, 0x3D, 0x37, 0xCA, 0xA6, 0x75, 0x46, 0xA3, 0x79, 0x86,
0x0F, 0xE7, 0x97, 0xA5, 0xFA, 0x51, 0xD3, 0xD8, 0xE4, 0x30, 0xC4, 0xEE, 0x54, 0x96, 0x2E, 0xE2,
0x65, 0x90, 0x6C, 0x3A, 0x40, 0xFB, 0x25, 0x95, 0xB7, 0x28, 0xFC, 0x49, 0x66, 0xAE, 0x34, 0x08,
0xD0, 0x4D, 0xD4, 0xA7, 0xF2, 0x0A, 0xE5, 0x5C, 0x64, 0xE6, 0x62, 0x0C, 0x88, 0x01, 0x3B, 0xE1,
0xA1, 0x74, 0x45, 0xB2, 0xDB, 0x6B, 0x99, 0x56, 0x47, 0x8E, 0x73, 0xF6, 0x7D, 0xD5, 0xBE, 0x39,
0x07, 0x42, 0x85, 0x20, 0x76, 0x5E, 0xA2, 0x7A, 0x1E, 0x58, 0x4C, 0x94, 0x14, 0xF7, 0x27, 0xCD,
0xCB, 0x4F, 0x91, 0x82, 0x53, 0xD9, 0x1F, 0xC1, 0xDD, 0xB4, 0x02, 0x8A, 0xBF, 0x59, 0x89, 0xAF,
0x9E, 0x3C, 0x11, 0x8B, 0xC9, 0xEA, 0xF0, 0xB0, 0x04, 0x7E, 0x31, 0x05, 0xEB, 0x26, 0x6E, 0x50,
0x77, 0x44, 0x78, 0xC7, 0x83, 0xEC, 0xDC, 0x9F, 0xFD, 0xFE, 0x35, 0x1A, 0x5D, 0x61, 0xF5, 0xF4,
0x4B, 0x12, 0xD1, 0x52, 0xCF, 0xF8, 0x3F, 0x93, 0xB9, 0xAB, 0x84, 0x4E, 0x2B, 0xBD, 0xC8, 0x5B,
0x16, 0x60, 0xB5, 0x15, 0x24, 0x6F, 0x48, 0x72, 0x13, 0x7B, 0x19, 0xA8, 0x2A, 0x3E, 0x7F, 0xB6,
```

Рисунок 1.6 – Приклад певного блоку у шістнадцятирічному форматі

Запускаємо застосунок та переконуємося, що він працює і з блоками у шістнадцятирічному форматі:

sbox – Блокнот

Файл Правка Формат Вид Справка

```
0x03, 0x6A, 0x57, 0xA4, 0xA9, 0xF3, 0x70, 0xF1, 0x6D, 0x00, 0x80, 0x87, 0x5A, 0x10, 0x81, 0x2C,
0x1C, 0x22, 0x9D, 0x67, 0x23, 0x71, 0x8F, 0x43, 0xAC, 0x21, 0xD2, 0x68, 0x18, 0xDE, 0x98, 0x41,
0x17, 0x69, 0x33, 0xC3, 0xCC, 0xA0, 0x4A, 0xE0, 0xB3, 0xEF, 0xDA, 0xD7, 0xC5, 0x55, 0x38, 0x29,
0x1B, 0x1D, 0xC6, 0x63, 0xBA, 0x8D, 0x9B, 0x2F, 0x8C, 0x7C, 0xAA, 0x0D, 0xCE, 0x06, 0xB1, 0xAD,
0x92, 0x9A, 0xD6, 0xB8, 0xBB, 0xC0, 0xE3, 0x32, 0xFF, 0xC2, 0xE9, 0x2D, 0xBC, 0xE8, 0x09, 0x5F,
0x0B, 0xF9, 0xDF, 0x36, 0x0E, 0x9C, 0xED, 0x3D, 0x37, 0xCA, 0xA6, 0x75, 0x46, 0xA3, 0x79, 0x86,
0x0F, 0xE7, 0x97, 0xA5, 0xFA, 0x51, 0xD3, 0xD8, 0xE4, 0x30, 0xC4, 0xEE, 0x54, 0x96, 0x2E, 0xE2,
0x65, 0x90, 0x6C, 0x3A, 0x40, 0xFB, 0x25, 0x95, 0xB7, 0x28, 0xFC, 0x49, 0x66, 0xAE, 0x34, 0x08,
0xD0, 0x4D, 0xD4, 0xA7, 0xF2, 0x0A, 0xE5, 0x5C, 0x64, 0xE6, 0x62, 0x0C, 0x88, 0x01, 0x3B, 0xE1,
0xA1, 0x74, 0x45, 0xB2, 0xDB, 0x6B, 0x99, 0x56, 0x47, 0x8E, 0x73, 0xF6, 0x7D, 0xD5, 0xBE, 0x39,
0x07, 0x42, 0x85, 0x20, 0x76, 0x5E, 0xA2, 0x7A, 0x1E, 0x58, 0x4C, 0x94, 0x14, 0xF7, 0x27, 0xCD,
0xCB, 0x4F, 0x91, 0x82, 0x53, 0xD9, 0x1F, 0xC1, 0xDD, 0xB4, 0x02, 0x8A, 0xBF, 0x59, 0x89, 0xAF,
0x9E, 0x3C, 0x11, 0x8B, 0xC9, 0xEA, 0xF0, 0xB0, 0x04, 0x7E, 0x31, 0x05, 0xEB, 0x26, 0x6E, 0x50,
0x77, 0x44, 0x78, 0xC7, 0x83, 0xEC, 0xDC, 0x9F, 0xFD, 0xFE, 0x35, 0x1A, 0x5D, 0x61, 0xF5, 0xF4,
0x4B, 0x12, 0xD1, 0x52, 0xCF, 0xF8, 0x3F, 0x93, 0xB9, 0xAB, 0x84, 0x4E, 0x2B, 0xBD, 0xC8, 0x5B,
0x16, 0x60, 0xB5, 0x15, 0x24, 0x6F, 0x48, 0x72, 0x13, 0x7B, 0x19, 0xA8, 0x2A, 0x3E, 0x7F, 0xB6,

Delta Uniformity = 6
LAT Max = 28
Non-linearity from LAT = 100
Cycle = 183
Fixed points ain't found
Minimal algebraic degree = 7
Algebraic Immunity = 3
```

Рисунок 1.7 – Файл з порахованими параметрами блоку

Як видно з наведених рисунків, застосунок добре виконує всі поставлені перед ним задачі.

1.3 Класифікація методів генерації S-блоків

Існуючі методи генерації нелінійних блоків замін поділяються на три типи: випадкової генерації [5], евристичні [6] та алгебраїчні [7].

Перший тип методів – випадкової генерації – заснований на процедурі випадкового синтезу блоків і подальшого відбору кандидатів, що відповідатимуть необхідним критеріям чи параметрам [5]. Основною перевагою таких методів є простота реалізації і рандомність отриманих блоків, що зумовлює відсутність збитковості та забезпечує складність алгебраїчного представлення. Головним недоліком таких методів є значне зниження ефективності синтезу зі збільшенням розмірності блоків, що синтезуються.

Другий тип – алгебраїчні методи [7]. Перевагою даного типу є низька складність обчислення. Це дозволяє генерувати блоки довільної розмірності з помірною обчислювальною складністю. Ще однією перевагою генерації з використанням такого типу методів є досягнення хороших основних параметрів криптографічної стійкості та ефективності – низької автокореляції та високої нелінійності. До недоліків таких методів можна віднести нерандомність сформованих блоків, і, як наслідок, присутню в них лінійну надмірність. Також блоки, отримані таким чином, є доволі вразливими до алгебраїчного криптоаналізу. Слід також зауважити, що алгебраїчні методи мають певні обмеження щодо кількості блоків, які можуть бути отримані шляхом їхнього використання.

Третім типом методів є евристичні методи. Основою евристичних методів є ітеративний підхід до генерації – тобто на кожній ітерації певного циклу генерації, такий метод намагається градієнтно покращити певний заданий показник стійкості, використовуючи для цього певні евристики [6]. Наприклад, метод може намагатися робити покращення, спираючись на якесь заздалегідь відоме гарне значення і на значення, які уже відвідував метод у спробах покращення, але залишив з метою знайти те, яке б відповідало заданим критеріям найкращим чином. Евристичним методам притаманні такі ж переваги, що і для методів, заснованих на випадковій генерації. І, хоча, евристичні методи все ж мають більшу ефективність, ніж методи

випадкової генерації, такі методи все ще потерпають від зниження ефективності генерації зі збільшенням розмірності блоків, що генеруються.

У таблиці 1.1 буде наведено класифікацію методів генерації нелінійних блоків заміни:

Таблиця 1.1 – Класифікація методів генерації нелінійних блоків заміни

Методи випадкової генерації	Алгебраїчні	Евристичні
Методи випадкової генерації з використанням фільтрації	Ступеневе відображення в полі	Метод Particle Swarm Optimization
Побітові методи (методи відбору функцій)	Інверсія в полі з використанням афінного перетворення	Метод Simulated Annealing
		Genetic Algorithms
		Метод Hill Climbing
		Метод Диференційної Еволюції

Знаючи, переваги та недоліки існуючих методів, можемо зробити порівняння ефективності методів [5-7], за максимальною нелінійністю, яку вдалося отримати кожним з типів методів. Порівняння надається в таблиці 1.2 [5-7]:

Таблиця 1.2 – Порівняння ефективності різних методів

Метод	Розмірність	Максимальна нелінійність
Методи випадкової генерації	8×8	98
Алгебраїчні методи	8×8	112

Продовження таблиці 1.2 – Порівняння ефективності різних методів

Метод	Розмірність	Максимальна нелінійність
Метод Hill Climbing	8×8	100
Genetic Algorithms	8×8	104
Метод Диференційної Еволюції	8×8	98
Метод Particle Swarm Optimization	8×8	98

Як можна бачити з даних у таблиці, найбільш ефективними з точки зору нелінійності сформованого блоку є алгебраїчні методи [7]. Проте такі методи, навідміну від методів випадкової генерації та евристичних методів, не здатні генерувати повністю випадкові блоки, що є навіть більшою потенційною загрозою для алгоритму, в якому такий блок буде використовуватися, ніж відставання блоку за показником нелінійності.

Дослідивши нелінійні вузли заміни, що використовуються у сучасних БСШ, а також природу їх генерації, можна сказати, що блоки, які мають максимально можливу нелінійність завжди згенеровані алгебраїчними методами, проте інші параметри таких блоків є гіршими, ніж у блоків, отриманих шляхом випадкових чи евристичних методів. Проте, не існує жодного алгоритму, який міг би згенерувати блок, в якому хорошими були б усі показники і тому, доводиться йти на компроміс, щоб задовольняти певні вимоги, при цьому жертвуючи показниками захищеності від інших типів криптоаналізу.

1.4 Обґрунтування вибору алгоритму для дослідження

Проведений у минулих пунктах аналіз, показує, що дослідження та подальша розробка ефективних та швидких методів генерації нелінійних вузлів заміни є перспективним напрямком досліджень, а також одним із найважливіших завдань при проектуванні сучасних симетричних криптоалгоритмів.

Зважаючи на це, було вирішено обрати основою для дослідження евристичні методи генерації блоків, а саме метод Particle Swarm Optimization [8]. Такий метод є евристичним методом глобальної оптимізації, який реалізовано на основі популяцій. PSO базується на уявленні про «ройовий» інтелект (поведінка зграї птахів, риб, комах, тощо). Даний метод допомагає знайти найбільш оптимістичне рішення саме завдяки співпраці кожного елементу популяції.

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 [8-9].

Застосування такого алгоритму до задачі генерації нелінійних вузлів замін показало, що він хоч і не здатний в теорії генерувати блоки з найкращою нелінійністю, проте може забезпечити відповідність інших показників ефективності вузлів заміни.

Також цей алгоритм є цікавим тим, що з його допомогою не тільки можна генерувати нові блоки, а і покращувати існуючі. Тому було вирішено реалізувати даний метод генерації і спробувати покращити його пошукові можливості у першій реалізації, а також забезпечити максимальну випадковість у другій реалізації.

1.5 Висновки до розділу

Нелінійні вузли заміни – один із основних компонентів сучасних блокових симетричних криптографічних перетворень. Їхні стійкість та надійність мають безпосередній вплив на ефективність механізмів захисту, в яких вони застосовуються.

До основних актуальних криптографічних показників нелінійних вузлів заміни можна віднести: Нелінійність, збалансованість, максимум таблиці лінійних апроксимацій, дельта-рівномірність, алгебраїчний ступінь, алгебраїчний імунітет, лінійну надмірність, тощо. Ці показники характеризують стійкість S-box’ів до лінійного, диференціального, алгебраїчного та інших методів криптоаналізу. У

розділі було представлено зразок програмного застосунку для обчислення цих показників для блоків у десятковому та шістнадцятирічному форматах.

Проведений аналіз показав, що найбільш ефективними з точки зору нелінійності сформованого блоку є алгебраїчні методи синтезу. Проте такі методи, навідміну від методів випадкової генерації та евристичних методів, не здатні генерувати повністю випадкові блоки, що є навіть більшою потенційною загрозою для алгоритму, в якому такий блок буде використовуватися, ніж відставання блоку за показником нелінійності.

Провівши аналіз сучасних БСШ, та їх вузлів заміни, можна сказати, що блоки, які мають максимально можливу нелінійність завжди згенеровані алгебраїчними методами, проте інші параметри таких блоків є гіршими, ніж у блоків, отриманих шляхом випадкових чи евристичних методів.

Зважаючи на це, було вирішено обрати основою для дослідження та подальшої розробки евристичні методи генерації блоків, а саме метод Particle Swarm Optimization. У наступному розділі буде детально описано одну із реалізацій даного методу, яка була розроблена в процесі виконання кваліфікаційної роботи магістра.

2 АЛГОРИТМ PARTICLE SWARM OPTIMIZATION

Ефективність симетричних криптографічних перетворень залежить від багатьох факторів [10]: базової структури, схеми, що використовується для розкладу ключів, показниками та параметрами стійкості криптопримітивів, тощо. Одним із найбільш значущих криптопримітивів, що є основою симетричної криптографії, є нелінійні вузли замін (S-boxes, нелінійні підстановки) [10-11]. Збалансованість, нелінійність, алгебраїчний імунітет, Δ-рівномірність, та інші параметри характеризують стійкість блоку (підстановки) до алгебраїчного, лінійного, диференційного та інших відомих методів криптоаналізу. Зважаючи на це, задача синтезу блоків з необхідними показниками є важливою науковою ціллю, яка у час стрімкого розвитку комп'ютерної техніки є як ніколи актуальною [10-12].

Як було показано в минулому розділі, для отримання криптографічно стійких підстановок використовуються різні методи: евристичні методи, алгебраїчні методи та методи випадкової генерації. Основним завдання роботи було обрано дослідження та реалізацію методу рою часток у застосуванні до задачі синтезу високоефективних нелінійних підстановок. Провівши аналіз відкритих наукових джерел, було виявлено, що в одній з нещодавніх робіт [13] пропонується використання методу рою часток (PSO). Однак результати, які наводяться в даній статті є недостовірними. Але все ж слід зазначити, що ідея використання PSO для генерації S-boxes може бути корисною [14-16]. Метою даного розділу є дослідження PSO для евристичної генерації нелінійних підстановок, експериментальна перевірка результатів та обґрунтування певних параметрів PSO для формування S-блоків з необхідними параметрами.

2.1 Опис алгоритму

Метод рою часток - це евристичний метод глобальної оптимізації, який реалізовано на основі популяцій. Запропоновано даний метод Кеннеді та Еберхартом у 1995 році [17]. 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 [8-9].

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 $pBest$ частинки), так і положення найбільш оптимальної частинки в її сусідстві (близький досвід, що називається найкращим серед усіх або $gBest$) [13]. Частинки зграї рухаються у пошуковому просторі завдяки своїм можливостям розвідки та експлуатації та використовують найкращі особисті та найкращі позиції у загальному відомому просторі, щоб досягти найкращого рішення в PSO. Крім того, кожна частинка характеризується швидкістю. Швидкість і положення кожної частинки переглядаються після кожної наступної ітерації алгоритму. Швидкість (2.1) і положення (2.2) кожної частинки визначаються на кожному кроці як [13]:

$$v_{id}^{k+1} = v_{id}^{k} + c_1 r_1 (pBest_{id}^{k} - x_{id}^{k}) + c_2 r_2 (gBest_{id}^{k} - x_{id}^{k}) \qquad (2.1)$$

$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1} \quad (2.2)$$

де, v_{id}^{k} та x_{id}^{k} це швидкість та позиція частки «i» на її «k» кроці відповідно та d-розмірне значення її позиції. $pBest_{id}^{k}$ представляє собою d-розмірне значення кожного «i» елемента у його найбільш оптимістичній позиції. $gBest_{id}^{k}$ це d-розмірне значення найбільш оптимістичної позиції для всього «рою». Параметри c_1, c_2, r_1, r_2 генеруються випадковим чином у межах [0,1].

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

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

2.1.1 Параметри захищеності S-Box'ів, що будуть застосовуватися

Алгоритм, описаний у статті, що розглядається [13], передбачає порівняння захищеності різних згенерованих S-блоків 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 блоків впливає безліч параметрів, таких як: нелінійність, Δ -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.

У статті пропонується обчислювати нелінійність блоків за допомогою перетворення Уолдша-Адамара наступним чином [13]:

$$NL_f = \frac{1}{2}(2^n - WH_{\max}(f)) \quad (2.3)$$

У нашому варіанті [14] зміненої реалізації нелінійність блоків обчислюється через максимум таблиці лінійних апроксимацій ($LAT_{\max}$) 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-блоків розмірності 2^8).
- Знаходиться абсолютний максимум таблиці.
- Обчислюється нелінійність за формулою:

$$NL_{Sbox} = (2^{n-1} - LAT_{\max}) \quad (2.4)$$

де n - розмірність S-блоку (в нашому випадку 8, оскільки ми працюємо з блоками розмірності 8×8.

2.2 Концепції базового та модифікованого алгоритмів

У даному пункті будуть детально та покроково розглянуті основні процеси, які відбуваються в процесі роботи кожного з алгоритмів, а також відмінності між цими алгоритмами. Крім того, буде надано псевдокод кожного з алгоритмів.

2.2.1 Алгоритм взятий за основу дослідження

У даному пункті описується запропонований у статті [13] алгоритм генерації вузлів заміни за допомогою методу Particle Swarm Optimization. Роботу алгоритму можна умовно поділити на наступні фази:

- Ініціалізація популяції

У випадку задачі оптимізації S-блоків, кожен блок розмірності 8×8 приймається за частку. Популяція блоків генерується випадковим методом, таким чином, щоб блоки залишалися бієктивними. Генерація повторюється доти, доки популяція розміру N не буде заповнено. Першим блоком у популяції завжди є блок шифру AES [13].

- Обчислення нелінійності

Далі для кожного блоку обчислюється нелінійність. Згідно з цією нелінійністю сортується сама популяція блоків (за спаданням нелінійності) [13].

- Ініціалізація вектору PSO

Відбувається заповнення вектору швидкості нулями і ініціалізація вектору розташування для кожного відповідного S-блоку в початковій популяції.

Оновлення вектору швидкості відбувається за формулою (2.1), а оновлення вектору розташування за формулою (2.2). Вектор, в якому містяться найкращі позиції, отримані при виконанні конкретної ітерації (*pBest*), оновлюється для кожної нової популяції тоді, коли значення нелінійностей згенерованих блоків є вищими, ніж для попередньої популяції. Найбільш оптимальною або ж найкращою частинкою буде частинка, нелінійність якої має найбільше значення [13].

- Ініціалізація параметрів PSO

Параметри PSO – c_1, c_2, r_1, r_2 – вибираються випадково з використанням вибірки Рені. Під час виконання так званої фази покращення алгоритму вони зазнають змін випадковим чином на кожній ітерації. Ще одним параметром є коефіцієнт інерційності (вага), який задається за наступною формулою [13]:

$$w_{curIter} = w_1 + (curIter - 1)(\frac{w_2 - w_1}{maxIter}) \quad (2.5)$$

де w_1 та w_2 є мінімальним та максимальним значенням коефіцієнту відповідно.

- Покращення та регулювання

Оновлення векторів швидкості та розташування відбувається згідно з законами в формулах (2.1) та (2.2) відповідно. Процес оновлення зумовлює отримання певних повторних або від'ємних значень. Щоб цьому зарадити, автори використовують певні методи та процеси обробки [13], замінюючи повторні значення, значеннями, яких не вистачає для збереження бієктивності. Алгоритм таких покращень у статті не надано.

- Фінальний крок ітерації

Для всіх синтезованих під час виконання ітерації блоків обчислюється значення нелінійності і відбувається сортування всіх блоків за спаданням нелінійності. Після цього останні блоки відкидаються, таким чином, щоб в популяції знову містилось N кращих за нелінійністю S-блоків, Відбувається оновлення *pBest* та *gBest* [13].

Псевдокод базового алгоритму PSO надається на наступних рисунках (рисунок 2.1 і рисунок 2.2)[13]:

```
Вхідні значення:
N ← число блоків в популяції
maxIter ← максимальна кількість ітерацій
xr ← початкове значення вибірки Рені
c ← параметр вибірки Рені

_____
Генерація початкової популяції S-box'ів:
xr ← renyi_map(xr, c, 100)
swarm ← zeros(2 x N, 256) // 256 для 8 x 8 S-box'ів
for i ← 1 to N
        [sbox, xr] ← gen_sbox(xr, c)
        swarm [i] ← sbox
end for
swarm [1] ← aes_sbox

_____
Обчислення нелінійності для кожної з часток:
for i ← 1 to N do:
NL[i] ← знаходження нелінійності (swarm [i])
end for
NL_відсортований ← sort(NL) // сортування у порядку спадання
gBest ← swarm [1]
pBesti ← swarm [i]
swarmVel ← zeros(N, 256)
Встановлення інерційної ваги w
```

Рисунок 2.1 – Псевдокод базового алгоритму PSO

```
_____
Початок фази покращення:
While (max_itr > 0) do:
        xr ← вибірка Рені(xr, c); c_1 ← 2*xr
        xr ← вибірка Рені (xr, c); c_2 ← 2*xr
        xr ← вибірка Рені (xr, c); r_1 ← xr
        xr ← вибірка Рені (xr, c); r_2 ← xr
        NL ← NL_відсортований
        for i ← 1 to N do:
                for j = 1 to 256 do:
                        swarmVel [i][j] ← ceil(w* swarmVel [i][j] + c_1*r_1*(pBest[i][j]
                        - swarm [i][j]) + c_2*r_2*(gBest[j]- swarm [i][j]))
                        if (swarmVel [i][j] < 0)
                                swarmVel [i][j] ← (swarmVel [i][j] + 256)mod(256)
                        end if
                        X[i, j] ← int(swarm [i][j] + swarmVel [i][j])mod(256)
                        temp_sbox[j] ← X[i, j]
                end for
                Застосування певного алгоритму регулювання для збереження бієктивності temp_sbox'у
                swarm [N + i] ← temp_sbox
        end for
        for i ← 1 to N do:
                NL_відсортований[N + i] ← нелінійність(swarm [N + i])
        end for
        NL_відсортований ← sort(NL_відсортований)
        Відкидання зайвих у популяції елементів згідно з нелінійністю
        for i ← 1 to N do:
                if (NL[i] < NL_відсортований[i])
                        pBest[i] ← swarm [i]
                end if
        end for
        Зміна gBest, якщо такий знайдено
        maxIter ← maxIter - 1
end while
```

Рисунок 2.2 – Продовження псевдокоду базового алгоритму PSO

2.2.2 Модифікований алгоритм

Представлена модифікація методу [14-15] є дуже схожою на запропонований алгоритм, головною різницею є перша ітерації циклу while під час отримання

першого блоку в оновленій популяції. Модифікація також впливає на заповнення векторів найкращих блоків. Опис модифікованого алгоритму надається далі:

- Ініціалізація популяції [14-16]

Як і в розглянутому раніше алгоритмі, кожен блок розмірності 8×8 вважається часткою. Популяція генерується випадковим чином із забезпеченням бієктивності сформованих блоків. Генерація відбувається до заповнення популяції розмірності N.

- Обчислення нелінійності [14-16]

Для кожного з отриманих блоків обчислюється значення нелінійності. Популяція відсортовується за її спаданням.

- Ініціалізація вектору PSO [14-16]

Відбувається заповнення вектору швидкості нулями і ініціалізація вектору розташування для кожного відповідного S-блоку в початковій популяції. Оновлення вектору швидкості відбувається за формулою (2.1), а оновлення вектору розташування за формулою (2.2).

Основна відмінність модифікованої версії алгоритму полягає в тому, що при отриманні нових блоків перший блок, що буде знаходитись у новій популяції, генерується шляхом взаємодії першого блоку, що знаходився в початковій популяції, із самим собою. Після цього відбувається перемішування невеликої кількості позицій. Це дає майже нульовий вектор швидкості на першій ітерації, що дозволяє поступово погіршувати показники блоку на наступних ітераціях. Якщо при використанні такого методу нелінійність є більшою за 98, то на наступних ітераціях відбувається додаткове перемішування S-блоку випадковим чином. Таке перемішування має на меті покращення таких параметрів, як дельта-рівномірність, лінійна збитковість, алгебраїчний імунітет, тощо. Покращення необхідне для досягнення компромісу або вирівнювання «хорошості» параметрів блоку, щоб алгоритм мав змогу отримувати не тільки блоки з одним дуже хорошим параметром, а і блоки, в яких всі параметри були б на достатньому рівні та задовольняли умовам захищеності. Вектор *pBest* оновлюється для кожної нової популяції тоді, коли значення нелінійностей згенерованих блоків є вищими, ніж

для попередньої популяції. У нашій модифікації оновлення найбільш оптимальної частинки відбувається на кожній ітерації – найкращий блок з *pBest* переходить також у *gBest*.

- Ініціалізація параметрів PSO [14-16]

Параметри PSO – c_1, c_2, r_1, r_2 – вибираються випадково. Під час виконання фази покращення (оптимізації) алгоритму вони зазнають змін випадковим чином на кожній ітерації, що покращує випадковість отримання нових блоків. Коефіцієнт інерційності (вага) задається за формулою (2.5).

- Покращення та регулювання [14-16]

Для забезпечення можливості збереження бієктивності S-блоків було розроблено спеціальний алгоритм. Після того, як блок зазнав змін, кожен елемент у ньому перевіряється на унікальність. Якщо перевірка якогось з елементів вказує на те, що такий елемент вже присутній в цьому блоці, тоді contains стає одиницею або true, і значення повторного елементу змінюється завдяки підсумовуванню цього елементу з випадковим значенням і взяття за модулем розмірності блоку. Програмний код алгоритму наведено нижче у вигляді рисунку:

```
for (int i = 0; i < N; ++i)
    for (int j = 0; j < size;)
        Оновлення значення за формулою (2.1)
        Отримання нового значення X для тимчасового S-блоку, формула (2.2)
        int contains;
        if (contains == 0)
            tempSbox[j] = X;
        end if
        else
            tempSbox[j] = myModulusDec((tempSbox[j] + rand()), 256);
        end else;
        contains = 0;
        for (int k = 0; k < j; ++k)
            if (tempSbox[k] == tempSbox[j])
                contains = 1;
                break;
            end if
        end for
        if (!contains)
            j++;
        end if
    end for
    ......
end for
```

Рисунок 2.3 – Алгоритм збереження бієктивності S-блоків

- Фінальний крок ітерації [14-16]

Для всіх синтезованих під час виконання ітерації блоків обчислюється значення нелінійності і відбувається сортування всіх блоків за спаданням нелінійності.Після цього останні блоки відкидаються, таким чином, щоб в популяції знову містилось N кращих за нелінійністю S-блоків, відбувається оновлення *pBest* та *gBest*.

Псевдокод модифікованого алгоритму PSO надається у додатку А [14-16]:

Така модифікація методу дозволяє за порівняно короткий час формувати блоки з нелінійністю 104, лінійною збитковістю 0 та алгебраїчним імунітетом 3.

2.3 Спрощений теоретичний приклад роботи алгоритмів

У даному пункті будуть розглянуті обидва алгоритми на зрозумілому спрощеному прикладі, щоб зробити процес роботи алгоритмів більш зрозумілим та наглядним.

2.3.1 Принцип роботи базового алгоритму

Візьмемо теоретичний S-Box (3,1,0,2) з найкращою можливою теоретичною нелінійністю, який буде взятий за основу, як AES-Sbox в алгоритмі [13]. Наприклад кількість блоків в популяції N - 4. Тому на початку виконання алгоритму буде згенеровано ще 3 випадкові блоки. Цими блоками буде заповнено масив популяції. Теоретична нелінійність буде вказуватися у відсотках від найкращої можливої нелінійності в популяції, тобто 100% - найкраща можлива нелінійність, і чим менше цей відсоток, тим гіршою є нелінійність блоку.

Маємо:

Таблиця 2.1 – Популяція на кроці ініціалізації

S-box	Нелінійність
(3,1,0,2)	100%
(2,1,3,0)	83%
(1,2,3,0)	85%
(2,3,1,0)	81%

Після генерації масив буде відсортовано у порядку спадання нелінійності. Отримуємо:

Таблиця 2.2 – Популяція після сортування

S-box	Нелінійність
(3,1,0,2)	100%
(1,2,3,0)	85%
(2,3,1,0)	83%
(2,1,3,0)	81%

Після цього заповнюються вектори *gBest* та *pBest*. До вектору *gBest* додається перший блок з масиву, тобто блок з найкращою нелінійністю, а до *pBest* всі інші блоки [13].

Маємо наступні масиви, в яких зберігаються елементи популяції, для подальшої фази пошуку та покращення, яка буде застосовуватися алгоритмом до цих теоретичних блоків:

Таблиця 2.3 – Масив *gBest*

S-box	Нелінійність
(3,1,0,2)	100%

Таблиця 2.4 – Масив *pBest*

S-box	Нелінійність
(1,2,3,0)	85%
(2,3,1,0)	83%
(2,1,3,0)	81%

Тепер відбувається ініціалізація інерційної ваги та починається фаза оптимізації. На цьому кроці спочатку генеруються 4 випадкові коефіцієнти, потім за формулою (2.5) змінюємо інерційну вагу та починаємо оновлювати значення вектору швидкості за формулою (2.1) [13]:

$$Vel[0][0] = ceil(w \cdot 0 + c1 \cdot r1 \cdot (1-3) + c2 \cdot r2 \cdot (3-3)) = c1 \cdot r1 \cdot (-2)$$

$$Vel[0][1] = ceil(w \cdot 0 + c1 \cdot r1 \cdot (2-1) + c2 \cdot r2 \cdot (1-1)) = c1 \cdot r1 \cdot (1)$$

...

$$Vel[3][3] = ceil(w \cdot 0 + c1 \cdot r1 \cdot (0-2) + c2 \cdot r2 \cdot (2-2)) = c1 \cdot r1 \cdot (-2)$$

Рисунок 2.4 – Отримання вектору швидкості

Після формування кожного елементу так, як показано вище, перевіряється, чи не є елемент від'ємним. Якщо елемент все-таки менший за 0, від береться за модулем розмірності блоку, тобто, для блоку $2 \times 2 - 4$, для блоку $8 \times 8 - 256$ і так далі. В нашому випадку алгоритм працює із розмірністю S-Box'ів 8×8, а отже всі від'ємні елементи будуть братися за модулем 256.

У прикладі наведено блоки розмірності 2×2, тому елемент $Vel[0][0] = -2$ буде взятий за модулем 4 і отримаємо $Vel[0][0] = 2$.

Упускаючи значення коефіцієнтів, отримуємо наступний вектор швидкості:

Таблиця 2.5 – Масив швидкостей для кожного блоку у популяції

Velocity
(2,1,0,3)
(3,2,1,2)
(2,3,1,0)
(0,1,3,2)

Далі відбувається формування нових блоків, шляхом додавання до значень початкової популяції значень вектору швидкості за формулою (2.2). На цьому кроці також відбувається перевірка того, що значення входять до потрібного діапазону. Якщо значення більше або менше необхідного, воно береться за модулем розмірності блоку. Також відбувається перевірка на бієктивність і виправлення небієктивності, якщо така є [13].

Після даної процедури ми отримуємо нову популяцію розміру 2N. Наприклад до старої популяції при покращенні додалося ще N теоретичних блоків. Маємо:

Таблиця 2.6 – Популяція після першої фази покращення і пошуку

S-box	Нелінійність
(3,1,0,2)	100%
(1,2,3,0)	85%
(2,3,1,0)	83%
(2,1,3,0)	81%
(3,2,1,0)	88%
(3,2,0,1)	85%
(2,3,0,1)	88%
(0,1,2,3)	81%

Далі знову відбувається процедура сортування популяції за нелінійністю. Після сортування останні N блоків відкидаються. Отримуємо:

Таблиця 2.7 – Відсортована популяція після першої фази покращення

S-box	Нелінійність
(3,1,0,2)	100%
(3,2,1,0)	88%
(3,2,1,0)	88%
(1,2,3,0)	85%

Далі всі процедури повторюються, заповнюються вектори *gBest* та *pBest*. До вектору *gBest* додається перший блок з масиву, а до *pBest* – всі інші блоки. Це відбувається доти, доки не закінчиться задана кількість ітерацій, або поки не буде знайдено необхідний блок [13].

2.3.2 Принцип роботи модифікованого алгоритму

Наша версія алгоритму [14-16] пропонує трохи інший підхід до пошуку блоків. Він полягає в тому, щоб за допомогою часток погіршувати завідома гарний блок для досягнення компромісу між всіма параметрами захищеності. Завдяки

такому методу вдається досягти того, щоб всі параметри були достатніми та задовольняли більшості умов. Отже, модифікований алгоритм працює наступним чином:

Спочатку все відбувається так само, як і у випадку алгоритму, розглянутого раніше: на початку виконання алгоритму у нас є завідома гарний блок, з високою нелінійністю, генеруються ще 3 випадкові блоки. Цими блоками буде заповнено масив популяції. Масив сортується у порядку спадання нелінійності блоків. Заповнюються вектори $gBest$ та $pBest$.

Відмінністю методу є те, що на першому кроці алгоритму відбувається перемішування першого блоку в новій популяції, щоб не сильно погіршити одні його показники та покращити інші і на його основі вже далі погіршувати одні показники та покращувати інші [14-16]. Припустимо, як і в минулому прикладі, у нас на початку першого кроку генерується популяція:

Таблиця 2.8 – Популяція на кроці ініціалізації модифікованого алгоритму

S-box	Нелінійність
(3,1,0,2)	100%
(1,2,3,0)	85%
(2,3,1,0)	83%
(2,1,3,0)	81%

Після цього невелика кількість значень з найкращого блоку перемішуються, щоб погіршити нелінійність не дуже значним чином. Як було описано раніше, це дає майже нульовий вектор швидкості на першій ітерації, що дозволяє поступово погіршувати показники блоку на наступних ітераціях. Таким чином, перший блок з нової популяції, яка буде додана до цієї, буде мати гарні показники, як у початкового блоку, проте він і буде дуже схожим на цей блок [14-16].

Отриманий таким чином блок має гарну нелінійність, але всі інші параметри, такі як алгебраїчний імунітет, Δ -рівномірність та лінійна збитковість є поганими, тому далі продовжується оптимізація популяції, як в алгоритмі, який описано вище.

Після сортування та відкидання отримуємо:

Таблиця 2.9 – Популяція після першої ітерації покращення модифікованого алгоритму

S-box	Нелінійність
(3,1,0,2)	100%
(3,1,2,0)	95%
(2,3,1,0)	83%
(2,1,3,0)	81%

Оптимізація відбувається доти, доки не закінчиться задана кількість ітерацій, або поки не буде знайдено необхідний блок. В нашому випадку алгоритм здатний знаходити S-блоки з нелінійністю 104 та іншими показниками, які задовольняють умовам захищеності.

2.4 Програмна реалізація та тестування роботи в реальних умовах

Розглянувши спрощену модель обох алгоритмів, пропонуємо ознайомитися з реальними прикладами виконання програми для блоків заміни розмірності 8×8, тобто довжини 256 у десятковому чи шістнадцятковому форматах.

Для роботи з блоками з використанням методу Particle Swarm Optimization було створено програмний застосунок на мові програмування C [14-16]. Даний застосунок при запуску, виконує всі кроки модифікованого алгоритму для заданої кількості ітерацій. Нами також було реалізовано і базовий алгоритм, що пропонується в статті [13]. Результати порівняння роботи двох алгоритмів будуть наведені нижче в даному розділі.

Для роботи із застосунком необхідно відкрити папку з застосунком, у якій містяться файли для налаштування режиму роботи, кількості блоків у популяції, кількості ітерацій, а також інструкція по роботі з програмою. Директорія виглядає наступним чином:

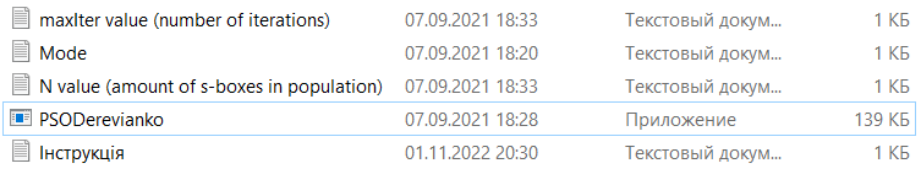

Рисунок 2.5 – Папка із застосунком PSO до запуску

У інструкції детально описано алгоритм роботи з програмним продуктом:

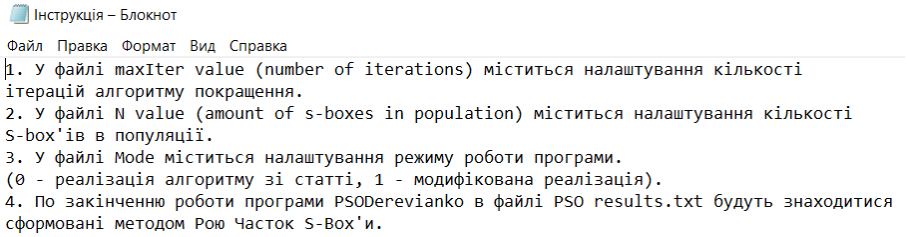

Рисунок 2.6 – Інструкція до PSO застосунку

Для демонстрації роботи було обрано наступні параметри: кількість ітерацій = 5, кількість блоків у популяції = 4:

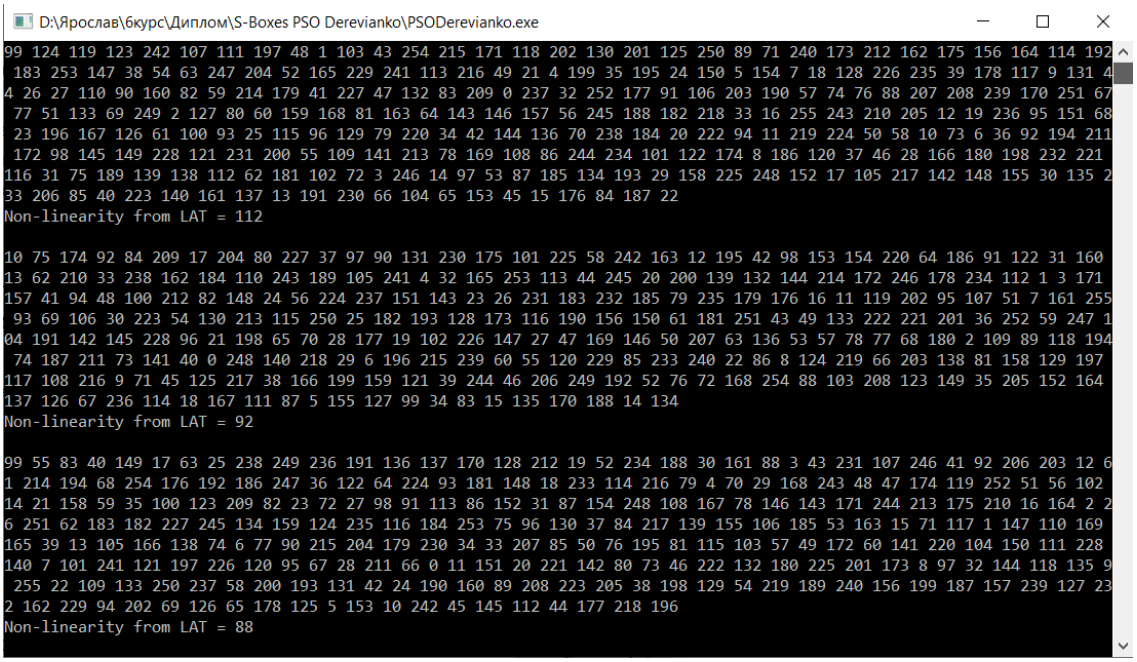

Рисунок 2.7 – Початок фази ініціалізації

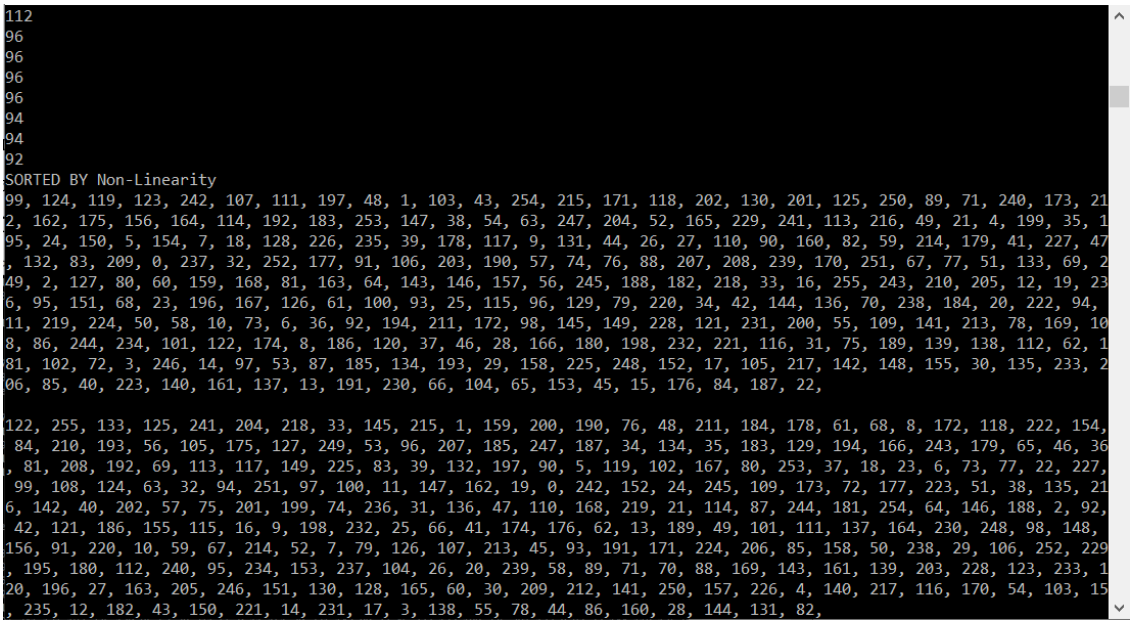

Рисунок 2.8 – Сортування в кінці останньої ітерації (mode 0)

Як видно з рисунку, популяція має значно кращі показники нелінійності, ніж на фазі ініціалізації, після генерації початкової популяції випадковим методом. Це показує, що алгоритм дійсно працює. Переглянемо файл з результатами:

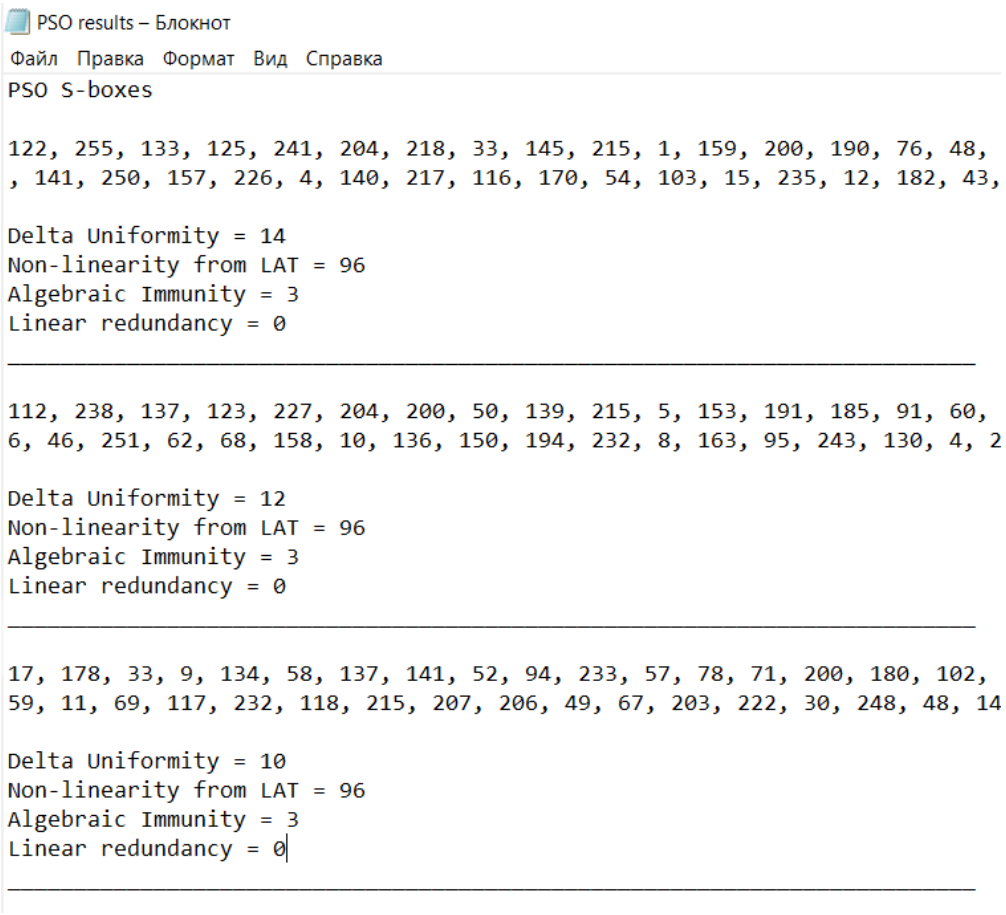

Рисунок 2.9 – Файл з результатами виконання застосунку (mode 0)

У файлі з результатами в кінці міститься N-1 блоків, оскільки AES-блок, взятий за основу, відкидається. Як можна бачити, застосунок також вираховує найбільш важливі параметри. Повний програмний код застосунку, а також деякі результати можна знайти на сервісі GitHub за посиланням [18].

Для режиму mode 1 – модифікованого алгоритму, результати наступні:

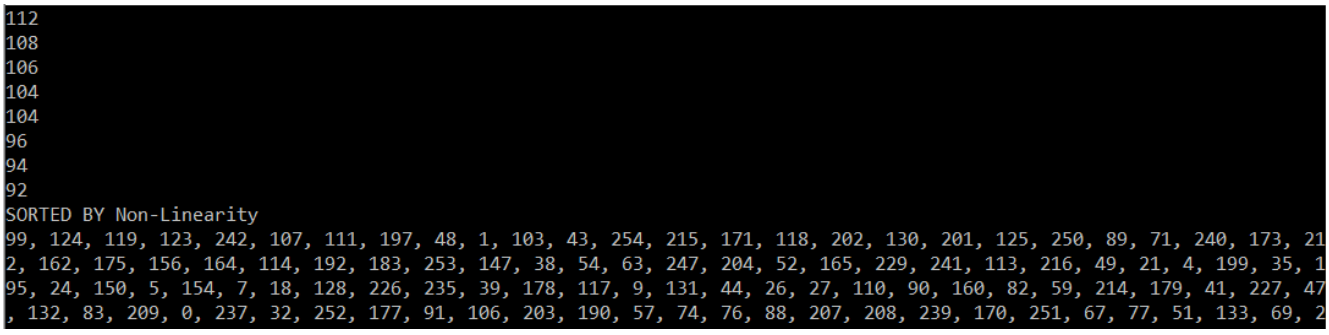

Рисунок 2.10 – Сортування в кінці останньої ітерації (mode 1)

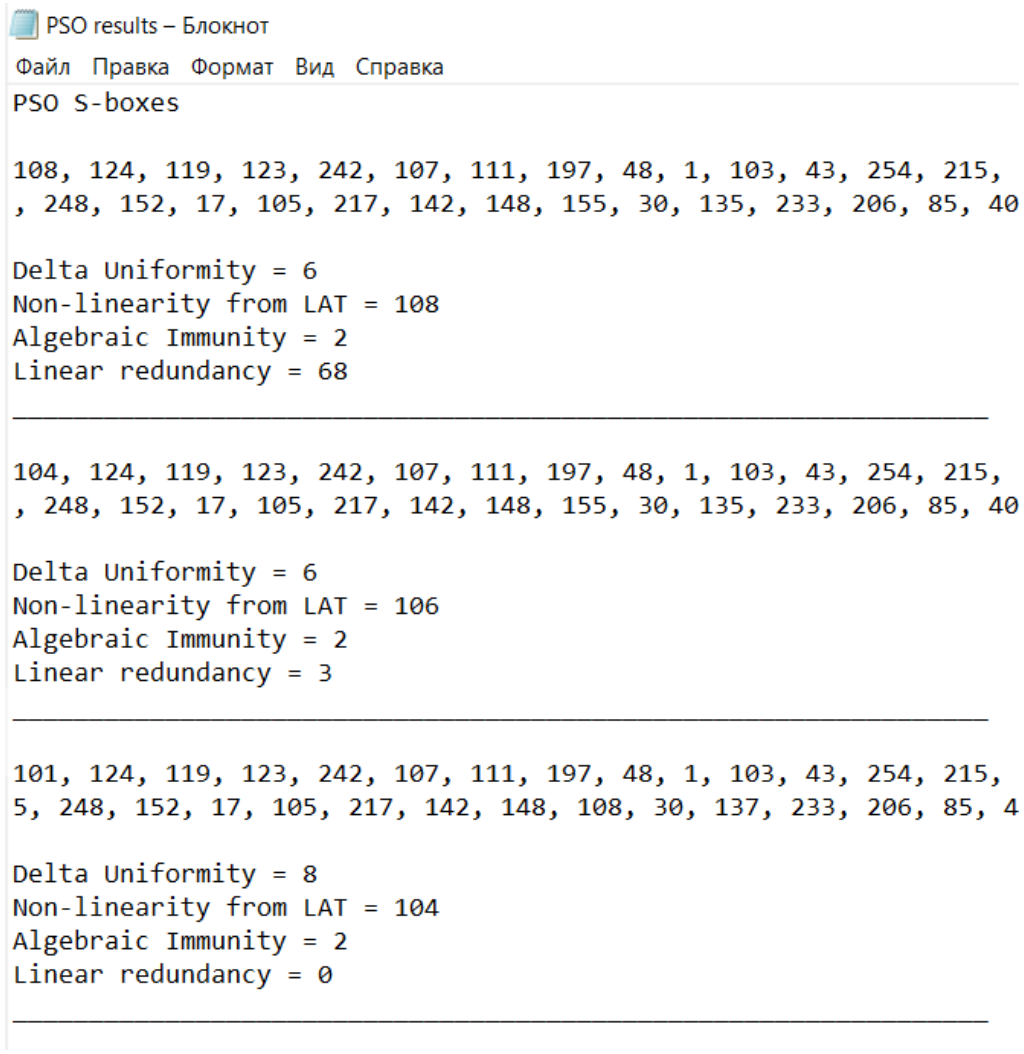

Рисунок 2.11 – Файл з результатами виконання застосунку (mode 1)

На рисунку 2.11 видно, що як і описувалося раніше, завдяки модифікації алгоритму, відбувається не поступове покращення блоку, а поступове погіршення для забезпечення оптимальності всіх параметрів. Це досягається тим, що на першій ітерації циклу алгоритму перший елемент масиву швидкостей майже нульовий, а отже при підсумовуванні з першим елементом популяції, він дасть дуже схожий блок, з хорошими показниками нелінійності, які потім треба буде прирівнювати з іншими для досягнення компромісу [14-16]:

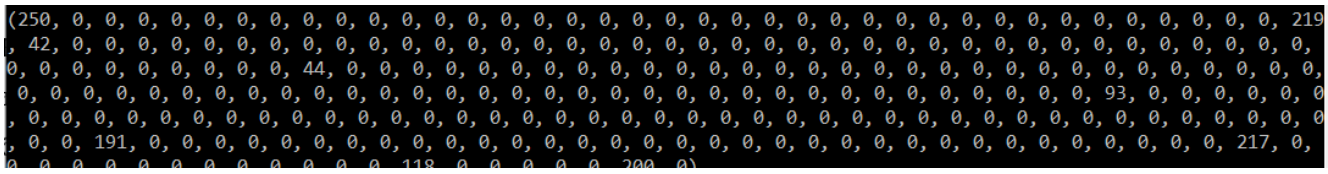

Рисунок 2.12 – Перший елемент масиву швидкостей

2.4.1 Результати тестових запусків оригінального алгоритму

При виконання тестування, були обрані такі критерії нелінійність = 104, алгебраїчний імунітет = 3, лінійна збитковість = 0 [14-16].

Таблиця 2.10 – Узагальнені результати 10 тестових запусків для різних N та MaxIter (базовий метод)

Узагальнені результати 10 тестових запусків					
N	MaxIter	Кількість ітерацій на знаходження необхідного блоку	Час роботи	Кількість позицій відмінності від AES	Коефіцієнт подібності з AES S-Box
5	10	Не знайдено	76,97 с	-	-
5	50	Не знайдено	441,12 с	-	-
5	100	Не знайдено	856,83 с	-	-
5	150	Не знайдено	1385,97 с	-	-
5	200	Не знайдено	1855,32 с	-	-
10	10	Не знайдено	183,71 с	-	-
10	50	Не знайдено	925,36 с	-	-
10	100	Не знайдено	1973,81 с	-	-
10	150	Не знайдено	2942,23 с	-	-
10	200	Не знайдено	3928,40 с	-	-
20	10	Не знайдено	380,58 с	-	-
20	50	Не знайдено	2017,78 с	-	-

Продовження таблиці 2.10 – Узагальнені результати 10 тестових запусків для різних N та MaxIter (базовий метод)

20	100	Не знайдено	3966,22 с	-	-
20	150	Не знайдено	5937,17 с	-	-
20	200	Не знайдено	7947,12 с	-	-
40	10	Не знайдено	799,80 с	-	-
40	50	Не знайдено	4009,69 с	-	-
40	100	Не знайдено	7955,15 с	-	-
40	150	Не знайдено	11546,33 с	-	-
40	200	Не знайдено	15167,66 с	-	-

Як можна бачити з результатів проведених тестів, оригінальний метод, що надається у статті [13], не спроможний знаходити S-блоки з необхідними високими параметрами. При цьому у статті стверджується, що такий метод спроможний формувати вузли зі значенням параметру нелінійності 110 та більше. Нижче наводиться приклад S-блоку, сформованого з використанням алгоритму зі статті:

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

Рисунок 2.13 – Блок, сформований з використанням базового алгоритму

Блок має наступні показники: нелінійність = 98, алгебраїчний імунітет = 3, лінійна збитковість = 0. В наступному пункті ми надаємо результати нашого модифікованого методу, показуємо, шо при незначній модифікації метод здатен отримувати блоки з хорошими показниками. До того ж модифікований метод [14-16] показує незначне, але все ж покращення, швидкодії порівняно з оригінальним.

2.4.2 Результати тестових запусків модифікованого алгоритму

Таблиця 2.11 – Узагальнені результати швидкодії тестових запусків для різних N та MaxIter і успішних знаходжень необхідних блоків (модифікований метод)

Узагальнені результати 10 тестових запусків					
N	MaxIter	Кількість ітерацій на знаходження необхідного блоку	Час роботи	Кількість позицій відмінності від AES	Коефіцієнт подібності з AES S-Box
5	10	7	51,37 с	32	0,875
5	50	Не знайдено	409,03 с	-	-
5	100	Не знайдено	827,22 с	-	-
5	150	Не знайдено	1259,54 с	-	-
5	200	Не знайдено	1642,72 с	-	-
10	10	Не знайдено	173,87 с	-	-
10	50	Не знайдено	880,23 с	-	-
10	100	25	391,46 с	32	0,875
10	150	Не знайдено	2574,56 с	-	-
10	200	119	1919,76 с	35	0,8632

Продовження таблиці 2.11 – Узагальнені результати швидкодії тестових запусків для різних N та MaxIter і успішних знаходжень необхідних блоків (модифікований метод)

20	10	Не знайдено	372,67 с	-	-
20	50	Не знайдено	1770,69 с	-	-
20	100	Не знайдено	3529,32 с	-	-
20	150	59	1910,47 с	31	0,8789
20	200	59	1892,72 с	32	0,875
40	10	Не знайдено	7598,78 с	-	-
40	50	Не знайдено	3629,01 с	-	-
40	100	Не знайдено	6943,78 с	-	-
40	150	106	6941,97 с	32	0,875
40	200	Не знайдено	13837,22 с	-	-

Слід прокоментувати дві останні колонки у таблиці 2.11. Ці колонки містять показники відмінності (абсолютні та відносні) від значень алгебраїчного S-блоку шифру AES [19]. Зазначений S-блок застосовується у якості начального заповнення однієї із часток рою, отже є певним вихідним значенням при генерації підстановок. Відповідні показники відмінності характеризують «віддалення» знайденого S-блоку від цього вихідного значення. Як бачимо, генеровані таблиці підстановок є дуже подібними до S-блоку шифру AES (80-90% подібності). Близькими за цим показником є також S-блоки, що генеровані за алгоритмом з [20].

Отримані результати свідчать про те, що алгоритм дійсно спроможний отримувати блоки з заданими параметрами. Більш детальні тестові результати надаються у [15]. Нижче наводиться інформація усіх тестових запусків у вигляді

гістограми. По осі x вказується кількість ітерацій (MaxIter), по осі y – кількість блоків в популяції (N), по z – кількість знайдених блоків.

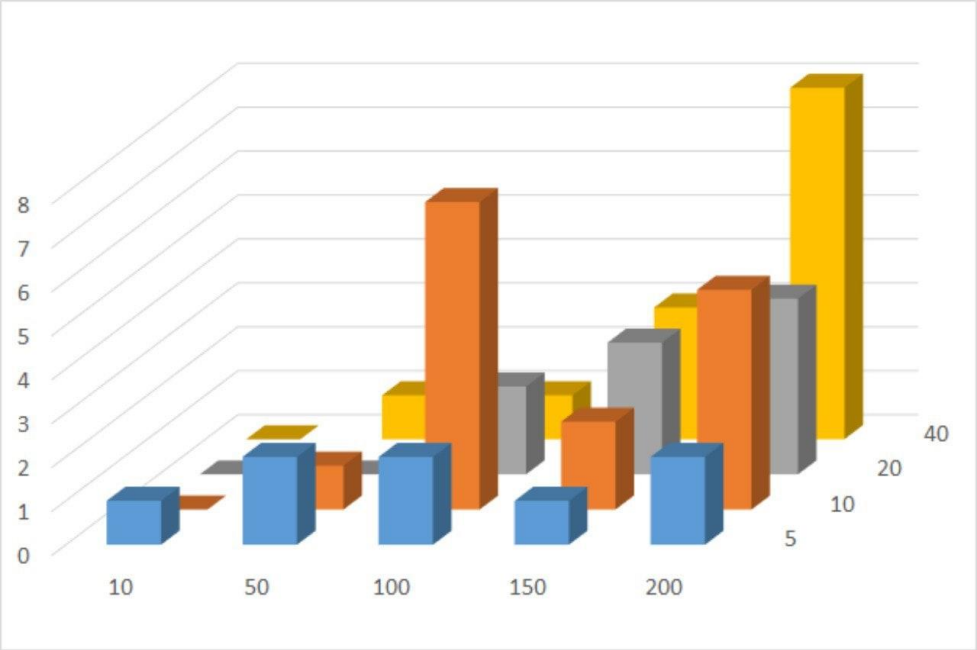

Рисунок 2.14 – Графік залежності кількості знайдених блоків від N та MaxIter

Як можна побачити, чим більшими є N та MaxIter, тим більшою є вірогідність знайти блок з необхідними параметрами.

Далі буде надано один з блоків, знайдених модифікованим алгоритмом. Нагадаємо, що блоками з необхідними параметрами є блок, який задовольняє наступним критеріям: нелінійність = 104, алгебраїчний імунітет = 3, лінійна збитковість = 0.

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

Рисунок 2.15 – Блок, сформований з використанням модифікованого алгоритму

Блок має наступні показники: нелінійність = 104, алгебраїчний імунітет = 3, лінійна збитковість = 0. Як видно, блок є доволі схожим на AES S-box та має 34 позиції відмінні від нього. Але все ж такий блок задовольняє всім необхідним критеріям, тобто має необхідний рівень показників: нелінійність, алгебраїчний імунітет та лінійна збитковість [14,16].

2.5 Висновки до розділу

Основним завдання розділу було дослідження та реалізація методу рою часток у застосуванні до задачі синтезу високоефективних нелінійних підстановок. Аналіз відкритих наукових джерел показав, що спроби застосувати Particle Swarm Optimization при генерації S-блоків були, але результати цих спроб виявилися недостовірними. Переваги методу дають змогу застосовувати його до багатьох областей оптимізації окремо та в поєднанні з іншими існуючими алгоритмами.

Розглянутий базовий алгоритм, пропонував пошук блоків лише за параметром нелінійності, що не є дуже ганою практикою, тому у модифікованому алгоритмі було прийнято рішення застосовувати декілька параметрів, зокрема, нелінійність, лінійну збитковість та алгебраїчний імунітет. Головною відмінністю в модифікованому алгоритмі є те, що при формуванні нової популяції перший блок нової популяції формується за рахунок взаємодії першого блоку з початкової популяції з самим собою і перемішування невеликої кількості позицій. Завдяки такому методу вдається досягти того, щоб всі параметри були достатніми та задовольняли більшості умов і відбувалося поступове вирівнювання всіх параметрів.

У розділі було показано реалізацію програмного застосунку для генерації блоків з використанням як базового, так і модифікованого методів. Результати, надані у розділі, показують, що оригінальний метод, не спроможний знаходити S-блоки з необхідними високими параметрами. В той час як модифікований алгоритм дійсно спроможний отримувати блоки з заданими параметрами. Хоча блок, отриманий таким чином, є доволі схожим на AES S-box, але все ж такий блок задовольняє всім необхідним критеріям і може використовуватися у БСШ чи інших криптографічних перетвореннях.

3 ВИКОРИСТАННЯ ФАКТОРІАЛЬНОЇ СИСТЕМИ ЧИСЛЕННЯ У АЛГОРИТМІ PSO

Однією з найважливіших складових комбінаторики є перестановки, вони найчастіше використовуються у реальних задачах, що стосуються різних сфер нашого життя. У відношенні до криптографічного захисту та кібербезпеки, перестановки (підстановки) можуть бути використані для шифрування повідомлень чи застосовуватися в різноманітних крипто-перетвореннях [21].

В певних ситуаціях, наприклад, при виконанні задачі пошуку або синтезу S-боксів з високими показниками, необхідно швидко переміщатися між підстановками або мати можливість отримання значення необхідної підстановки, знаючи лише її номер у просторі пошуку. Впоратися з цією задачею допоможе факторіальна система числення (ФСЧ) [22]. Її застосування дозволяє швидко відтворити підстановку знаючи лише номером цієї підстановки у просторі, а також виконати зворотну процедуру – отримати номер підстановки із самої підстановки.

У розділі пояснюються основні принципи застосування ФСЧ при роботі з підстановками, а також наводяться алгоритми, завдяки яким реалізовано переведення номерів з десяткового представлення у факторіальне, а потім із факторіального у перестановку. Також у розділі надається програмна реалізація для роботи з підстановками (S-блоками) розмірності 2^8, тобто довжини 256, яку мають S-блоки. Такий підхід має дозволити швидко переміщатися у необхідному просторі перестановок, а також дозволить збільшити випадковість отримуваних блоків, оскільки алгоритм не передбачає використання модульної логіки, як алгоритм з минулого розділу, а також не використовує завідома хороших блоків в якості цілі для пошуку.

3.1 Опис принципів факторіальної системи числення

Основами для факторіальної системи числення (ФСЧ) є послідовність факторіалів. Натуральне число (число без знаку) у такій системі буде представлятися наступним чином [21-22]:

$$x = \sum_{k=1}^{n} q_k k! \tag{3.1}$$

де $0 \le q_k \le k$

Такий спосіб можна використовувати для декодування перестановок (підстановок) із застосуванням списків інверсій. Це означає, що знаючи номер підстановки (перестановки) у просторі, є можливим її відтворення як показано далі: номер підстановки може бути представлений у ФСЧ так, що коефіцієнт при $k!$ означатиме число інверсій елементу $k+1$ у множині самої перестановки (розмірності) – кількість елементів, що є меншими за $k+1$, але розміщені справа від елементу в перестановці, що шукається).

Далі більш детально розглянемо алгоритм представлення натуральних чисел у ФСЧ. Для того, щоб представити число без знаку x у ФСЧ, необхідно знайти набір q_k, де $k \in [1, n]$, таких, що [21-22]:

$$x = \sum_{k=1}^{n} q_k k! = q_n n! + q_{n-1}(n-1)! + \ldots + q_2 2! + q_1 1! \tag{3.2}$$

де $0 \le q_k \le k$

Представлення відбуватиметься наступним чином [21]:

$$\begin{aligned} x &= q_n n! + r_n \\ r_n &= q_{n-1}(n-1)! + r_{n-1} \\ &\ldots \\ r_2 &= q_2 2! + r_2 \\ r_2 &= q_1 \end{aligned} \tag{3.3}$$

Отже, натуральне число у ФСЧ буде представляти собою набір $q_n, q_{n-1}, \ldots, q_2, q_1$. Як приклад, наведемо перетворення числа 100 з десяткового представлення до ФСЧ [21]:

$$\begin{aligned} 100 &= 4 \cdot 4! + 4 \to q_4 = 4 \\ 4 &= 0 \cdot 3! + 4 \to q_3 = 0 \\ 4 &= 2 \cdot 2! + 0 \to q_2 = 2 \\ 0 &= 0 \to q_1 = 0 \end{aligned} \tag{3.4}$$

Оскільки q_0 завжди буде нульовим, маємо наступне представлення числа: {4,0,2,0,0}. Проте, такий алгоритм має певні труднощі, оскільки потребує постійного обчислення факторіалів $k!$. При збільшенні k це може стати серйозною проблемою, що значним чином впливатиме на погіршення швидкодії. Щоб запобігти цьому, можна скористатися схемою Горнера для представлення [22]:

$$x = q_n n! + q_{n-1}(n-1)! + \ldots + q_2 2! + q_1 1! = (\ldots(q_n n + a_{n-1})(n-1) + \ldots + a_2)2 + a_1 \qquad (3.5)$$

Представлення відбуватиметься наступним чином [21]:

$$\begin{aligned} x &= r_1 2 + q_1 \\ r_1 &= r_2 3 + q_2 \\ r_2 &= r_3 4 + q_3 \\ &\ldots \\ r_{n-3} &= r_{n-2}(n-1) + q_{n-2} \\ r_{n-2} &= r_{n-1} n + q_{n-1} \\ r_{n-1} &= q_n \end{aligned} \qquad (3.6)$$

Натуральне число у ФСЧ буде представлене набором $q_n, q_{n-1}, ..., q_2, q_1$. Як приклад, наведемо перетворення числа 100 з десяткового представлення до ФСЧ:

$$\begin{aligned} 100 &= 50 \cdot 2 + 0 \rightarrow q_1 = 0 \\ 50 &= 16 \cdot 3 + 2 \rightarrow q_2 = 2 \\ 16 &= 4 \cdot 4 + 0 \rightarrow q_3 = 0 \\ 4 &= 4 \rightarrow q_4 = 4 \end{aligned} \qquad (3.7)$$

Оскільки q_0 завжди буде нульовим, маємо наступне представлення числа у ФСЧ: {4,0,2,0,0}. Такий метод представлення є значно швидшим та кращим з точки зору продуктивності і використання ресурсів обчислювальної техніки. Це зумовлено тим, що він потребує виконання значно меншої кількості обчислювальних операцій.

Після отримання числа у факторіальному представленні, представлення цього числа у вигляді підстановки (перестановки) відбувається наступним чином (ФЧ ={4,0,2,0,0}) [21]:

- перший залишок отриманого числа у ФСЧ показує кількість елементів, що менші за n (в нашому випадку n=5, оскільки підстановка складається з 5

елементів), але розміщені справа від нього у перестановці, отже, починаємо заповнювати перестановку:

$$\{5, \_, \_, \_, \_\} \tag{3.8}$$

- другий залишок вказує на кількість елементів, що менші за n-1, але розміщені справа від нього:

$$\{5, \_, \_, \_, 4\} \tag{3.9}$$

- третій залишок показує кількість елементів, що менші за n-2, але розміщені справа від нього:

$$\{5, 3, \_, \_, 4\} \tag{3.10}$$

- четвертий залишок вказує на кількість елементів, що менші за n-3, але розміщені справа від нього:

$$\{5, 3, \_, 2, 4\} \tag{3.11}$$

- останній залишок займає останнє вільне місце, оскільки місць не лишилось. Отже, після виконання переходу підстановка буде наступною:

$$\{5, 3, 1, 2, 4\} \tag{3.12}$$

Для отримання номеру підстановки необхідно спочатку перевести її у число у факторіальному вигляді. Для цього слід для кожного залишку, починаючи з найбільшого, порахувати кількість залишків (значень), які будуть меншими за значенням. При цьому такі залишки маються знаходитися правіше у ФЧ.

Для перестановки (5,3,1,2,4) наприклад, кількість елементів, менших за 5 (значення конкретного елементу підстановки), але які стоять у підстановці правіше цього елементу дорівнює 4, таких, що були б меншими за 4, але знаходилися правіше – 0. І так далі, поки не будуть заповнені всі залишки [21].

Шляхом виконання алгоритму, описаного вище з підстановки (5,3,1,2,4) було отримано ФЧ {4,0,2,0,0}. Тепер, щоб перейти від ФЧ до десяткового представлення кожен залишок необхідно помножити на відповідну йому основу – факторіал $k!$.

Для того, щоб отримати десяткове число з числа у ФСЧ {4,0,2,0,0} слід зробити наступне:

$$0 \cdot 0! + 0 \cdot 1! + 2 \cdot 2! + 0 \cdot 3! + 4 \cdot 4! = 0 + 0 + 4 + 0 + 96 = 100 \tag{3.13}$$

Нами було реалізовано спочатку алгоритми, які дають змогу виконувати такі перетворення стосовно стандартних типів даних (int, long, тощо). Для виконання таких дій було реалізовано програмний застосунок, робота якого буде показана в наступному пункті.

3.1.1 Програмні методи представлення чисел у різних СЧ

Для виконання операцій переведення чисел з однієї системи числення у іншу, було розроблено застосунок, на вхід застосунку подається розмірність перестановок (межа у вигляді факторіалу для перестановок такої розмірності) і саме число, або ж перестановка, в залежності від операції, яку необхідно виконати. Програмний код застосунку доступний на [23].

Нижче надається програмний код методів:

Алгоритм на рисунку 3.1 повертає масив з залишками факторіального числа для подальшого використання у переведенні [23]:

```
int *numberToFactorial(int x, int n){
    int q = 2; int counter = n-2;
    int *positions = calloc (n,sizeof(int));
    positions[n-1] = 0;
    while (counter > 0){
        int temp = floor( _X: x/q);
        int val = x - (temp*q);
        positions[counter] = val;
        x = temp;
        ++q; counter--;
    }
    positions[counter] = x;
    int *positionsRev = calloc (n,sizeof(int));
    for (int r = 0,t = n-1; r < n, t>=0; ++r, t--){
        positionsRev[t] = positions[r];
    }free(positions);
    return positionsRev;
}
```

Рисунок 3.1 – Програмний код алгоритму переведення числа з десяткового представлення до факторіального

Алгоритм на рисунку 3.2 повертає масив з підстановкою довжини n [23]:

```
int *numberToSubstitution(int *number, int size){
    int *S = calloc (size,sizeof(int)); int *result = calloc (size,sizeof(int));
    int newSize = size; int counter = 0; int coeffNum = 0; int innerCounter = 0;
    while(counter < size){
        int *emptyPos = calloc (newSize,sizeof(int));
        for (int i = 0; i < size; ++i){
            if (S[i]==0){
                emptyPos[innerCounter] = i;
                innerCounter++;
            }
        }
        for (int q = 0; q < newSize; ++q){
            if (q==number[newSize-1]){
                S[emptyPos[q]] = 1;
                result[coeffNum] = emptyPos[q];
                ++coeffNum;
            }
        }
        innerCounter = 0; ++counter; newSize--; free(emptyPos);
    }
    int numberInArr = size; int *sub = calloc (size,sizeof(int));
    for (int u = 0; u < size; ++u){
        sub[result[u]] = numberInArr;
        numberInArr--;
    } free(result);
    int *subRev = calloc (size,sizeof(int));
    for (int r = 0,t = size-1; r < size, t>=0; ++r, t--){
        subRev[t] = sub[r];
    } free(sub); free(S);
    return subRev;
}
```

Рисунок 3.2 – Програмний код алгоритму переведення числа у ФСЧ до перестановки

Алгоритм на рисунку 3.3 повертає масив з залишками ФЧ [23]:

```
int *substitutionToFactorial(int *sub, int size) {
    int *result = calloc(size,sizeof(int)); int value = size;
    int flag = 0; int innerCounter = 0; int counter = 0;
    while (counter<size) {
        for (int i = 0; i < size; ++i) {
            if (flag == 1 && sub[i] < value) {
                ++innerCounter;
            }
            if (sub[i] == value) {
                flag = 1;
            }
        }
        result[counter] = innerCounter;
        innerCounter = 0; flag = 0; ++counter; value--;
    }
    return result;
}
```

Рисунок 3.3 – Програмний код алгоритму переведення перестановки до числа у ФСЧ

Алгоритм на рисунку 3.4 повертає число у десятковій системі числення [23]:

```
int factorialNumberToNumber(int *number, int size){
    int *numberRev = calloc (size,sizeof(int));
    int result = 0;
    for (int r = 0,t = size-1; r < size, t>=0; ++r, t--){
        numberRev[t] = number[r];
    }
    for (int i = 0; i < size; ++i){
        result += numberRev[i]*factorialCounting(i);
    }
    free(numberRev);
    return result;
}
```

Рисунок 3.4 – Програмний код алгоритму переведення числа у ФСЧ до числа у десятковому вигляді

Нижче надаються декілька прикладів переведення чисел обох напрямках:

```
START NUMBER
100

FACTORIAL NUMBER
4, 0, 2, 0, 0,

SUBSTITUTION
5, 3, 1, 2, 4,

FACTORIAL NUMBER
4, 0, 2, 0, 0,

FINAL NUMBER
100
```

Рисунок 3.5 – Приклад представлення числа 100 у вигляді перестановки і зворотного перетворення

```
START NUMBER
3628000

FACTORIAL NUMBER
9, 8, 7, 5, 5, 1, 2, 2, 0, 0,

SUBSTITUTION
10, 9, 8, 6, 7, 3, 4, 1, 5, 2,

FACTORIAL NUMBER
9, 8, 7, 5, 5, 1, 2, 2, 0, 0,

FINAL NUMBER
3628000
```

Рисунок 3.6 – Приклад представлення числа 3 628 000 у вигляді перестановки і зворотного перетворення

Як видно з рисунків відносно підстановок, номер яких не перевищує ліміти стандартних типів даних, розроблені алгоритми працюють правильно. Основне обмеження, яке не дає реалізувати такі методи для великої кількості перестановок стандартними засобами мови програмування це розмір типів даних, в яких розміщуються їх номери.

Оскільки S-блоки мають вигляд перестановки з 256 елементів, то кількість таких перестановок буде 256!. Таке число не можливо помістити до звичайних типів даних. У наступному пункті буде показано можливість застосування даних методів для перебору нелінійних перестановок S-блоків із застосуванням бібліотек для роботи з дуже великими цілими числами.

3.2 Застосування факторіальної системи числення до алгоритму PSO

Як описувалось раніше, факторіальна система може застосовуватися для виконання дій над підстановками, отримання їх номерів, маючи тільки саму підстановку і навпаки отримання підстановки за номером.

Оскільки S-Box [2-4] можна представити як перестановку елементів від 0 до 255, застосування факторіальної системи числення по відношенню до нелінійних перестановок може бути корисним, наприклад з метою їхнього перебору за номерами з метою отримати перестановки з гарними показниками.

3.2.1 Загальна концепція та програмна реалізація

Принцип роботи методу Particle Swarm Optimization було детально описано в минулому розділі роботи. Нагадаємо, що метод рою часток - це евристичний метод глобальної оптимізації, який реалізовано на основі популяцій. PSO базується на уявленні про «ройовий» інтелект.

Щоб якось пристосувати роботу з нелінійними вузлами заміни до концепції ройового інтелекту, де кожна частинка має положення і швидкість, ми пропонуємо використовувати номер перестановки у множині всіх перестановок як значення певного розміщення, тобто положення в цій множині. Швидкість частки буде також певним числом, яке буде отримуватись на основі віддаленості перестановки від найкращої перестановки у всій популяції та на основі найкращої перестановки на місці якої була частка за весь час роботи алгоритму.

Швидкість кожної частинки визначається на кожному кроці за формулою:

$$v_{id}^{k+1} = v_{id}^{k} + \alpha(pBest_{id}^{k} - x_{id}^{k}) + \beta(gBest - x_{id}^{k}) \quad (3.14)$$

де $pBest_{id}^{k}$ – номер найкращої перестановки на місці якої була частка за весь час роботи алгоритму (певна властивість пам'яті частки).

$gBest$ – номер найкращої з усіх частки, знайденої популяцією за весь час роботи алгоритму (найкраща частинка в популяції в глобальному плані).

Коефіцієнти α та β обираються таким чином: α обирається випадковим чином в діапазоні [0,1], а $\beta = 1 - \alpha$.

Положення частинки визначається на кожному кроці за формулою [13]:

$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1} \quad (3.15)$$

де, v_{id}^{k} та x_{id}^{k} це швидкість та позиція частки «i» на її «k» позиції (ітерації).

Для роботи з номерами перестановок, які лежать в діапазоні від 0 до 256! не підходять стандартні типи даних, тому було прийнято рішення використовувати бібліотеку для роботи з дуже великими цілими числами – NTL [24]. NTL – це високопродуктивна бібліотека C++, що забезпечує структури даних і алгоритми для маніпулювання цілими числами довільної довжини зі знаком.

Оскільки бібліотека призначена для роботи з цілими числами, замість коефіцієнтів з плаваючою крапкою α та β було прийнято рішення спочатку обирати числа цілі α та β в діапазоні [0,1000000] і після домноження чисел на ці коефіцієнти, ділити їх на праву межу діапазону, отримуючи таким чином то й же самий ефект, все ще працюючи з цілими числами.

Враховуючи роботу з великими числами, зміни набули деякі методи перетворення [25]:

```
NTL::ZZ factorialCounting(int x){
        ZZ result = conv<ZZ>(1);
        for (int i = 1; i <= x; ++i){
        result = result*i;
    }
    return result;
}
```

Рисунок 3.7 – Програмний код обчислення факторіалу для великих чисел

Функціонування методу є доволі простим, він бере число, яке подається як вхідні дані і у циклі множить початкове число, яке спочатку дорівнює одиниці на номер ітерації циклу, такий цикл триває доки кількість ітерації не буде більшою за число, яке подане на вхід. Застосування типу даних NTL::ZZ [24] зумовлено тим, що для розрахунку чисел більших ніж 12! Не підходять звичайні типи даних.

Сутність роботи алгоритму переведення числа з десяткового представлення до факторіального, така ж як і для невеликих чисел (рис 3.1) і полягає в наступному: останній елемент масиву прирівнюється до 0, далі, поки кількість ітерацій циклу менша, ніжно розмір масиву виконується ділення вхідного числа на певну змінну q, яка дорівнює 2 на першій ітерації і збільшується на 1 з кожною новою ітерацією. Залишок від такого ділення і присвоюється елементам масиву починаючи з передостаннього і закінчуючи першим.

Принцип роботи алгоритму переведення числа у ФСЧ до перестановки (рис 3.2) той самий, що і для невеликих чисел [21], спочатку створюється масив вільних місць в перестановці, і зважаючи на правила, за якими відбуваються перетворення факторіального числа в перестановку поступово заповнюється масив з результатом. По мірі заповнення результуючого масиву, елементи з масиву вільних позицій викреслюються і не враховуються при розстановці подальших елементів.

Далі розглянемо зворотні перетворення [25]:

```
NTL::ZZ factorialNumberToNumber(int *number, int size){
    int *numberRev = (int *) calloc (size,sizeof(int));
    ZZ result = conv<ZZ>(0);
    for (int r = 0,t = size-1; r < size, t>=0; ++r, t--){
        numberRev[t] = number[r];
    }
    for (int i = 0; i < size; ++i){
        result += numberRev[i]*factorialCounting(i);
    }
    free(numberRev);
    return result;
}
```

Рисунок 3.8 – Програмний код алгоритму переведення числа у ФСЧ до числа у десятковому вигляді для великих чисел

Сутність методу полягає у домноженні кожного з залишків факторіального числа на відповідн основу $k!$ та підсумовуванні результатів таких множень.

Алгоритм переведення перестановки до числа у ФСЧ (рис 3.3) є таким самим, як і для невеликих чисел і полягає в наступному: для кожного елементу перестановки, починаючи з найбільшого, рахується число елементів, які розміщуються правіше нього і є меншими за цей елемент, таким чином утворюється кожен з залишків факторіального числа.

Концепцію методу Particle Swarm Optimization з використанням ФСЧ буде наведено нижче.

3.2.2 Послідовність роботи алгоритму

Робота алгоритму складається з таких пунктів:

- Ініціалізація популяції

У випадку використання Particle Swarm Optimization методу у відношенні нелінійних вузлів заміни – S-Box'ів, кожен блок на початковій стадії є часткою рою. В залежності від заданого розміру генерується N випадкових S-блоків і кожен з них є однією часткою рою відповідно.

Після заповнення популяції знаходиться нелінійність всіх блоків у популяції, для знаходження блоку з найкращою нелінійністю. Ініціалізуються масив для зберігання значень номеру найкращої з усіх частки, знайденої популяцією за весь час роботи алгоритму (найкраща частинка в популяції в глобальному плані) – $gBest[1]$ та масив для зберігання номерів найкращих перестановок на місці яких була кожна з часток за весь час роботи алгоритму (певна властивість пам'яті частки) – $pBest[N]$, а також масиви для зберігання нелінійностей блоків, які знаходяться в цих масивах – $gBestNL[1]$ та $pBestNL[N]$.

- Обчислення нелінійності, заповнення gBest та pBest.

Для кожного блоку з популяції обчислюється значення нелінійності для знаходження кращого за нелінійністю блоку, цей блок поміщається у $gBest$, а вся популяція поміщається до масиву $pBest$, оскільки на першому кроці немає відомих кращих значень для елементів популяції, так як популяція ще не починала рух і

кожна частка популяції на першому кроці пам’ятає початковий блок, як найкращий, в якому вона побувала. Також на даному кроці заповнюються масиви нелінійностей *gBestNL*[1] та *pBestNL*[N].

- Представлення кожного з блоків популяції у вигляді номеру підстановки, якій відповідає даний блок

Для цього спочатку ініціалізуються масиви для зберігання перетворених у великі цілі числа типу NTL::ZZ [24] номери кожного з блоків. Відбувається ініціалізація масивів для зберігання номерів блоків з *pBest*, *gBest* та самої популяції, яка буде рухатися та змінювати своє положення на кожному кроці, а також ініціалізується масив швидкості, яка саме і буде розраховуватися за формулою (3.14) на основі віддаленості блоків популяції від найкращого блоку, а також на основі індивідуального досвіду кожного блоку.

Кожен з масивів має тип даних NTL::ZZ, який може зберігати в собі нескінченно великі цілі числа. Масив ZZ p[N] – призначений для зберігання номерів S-Box’ів, що знаходилися в популяції *pBest*, масив ZZ g[1] призначений для зберігання номеру S-Box’у, що знаходився в *gBest* – глобального максимуму, в якому побувала популяція за весь час руху, масив ZZ v[N] призначений для зберігання швидкостей кожного з блоків популяції, а масив ZZ x[N] призначений для зберігання самої популяції, яка буде рухатися за заданими правилами в залежності від наближеності до найкращого рішення.

Далі буде надано псевдокод частин, в яких відбувається заповнення перерахованих вище масивів. Псевдокод показано на рисунках 3.10 – 3.13 [25]:

```
int * factorialNumber ← перетворення перестановки на число у ФСЧ
ZZ gRes← перетворення числа у ФСЧ на число у десятковій СЧ
g[0] ← gRes
```

Рисунок 3.9 – Заповнення gBest

```
for (int i = 0; i < N; ++i)
int * factorialNumber ← перетворення перестановки на число у ФСЧ
ZZ pRes← перетворення числа у ФСЧ на число у десятковій СЧ
p[i] ← pRes
end for
```

Рисунок 3.10 – Заповнення pBest

```
for (int i = 0; i < N; ++i)
int * factorialNumber ← перетворення перестановки на число у ФСЧ
ZZ xRes← перетворення числа у ФСЧ на число у десятковій СЧ
x[i] ← xRes
end for
```

Рисунок 3.11 – Заповнення x

```
for (int i = 0; i < N; ++i)
x[i] ← 0
end for
```

Рисунок 3.12 – Заповнення v

- Фаза пошуку нових оптимальних блоків

Після завершення всіх попередніх кроків, алгоритм готовий до фази оптимізації та пошуку нових блоків, на основі створеної популяції методом рою часток.

Псевдокод, оптимізаційної фази алгоритму Particle Swarm Optimization з використання ФСЧ надано на рисунку 3.13:

```
Генерація випадкових блоків
Заповнення gBest
Заповнення pBest
Заповнення x
Заповнення v
while (chack){
    Генерація випадкових параметрів   та
    Ініціалізація швидкості роботи
    for (int i = 0; i < N; ++i)
        Обчислення масиву швидкостей (формула 3.14)
        Обчислення нової популяції (формула 3.15)
        int *factNum ← перетворення числа x[i] у десятковій СЧ до числа у ФСЧ
        int *subFin ← перетворення числа factNum  у перестановку
        Обчислення нелінійності (NL)
        if (NL >= pBestNL[i])
                if (NL >= gBestNL[0])
                        Оновлення gBest та gBestNL
                end if
                Оновлення pBest та pBestNL
        end if
        Обчислення алгебраїчного імунітету (ai)
        if (NL >= 102 && ai == 3)
                Вивід найкращого блоку у файл, за виконання умови
                chack ← 0;
        end if
        Виведення всієї популяції у файл
    end for
end while
Вихід з алгоритму PSO
```

Рисунок 3.13 – псевдокод оптимізаційної фази алгоритму PSO

Після демонстрації псевдокоду алгоритму, наводиться детальне пояснення оптимізаційної фази алгоритму PSO із застосуванням номерів перестановок як певних мір розміщення в просторі.

Спочатку створюється потік, який буде використовуватися для запису даних у файл PBResults.txt – тобто номери елементів популяції, відповідні їм перестановки та показники нелінійності цих перестановок. Далі ініціалізується змінна, яка буде відповідати за цикл while та завдяки якій цикл оптимізації буде перервано при знаходженні блоку з необхідними показниками. Необхідними показниками при застосуванні даного методу були визначені такі показники як нелінійність та алгебраїчний імунітет.

Після цього першою операцією на кожній ітерації циклу є генерація випадкових коефіцієнтів α та β, які будуть використовуватися при знаходженні швидкості кожної частки за формулою (3.14).

Далі за формулою (3.14) обчислюється швидкість для кожної частки на основі p[i] та g[0], тобто на основі особистого кращого значення, в якому була частка за весь час роботи алгоритму та найкращої частки для всієї популяції за весь час роботи алгоритму. За формулою (3.15) обчислюється значення нової популяції, яка за рахунок додавання до старої частки швидкості буде поступово наближатися до частки яка є найкращою у глобальному плані.

Після обчислення нового значення номеру перестановки відповідної частки, який буде належати до типу даних NTL::ZZ, виконується перетворення цього номеру у перестановку шляхом використання методів, які наведені вище.

Після перетворення нових номерів перестановок у самі перестановки виконується знаходження показників цих перестановок. Якщо показник нелінійності нової перестановки є кращим, за показник перестановки до оптимізації відповідні значення в масивах замінюються кращими – тобто відбувається оновлення популяцій *gBest* та *pBest* – масиву для зберігання значень номеру найкращої з усіх частки, знайденої популяцією за весь час роботи алгоритму та масиву для зберігання номерів найкращих перестановок на місці яких була кожна з часток за весь час роботи алгоритму відповідно.

Останнім кроком є перевірка виконання умови відповідності блоків заданим параметрам нелінійності та імунітету. Якщо необхідних параметрів досягнуто – робота алгоритму закінчується і значення номеру, перестановки та нелінійності найкращого блоку записується у файл GBest.txt. Реалізація на мові програмування C++, в якій надано всі описані алгоритми та методи доступна за наступним посиланням [25].

3.3 Демонстрація тестового запуску алгоритму

Було проведено тестовий запуск описаної реалізації для рою розміру 10000 блоків, далі буде показана поведінка та зміни на кожній ітерації для однієї з часток цього рою.

Результати в таблиці описуються так: спочатку на стадії ініціалізації серед усіх блоків було знайдено найкращий і популяція, і обраний блок в тому числі, почала поступово наближатися до цього найкращого блоку. Зміни в *pBest* масиві відбувались тільки якщо було знайдено блок з кращими показниками нелінійності.

Таблиця 3.1 – Рух однієї з частинок популяції за 20 ітерацій (надаються тільки ітерації, коли алгоритм знаходив кращу частинку)

Номер блоку	Iter	NL
63754815452671140378387332090814083944221881130654...17227	1	94
51299366523252497274827853620412934081386409406132...33823	2	96
503327304794628129691553254954982259624648586668886...74323	7	96
490760282076604426033213079768737723267121502893 76...24621	15	98

На ітерації 15 блок потрапив до перестановки значення нелінійності якої дорівнювало 98, тому до кінця 20 ітерацій його значення в *pBest* не змінювалось, оскільки кращих блоків не було знайдено. Повну таблицю надано у додатку Б. Як видно з результатів в таблиці, обраний блок поступово від ітерації до ітерації, як і інші блоки у популяції, рухався, досліджуючи пошуковий простір, аж доки не натикався на блок з кращими показниками і після цього вже змінював курс руху на основі знайденого блоку, свого індивідуального досвіду, а також загального

досвіду – найкращого відомого блоку, знайденого усім роєм за весь час. Отже, шляхом виконання 20 ітерацій вдалося знайти наступний блок:

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

Рисунок 3.14 – S-блок знайдений шляхом виконання 20 ітерацій пошуку за допомогою PSO з використанням ФСЧ

Показники такого блоку є наступними: нелінійність = 98, алгебраїчний імунітет = 3, лінійна збитковість = 0 та дельта-рівномірність = 10.

Отже, аналізуючи отримані результати використання алгоритму на практиці, можна переконатися, що алгоритм PSO з використанням факторіальної системи числення та при роботі з простором блоків як з одновимірним простором, у якому кожен блок (перестановка) буде оцінюватися як номер у загальній кількості перестановок, є не дуже ефективним, оскільки він не може отримати блоки з нелінійністю більш як 98, принаймні за розумний час та з використанням звичайних комп’ютерів.

В наступному пункті буде запропоновано можливе рішення, яке покращить процедуру переміщення популяції саме до блоків, які будуть кращими не тільки за показником нелінійності, тобто за одним з коефіцієнтів спектру Уолдша-Адамара, а за сукупністю декількох найбільших коефіцієнтів з цього спектру. Щоб досягти цього будуть використовуватися різноманітні цінові (cost) функції.

3.4 Використання цінових функцій для покращення роботи алгоритму

Для покращення можливостей алгоритму знаходити блоки з необхідними значеннями показників, було прийнято рішення використовувати в якості критерію пошуку цінову функцію кожного блоку замість його нелінійності. Аналізуючи наукові роботи в області нелінійних підстановок, можна побачити, що при використанні тільки нелінійності, як певного критерію пошуку, вдається досягти значень нелінійності не більше ніж 96-98 в просторі блоків 8×8. Пов'язано це з тим, що нелінійність містить лише інформацію про найбільше значення спектру і не враховує жодної інформації про інші значення спектру.

З метою покращення можливостей пошуку алгоритмом хороших за спектром блоків, нами було використано три цінові функції: WHS, запропоновану Clark et al [26], WCF, запропоновану Alejandro Freyre-Echevarr´ıa та Ismel Mart´ınez-D´ıaz у [27] та PCF [28]. Кожна з цих функцій підвищує точність вибору саме «хороших» блоків, оскільки враховує не один найбільший коефіцієнт, а декілька коефіцієнтів, які є наближеними до найбільшого. Далі детально розглянемо кожну з функцій та їх реалізацію на мові програмування C++ [29].

3.4.1 WHS

Функція визначається як [26]:

$$WHS = \sum_{y \in F_2^m} \sum_{x \in F_2^n} \| W_f(x, y) | - X |^R \tag{3.16}$$

Код, на мові програмування C++ надається далі [28]:

```
long long costFunctionWHS(int *sbox, int size, int count) {
    long long res = 0;
    int X1 = 21;
    int R1 = 7;
    long long calc = 1;
    int *sbox_b = SBoxToBooleanFunc(sbox,size,count);
    for (int i = 0; i < count; ++i) {
        int *fxarr = HadamardCoefficients( func: sbox_b + i * size, size, count);
        for (int j = 0; j < size; ++j) {
            int w = abs( _X: abs(fxarr[j]) - X1);
            res += raiseToPowerLong(w, R1);
        }
        free(fxarr);
    }
    return res;
}
```

Рисунок 3.15 – Програмний код цінової функції WHS

Як видно з виразу 3.16 обчислення вартості полягає в наступному: береться абсолютне значення кожного з коефіцієнтів спектру Уолдша-Адамара всіх булевих функцій S-box'у – від нього віднімається певне стале значення (у нашому випадку 21). Абсолютне значення після такого віднімання зводиться до певного ступеня (у нашому випадку 7). Ціновою функцією блоку буде сума всіх коефіцієнтів після виконання над ними дій, перерахованих вище. Далі буде показано залежність вартості блоків від їх нелінійності з використанням цінової функції:

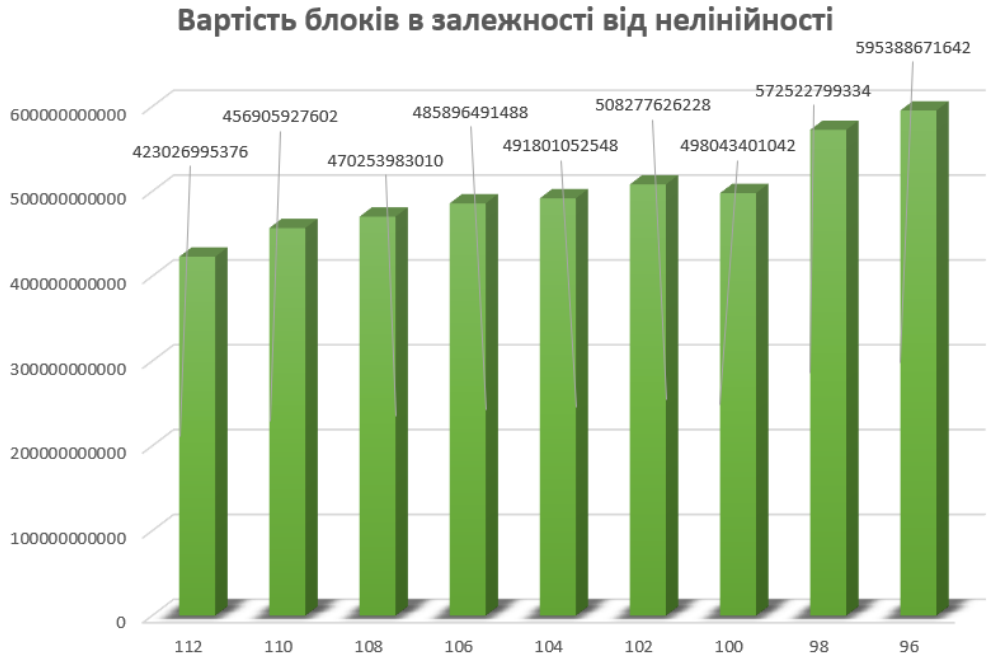

Рисунок 3.16 – Залежність вартості блоків від їх нелінійності (WHS)

Як видно з графіку, «вартість» блоку майже для всіх блоків знаходиться в прямій залежності від нелінійності, проте деякі з блоків можуть мати меншу «вартість» при тому, що будуть мати меншу нелінійність. Також з отриманих даних видно, що сусідні за значеннями нелінійності блоки дуже схожі за значеннями «вартості», що може стати проблемою та ускладнювати пошук у великому просторі блоків.

Також було запущено алгоритм PSO, описаний раніше, з використанням цінової функції WHS у якості оціночної міри блоків, замість нелінійності. Єдиною відмінністю є те, що замість розрахування нелінійності і сортування на її основі, в алгоритмі для цього використовуються функція WHS.

Використання цінової функції пришвидшило пошук блоків з нелінійністю 96-98, проте ніяк не допомогло у пошуку більш «хороших» блоків. З деякими

результати, отриманими даним алгоритмом можна ознайомитися за посиланням [30]. Для можливості вирішення цієї проблеми, було вирішено спробувати використати в якості цінової функції другий варіант, який описується в [27].

3.4.2 WCF

Функція визначається як [27]:

$$WCF = \sum_{y \in F_2^m} \sum_{x \in F_2^n} (\prod_{z \in C} \| W_f(x, y) | - z |) \tag{3.17}$$

Код, на мові програмування C++ надається далі [29]:

```
long long costWCF(int *sbox_d, int size, int count) {
    long long res = 0;
    long long calc = 1;
    int C [] = {0,4,8,12,16,20,24,28,32};
    int *sbox_b = SBoxToBooleanFunc(sbox_d,size,count);
    int *ar1 = linearCombinations(sbox_b, size, count);
    for (int i = 0; i < size - 1; ++i) {
        int *temp = calloc(size, sizeof(int));
        for (int j = 0; j < size; ++j) {
            temp[j] = ar1[i * size + j];
        }
        int *fxarr = HadamardCoefficients(temp, size, count);
        for (int j = 0; j < size; ++j) {
            for (int k = 0; k < 9; ++k){
                long long w = abs( _X: abs(fxarr[j]) - C[k]);
                calc = calc*w;
            }
            res += calc;
            calc = 1;
        }
        free(fxarr);
        free(temp);
    }
    free(ar1);
    free(sbox_b);
    return res;
}
```

Рисунок 3.17 – Програмний код цінової функції WCF

C визначається як $C = \{0,4,8,...,2^{\frac{n}{2}+1}\}$. Набір C містить всі можливі абсолютні значення, які є меншими або дорівнюють таким, завдяки яким може бути досягнуто максимальної нелінійності для бієктивного S-box'у. Функція заснована на принципі, що максимальне абсолютне значення спектру Уолша-Адамара не може

бути нижче $L \geq 2^{\frac{n}{2}+1}$, а коли це значення мінімальне, то всі абсолютні значення коефіцієнтів спектру містяться в наборі C.

Значення WCF не змінюється, якщо коефіцієнт спектру належить набору C, оскільки добуток, що включає абсолютне значення z ∈ C, дорівнює нулю. Отже, остаточне значення WCF виводиться з тих коефіцієнтів, абсолютне значення яких більше, ніж $L \geq 2^{\frac{n}{2}+1}$. Таким чином, мінімальне значення WCF досягається, коли максимальне абсолютне значення спектра Уолша-Адамара задовольняє рівнянню $L \geq 2^{\frac{n}{2}+1}$. В такому випадку WCF буде дорівнювати нулю. Далі буде показано залежність вартості блоків від їх нелінійності з використанням цінової функції:

Рисунок 3.18 – Залежність вартості блоків від їх нелінійності (WCF)

Як видно з графіку, тепер різниця у «вартості» блоків в залежності від нелінійності є дуже значною, і якщо блок дійсно є кращим за спектром Уолдша-Адамара, то його буде значно простіше знайти серед інших з використанням саме цієї цінової функції.

Використання цінової функції WCF також значно пришвидшило пошук блоків з нелінійністю 96-98, проте також не допомогло у пошуку більш «хороших»

блоків. З певними результати, отриманими даним алгоритмом, можна ознайомитися за посиланням [31].

3.4.3 PCF

Функція визначається як [28]:

$$PCF(S) = \sum_{i=0}^{N-1} 2^{-i} H(S)_{l-i} \tag{3.18}$$

Код, на мові програмування C++ надається далі [29]:

```
double costFunctionPCF(int *sbox_d, int size, int count) {
    int N = 10; long long calc = 1; short l = 0;
    int *H_S = calloc ( _NumOfElements: 24, sizeof(int));
    int *sbox_b = SBoxToBooleanFunc(sbox_d,size,count);
    int *ar1 = linearCombinations(sbox_b, size, count);
    for (int i = 0; i < size - 1; ++i) {
        int *temp = calloc(size, sizeof(int));
        for (int j = 0; j < size; ++j) {
            temp[j] = ar1[i * size + j];
        }
        int *fxarr = HadamardCoefficients(temp, size, count);
        for (int q = 0; q < size; ++q) {
            H_S[abs( _X: fxarr[q]/4)]++;
        }
        free(fxarr); free(temp);
    }
    free(ar1); free(sbox_b);
    for (int i = 0; i < 24; ++i){
        if (H_S[i] != 0){
            l = (short) i;
        }
    }
    double cost; double finCost = 0;
    for (int i = 0; i < N; ++i){
        if ((l-i) < 0){
            break;
        }
        cost = pow( _X: 2,-i)*H_S[l-i];
        finCost = finCost + cost;
    }
    free(H_S);
    return finCost;
}
```

Рисунок 3.19 – Програмний код цінової функції PCF

$H(S)$ є гістограмою абсолютних значень спектру Уолдша-Адамара. У виразі 3.18 з використанням множнику 2^{-i} встановлюється «важливість» елементів гістограми. З виразу видно, що коефіцієнти з максимально можливим абсолютним

значенням будуть помножені на $2^{-0}=1$. Це зробить ці коефіцієнти найбільш впливовими при обчисленні значення функції вартості.

Використання даної функції пришвидшило пошук блоків, навіть у порівнянні із функціями WHS та WCF, проте також не допомогло у пошуку блоку з нелінійністю більше 98, принаймні за розумний час.

3.5 Висновки до розділу

Певні задачі, наприклад, виконання синтезу S-боксів з високими показниками, потребують швидкого переміщення між підстановками. З цією задачею допоможе факторіальна система числення (ФСЧ), оскільки вона дозволяє отримати перестановку за номером, та навпаки, дуже швидко і просто.

У розділі наводяться алгоритми для роботи з ФСЧ – тобто для переведення номерів з десяткового представлення у факторіальне, а потім із факторіального у перестановку і навпаки. Також у розділі було надано програмний застосунок для роботи з перестановками розмірності 2^8, тобто довжини 256, яку мають S-блоки.

Такий підхід дозволив швидко переміщатися у необхідному просторі перестановок, а також збільшив випадковість отримуваних блоків, оскільки розроблений алгоритм не передбачає використання модульної логіки, як алгоритм з минулого розділу, а також не використовує завідома хороших блоків в якості цілі для пошуку.

Розроблений застосунок спирається тільки на отримані випадковим чином на фазі ініціалізації популяції блоки, а також на особистий досвід кожного блоку та досвід всього «рою» загалом. Це дозволяє блокам рухатися у просторі абсолютно випадковим чином, тому такі показники, як алгебраїчний імунітет, лінійна збитковість та дельта-рівномірність таких блоків будуть завжди на високому рівні. Що не можна сказати про нелінійність. Оскільки, хоч метод рою часток і є евристичним, проте він покладається в основному на дуже випадкові події та значення.

Отримані результати використання алгоритму на практиці свідчать про те, що алгоритм не є дуже ефективним, оскільки він не може отримати блоки з нелінійністю більш як 98.

Прагнення покращення алгоритмом пошукових можливостей хороших за спектром блоків, спонукало нас до використання трьох цінових функції: WHS, запропоновану Clark et al, WCF, запропоновану Alejandro Freyre-Echevarr´ıa та Ismel Mart´ınez-D´ıaz та PCF. Кожна з цих функцій в теорії підвищує точність вибору саме «хороших» блоків, оскільки враховує не один найбільший коефіцієнт, а декілька коефіцієнтів, які є наближеними до найбільшого.

Використання цих цінових функцій все ж значно пришвидшило пошук блоків з нелінійністю 96-98, проте також не допомогло у пошуку більш «хороших» блоків. Це можна пояснити величезним – 256! – пошуковим простором, особливо для методу, який є максимально ймовірнісним і залежить від багатьох випадкових подій.

Отже, після дослідження, реалізації та тестування в реальних умовах, можна сказати, що алгоритм PSO з використанням факторіальної системи числення та при роботі з простором блоків як з одновимірним простором, у якому кожен блок буде оцінюватися як номер у загальній кількості перестановок, є не дуже ефективним. Такий алгоритм не здатен отримувати блоки з високими показниками нелінійності за короткий час та з використанням звичайної комп’ютерної техніки. Нашою пропозицією є використання в майбутньому подібних алгоритмів для синтезу нелінійних вузлів заміни на квантових комп’ютерах, які мають значно вищу потужність пошуку, а отже будуть значно ефективнішими, особливо за таких складних умов.

4 ДОСЛІДЖЕННЯ ПРОСТОРУ НЕЛІНІЙНИХ ВУЗЛІВ ЗАМІН

У якості наочної демонстрації можливостей описаного у минулому розділі методу, а також для отримання детальної картини розміщення блоків у загальному просторі блоків, нами було виконано дослідження нелінійних вузлів заміни поряд з S-блоком, що використовується як основний криптопримітив у шифрі AES [19]. Застосування ФСЧ дозволило легко перейти від підстановки – блоку AES – до її номеру, а потім рухатися у просторі на певну кількість номерів уперед і назад.

4.1 Дослідження простору нелінійних вузлів замін поряд з AES-блоком

Блок шифру AES у десятковому табличному представленні матиме такий вигляд [18,20]:

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

Рисунок 4.1 – S-блок шифру AES

Показники такого блоку є наступними: нелінійність = 112, алгебраїчний імунітет = 2, лінійна збитковість = 254 та дельта-рівномірність = 4 [21].

Як видно, 2 з 4 показників є гарними, а інші 2 є незадовільними, тому такий блок є не найкращим варіантом для використання у реальних задачах. Використовуючи факторіальну СЧ, ми провели дослідження, в процесі якого було

знайдено параметри блоків, що віддалені на -100 та +100 за значенням номеру у просторі від цільового (AES) блоку. Отримані дані надані у вигляді гістограм:

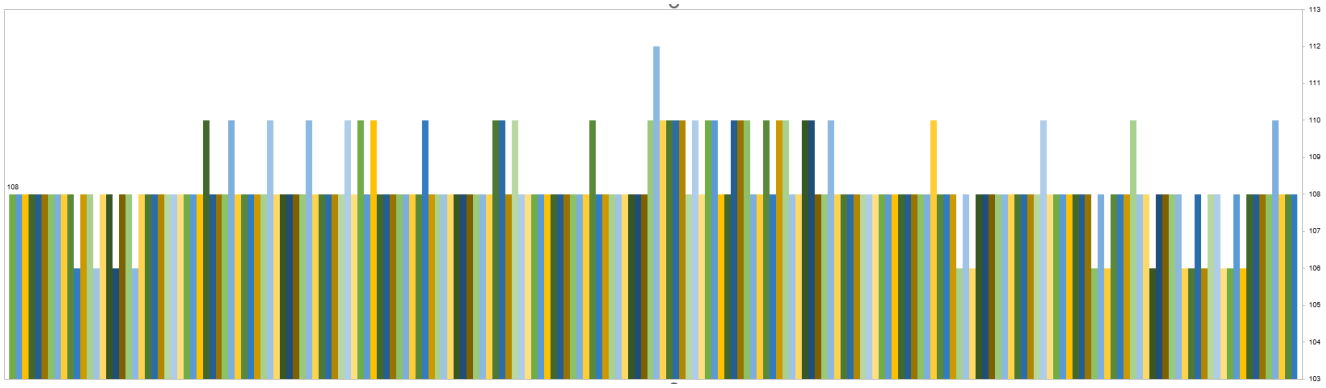

Рисунок 4.2 – Нелінійність блоків навколо AES-блоку

По осі *x* на рисунку 4.2 розміщені блоки діапазону [номер_блоку_AES-100 – номер_блоку_AES+100], а по осі *y* показана їхня нелінійність. Я видно з рисунку, більшість блоків по сусідству з блоком AES мають нелінійність 108, менша кількість 106 або 110 і тільки сам AES S-Box має нелінійність 112. Це показує, що у просторі S-блоків є зони, в яких є чітко виражений локальний максимум за нелінійністю, а поруч з ним знаходяться блоки зі схожими, але гіршими показниками.

На рисунку 4.3 такі ж результати надаються для лінійної збитковості:

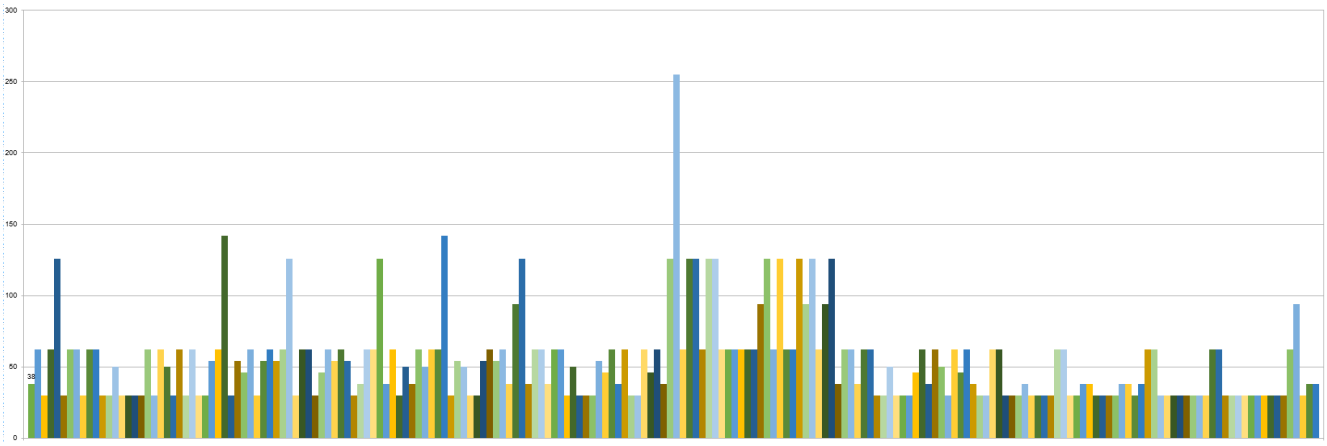

Рисунок 4.3 – Лінійна збитковість блоків навколо AES-блоку

По осі *x* розміщені блоки діапазону [номер_блоку_AES-100 – номер_блоку_AES+100], а по осі *y* показана їхня лінійна збитковість. Я видно з рисунку, лінійна збитковість більшості блоків не перевищує значення 100, інших

150, і тільки сам AES S-Box має найбільшу лінійну збитковість 254 – що є дуже поганим показником.

На рисунку 4.4 надаються результати для алгебраїчного імунітету:

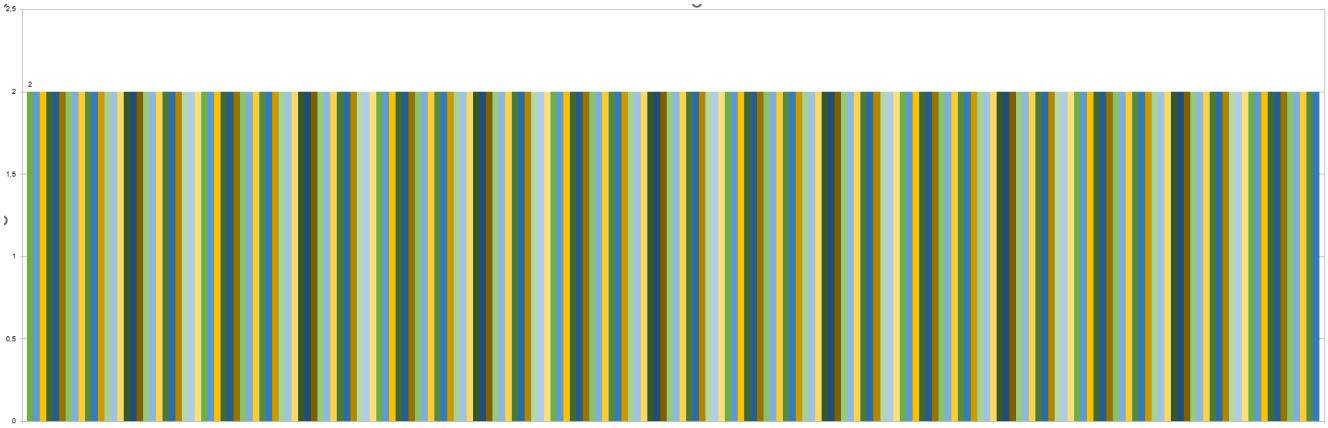

Рисунок 4.4 – Лінійна збитковість блоків навколо AES-блоку

Як і очікувалося імунітет всіх блоків дорівнює 2, оскільки такі блоки є не випадковими, а просто стоять поруч з блоком знайденим алгебраїчним методом.

На рисунку 4.5 надаються результати для дельта-рівномірності:

Рисунок 4.5 – Дельта-рівномірність блоків навколо AES-блоку

Як видно з рисунку дельта-рівномірність майже всіх блоків дорівнює 6, і тільки для невеликої кількості 4, як і в самого AES S-Box’у.

Такі результати свідчать про те, що поблизу блоків з хорошими показниками знаходяться блоки зі схожими показниками, тому використання методу Particle Swarm Optimization є доцільним при роботі з S-блоками, оскільки його концепція саме і полягає в пошуку певних кращих значень поблизу схожих хороших значень.

Виходячи з отриманих результатів помітно, що даний метод не є досить ефективним при такому величезному просторі, як для нелінійних вузлів заміни

розмірності 2^8. Завдяки використанню Particle Swarm Optimization, який працював з номерами блоків у просторі, вдалося отримати максимальне значення нелінійності лише 98. Такий поганий результат може бути пов'язаний з розташуванням блоків у просторі і дуже великими розмірами цього простору, тобто, мається на увазі, що блоки з гарними значеннями нелінійності, алгебраїчного імунітету та інших показників знаходяться дуже близько один до одного і зустрічаються у просторі дуже рідко.

Тому такий метод як Particle Swarm Optimization не буде ефективним, якщо просто пересувати популяцію блоків до кращих знайдених блоків, оскільки щоб знайти хороший блок таким чином потрібні величезні популяції в мільйони або мільярди блоків. Використання таких розмірів популяцій буде недоцільним і з точки зору обчислювальних потужностей, і з точки зору швидкодії.

У наступному пункті буде проведено схоже дослідження, але уже з використанням цінових функцій, які, як було показано в минулому розділі, дають значну різницю у «вартості» блоків в залежності від хорошості спектру блоку [20].

4.2 Дослідження простору нелінійних вузлів замін поряд з AES-блоком з використанням цінових функцій

Для демонстрації доцільності використання цінових функцій (ЦФ, cost-functions) у алгоритмах пошуку блоків будь-якими методами, було прийнято дослідити простір блоків, які лежать поряд із блоком, який застосовується у алгоритмі AES [21].

Такі дослідження показують, що оцінка за ціновими функціями допомагає значним чином упорядкувати простір блоків, навідміну від оцінки з використанням лише нелінійності. Упорядкування дозволяє отримати більш чітку картину розміщення блоків, до того ж при такому упорядкуванні більш чітко видно мінімуми та максимуми за необхідними характеристиками, а отже, таким чином значно простіше шукати «хороші» блоки та уникати «поганих».

У даному пункті будуть наведені графіки, для демонстрації більшої впорядкованості простору, при використанні оцінки блоків за допомогою цінової функції. Таблиці з даними, отриманими для кожної цінової функції для блоків -30

та +30 відносно S-блоку AES надаються у додатку В. Більш детальні дослідження, та повна таблиця порівнянь для -100 та +100 блоків доступна у [21].

Далі будуть наведені графіки [21] для діапазону [номер_блоку_AES-100 – номер_блоку_AES+100], для кожної з використаних цінових функцій (рис 4.6-4.9), на яких буде показано як обраний метод оцінки блоків впливає на впорядкованість простору пошуку.

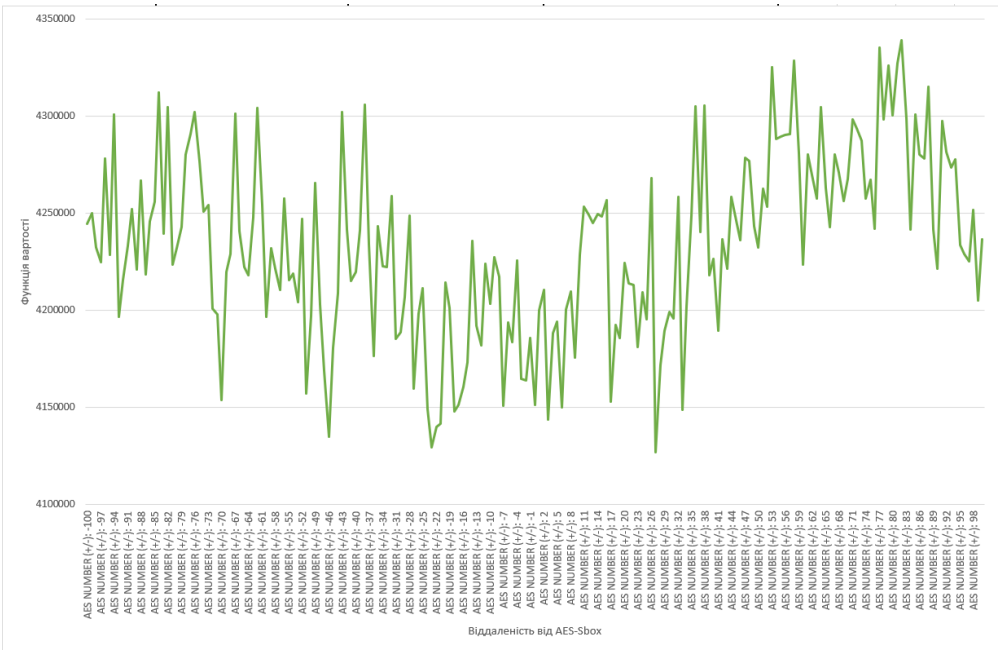

Рисунок 4.6 – Графік значень вартості блоків поруч з AES з використанням WHS функції

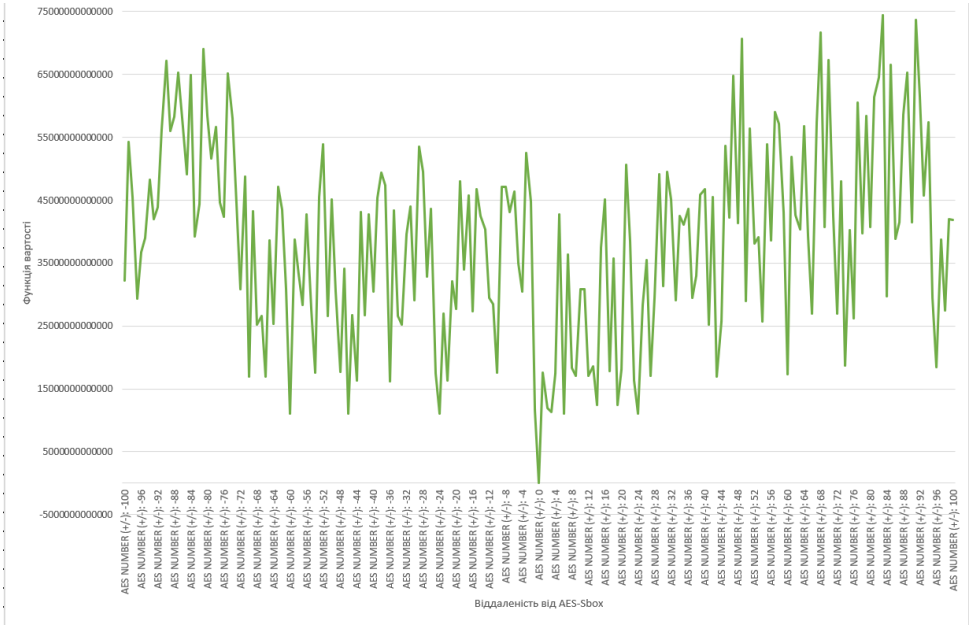

Рисунок 4.7 – Графік значень вартості блоків поруч з AES з використанням WCF функції

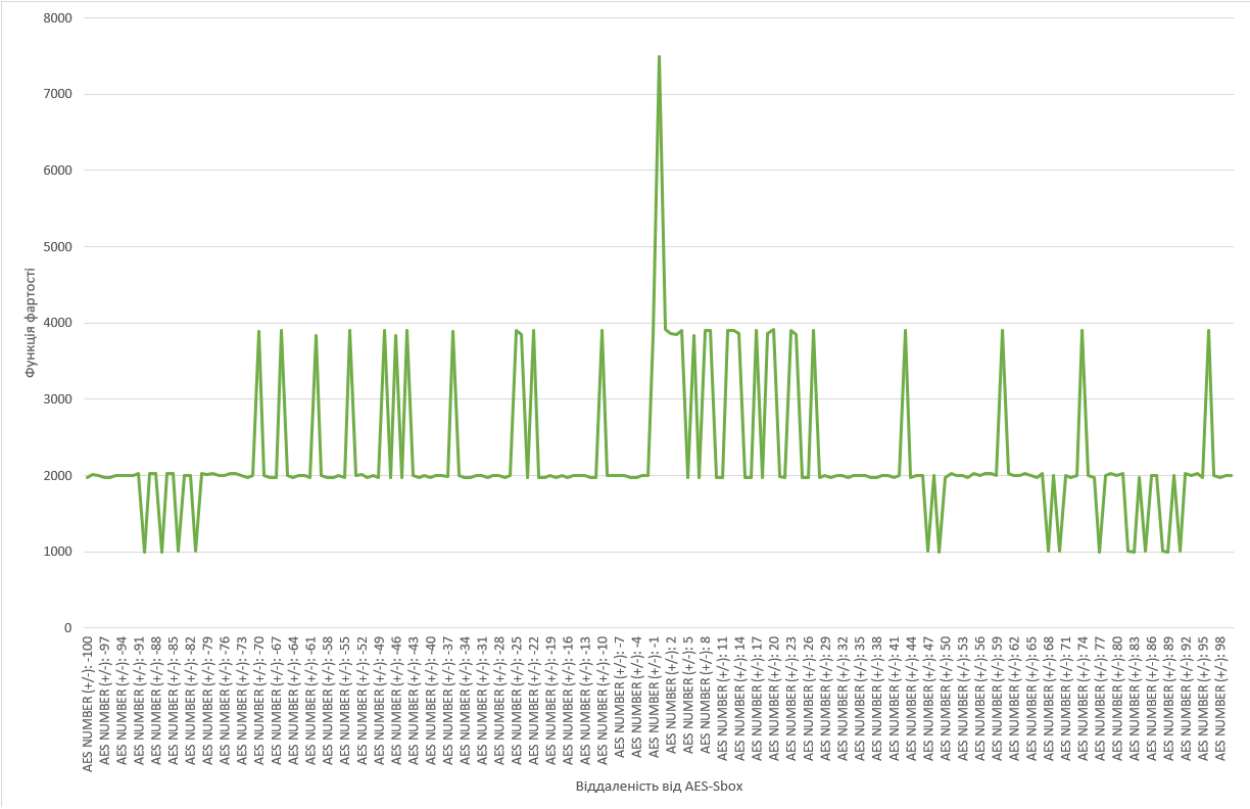

Рисунок 4.8 – Графік значень вартості блоків поруч з AES з використанням PCF функції

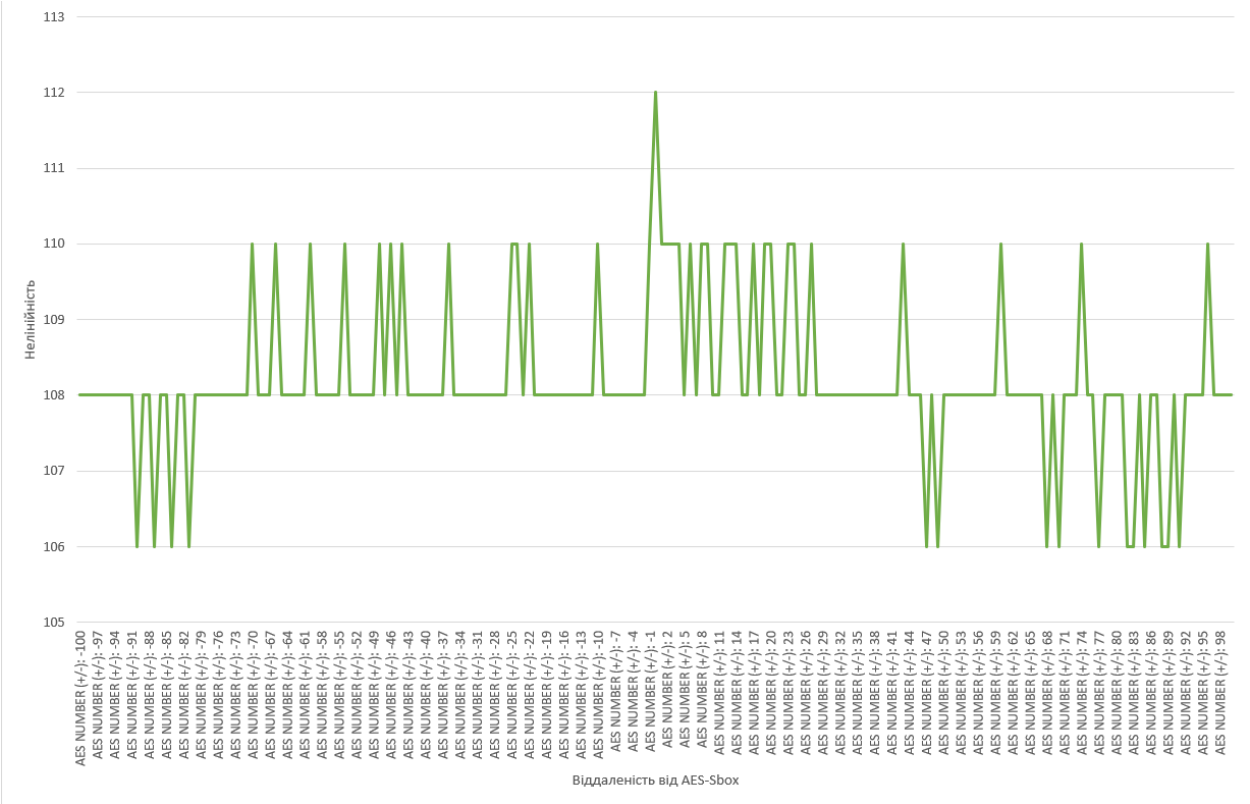

Рисунок 4.9 – Графік значень нелінійності блоків поруч з AES

Така демонстрація є необхідною, оскільки візуалізація пошукового простору може допомогти краще бачити поведінку та концепцію розміщення блоків у просторі, а отже, це має бути корисним при розробці нових методів пошуку.

Як можна бачити з отриманих результатів, певні цінові функції дійсно значно покращують розпізнавання блоків у просторі, наприклад WHS [26] та WCF [27] набагато чіткіше виражають локальні мінімуми та максимуми, в яких алгоритм повинен шукати блоки та оминати відповідно. Це вказує на те, що в теорії використання cost-функцій має покращити процес відбору блоків, а отже і пришвидшити процес пошуку, оскілки алгоритм працюватиме одразу зі блоком, який матиме хороший спектр, а не тільки найбільший з коефіцієнтів.

4.3 Висновки до розділу

Метою розділу було отримання детальної картини розміщення блоків у загальному просторі блоків. Для цього нами було виконано дослідження нелінійних вузлів заміни поряд з S-блоком, що використовується як основний криптопримітив у шифрі AES. Спочатку були визначені показники самого AES S-блоку, і виявилося, що 2 з 4 показників у такого блоку є гарними, проте інші 2 є незадовільними. Це каже про те, що такий блок є не найкращим варіантом для використання у реальних задачах.

Використовуючи факторіальну СЧ, ми провели дослідження, в процесі якого було знайдено показники блоків, що віддалені на -100 та +100 за значенням номеру у просторі від цільового (AES) блоку. Отримані результати свідчать про те, що у просторі поблизу блоків з хорошими показниками знаходяться блоки зі схожими показниками, тому застосування методу PSO може стати у нагоді під час роботи з S-блоками або при виконання задачі синтезу нових блоків. Це пов’язано з тим, що концепція цього методу полягає в пошуку певних кращих значень поблизу схожих хороших значень.

Аналізуючи отримані результати помітно, що даний метод буде не дуже ефективним при такому величезному просторі, як для нелінійних вузлів заміни розмірності 2^8, оскільки рух в такому просторі, тим паче з імовірнісним впливом, не зможе покрити всі можливі хороші варіанти при пошуку, а отже буде втрачати

результативність з кожним кроком і рухаючись у випадковому напрямку, минати на шляху блоки з дійсно гарними показниками. Аналіз також показав, що блоки з гарними значеннями нелінійності, алгебраїчного імунітету та інших показників знаходяться дуже близько один до одного і зустрічаються у просторі дуже рідко, що ще більше ускладнюватиме пошук.

Для демонстрації доцільності використання цінових функцій у алгоритмах пошуку блоків з використанням різноманітних методів, було досліджено простір блоків, які лежать поряд із блоком, який застосовується у алгоритмі AES, з використанням цих функцій. Отримані результати показують, що певні цінові функції дійсно значно покращують розпізнавання блоків у просторі, наприклад WHS та WCF набагато чіткіше виражають локальні мінімуми та максимуми, в яких алгоритм повинен шукати блоки та оминати відповідно.

Для певних алгоритмів використання cost-функцій дійсно може покращити процес відбору блоків, а отже і пришвидшити процес пошуку. Проте, як було показано у минулому розділі, використання цих цінових функцій для PSO, хоча і пришвидшує пошук блоків з нелінійністю 96-98, проте не допомагає знайти більш хороші блоки. І хоча локальні мінімуми та максимуми є більш вираженими, алгоритм не може осягнути всі місця у такому величезному просторі.

ВИСНОВКИ

1) Нелінійні вузли заміни (S-блоки, нелінійні підстановки) є одним з головних компонентів сучасних блокових симетричних криптоперетворень. Показники захищеності таких криптопримітивів безпосередньо впливають на ефективність криптографічних перетворень, в яких вони застосовуються. У першому розділі роботи було представлено зразок програмного застосунку для розрахунку цих показників для блоків у десятковому та шістнадцятирічному представленнях.

2) Аналіз методів генерації блоків демонструє, що найбільш ефективними з точки зору нелінійності сформованого блоку є алгебраїчні методи синтезу. Головним недоліком цих методів є те, що вони не здатні генерувати повністю випадкові блоки, що є потенційною загрозою для алгоритму, в якому такий блок буде використовуватися. Алгебраїчні методи можуть надати блоки, що будуть хорошими тільки за показником нелінійності, у порівнянні з іншими методами, проте всі інші параметри будуть гіршими, а іноді і незадовільними. Зважаючи на це, було вирішено обрати основою для дослідження та подальшої розробки евристичні методи генерації блоків, а саме метод Particle Swarm Optimization.

3) Аналіз відкритих наукових джерел показав, що спроби застосування методу рою часток (PSO) до роботи з S-блоками були, проте вони не мали успіху.

4) У другому розділі роботи було показано реалізацію програмного застосунку для генерації блоків з використанням як базового методу, що пропонується в одній з недавніх наукових робіт, так і модифікованого методу. Результати, показують, що модифікований алгоритм дійсно спроможний отримувати блоки з заданими параметрами, навідміну від базового.

5) Одним з недоліків розробленого алгоритму є те, що S-блок, отриманий з його застосування, є доволі схожим на AES S-box, але все ж такий блок задовольняє всім необхідним критеріям і може використовуватися у БСШ чи інших криптографічних перетвореннях.

6) У третьому розділі роботи для забезпечення можливості швидкого переміщення між підстановками при виконанні задачі синтезу блоків з використанням PSO було використано факторіальну систему числення. ФСЧ дозволила швидко отримувати перестановку за номером, та робити зворотну операцію дуже швидко і просто. Це в свою чергу дало можливість представлення простору перестановок у вигляді одновимірного простору (за номером перестановки у загальному просторі всіх можливих перестановок).

7) Представлення у вигляді одновимірного простору дозволило швидко переміщатися між підстановками, а також збільшило випадковість отримуваних блоків, оскільки завдяки такому представленню розроблений алгоритм не використовує жодних модульних операцій та модульної логіки для зміни положення блоків у просторі. Випадковість також отримується за рахунок відмови від використання завідома хороших блоків в якості цілі, що впливатиме на пошук.

8) Розроблений програмний застосунок для роботи з PSO з використанням ФСЧ, спирається тільки на випадкові блоки на початку роботи, а також на особистий досвід кожного блоку та досвід всього «рою» загалом після першої ітерації алгоритму оптимізації. Такий механізм надає блокам можливість руху у просторі тільки на основі власного досвіду, без використання завідома хороших блоків. Це дозволяє зберегти такі показники, як алгебраїчний імунітет, лінійну збитковість та дельта-рівномірність блоків на високому рівні.

9) Отримані результати використання алгоритму на практиці показують, що алгоритм PSO з використанням факторіальної системи числення та при роботі з простором блоків як з одновимірним простором, у якому кожен блок (перестановка) буде оцінюватися як номер у загальній кількості перестановок, є не дуже ефективним, оскільки він не може отримати блоки з нелінійністю більш як 98, принаймні за розумний час та з використанням звичайних комп’ютерів.

10) Прагнення покращення алгоритмом пошукових можливостей хороших за спектром блоків, спонукало нас до використання трьох цінових функції: WHS, WCF, та PCF. Принцип роботи цих функцій полягає у підвищенні точності вибору «хороших» за спектром блоків, оскільки функція бере до уваги не тільки

найбільший коефіцієнт спектру, а декілька коефіцієнтів, які є наближеними до найбільшого. Використання цінових функцій пришвидшило пошук блоків з нелінійністю 96-98, проте не допомогло у пошуку більш «хороших» блоків. Це пояснюється величезним, як для методу, який є максимально ймовірнісним і залежить від багатьох випадкових подій, пошуковим простором.

11) Спираючись на отримані у третьому пункті результати, нашою пропозицією є використання в майбутньому подібних алгоритмів для синтезу нелінійних вузлів заміни на квантових комп'ютерах. Такі комп'ютери дозволять виконувати мільярди операцій пошуку за одиницю часу, а отже будуть значно ефективнішими, особливо за таких складних умов пошуку.

12) У четвертому розділі, використовуючи факторіальну систему числення, ми провели дослідження, в процесі якого було знайдено показники блоків, що віддалені на -100 та +100 за значенням номеру у просторі від цільового (AES) блоку. Отримані результати свідчать про те, що у просторі поблизу блоків з хорошими показниками знаходяться блоки зі схожими показниками.

13) Аналіз також показав, що блоки з гарними значеннями нелінійності, алгебраїчного імунітету та інших показників знаходяться дуже близько один до одного і зустрічаються у просторі дуже рідко, що ще більше ускладнюватиме пошук, особливо з використанням евристичних або імовірнісних методів пошуку.

14) Результати перевірки доцільності використання цінових функцій у алгоритмах пошуку показують, що певні цінові функції дійсно значно покращують розпізнавання блоків у просторі, наприклад WHS та WCF набагато чіткіше виражають локальні мінімуми, в яких алгоритм повинен шукати блоки. І хоча локальні мінімуми та максимуми є більш вираженими при використанні цінових функцій, такі методи, як метод рою часток чи подібні до нього, не може осягнути всі місця у такому величезному просторі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Kazuo Sakiyama, Yu Sasaki, Yang Li. Security of Block Ciphers: From Algorithm Design to Hardware Implementation. URL: https://www.researchgate.net/publication/288251231_Security_of_Block_Ciphers_From_Algorithm_Design_to_Hardware_Implementation (дата звернення: 25.09.2022).
2. Kamsiah Mohamed et al. Study of S-box Properties in Block Cipher. 2014. URL: https://www.researchgate.net/publication/277613994_Study_of_S-box_Properties_in_Block_Cipher (дата звернення: 25.09.2022).
3. Howard M. Heys. A Tutorial on Linear and Differential Cryptanalysis. 2001. URL: http://www.cs.bc.edu/~straubin/crypto2017/heys.pdf (дата звернення: 25.09.2022).
4. Казимиров Александр Владимирович. Методы и средства генерации нелинейных узлов замены для симметричных криптоалгоритмов. 2013. URL: https://okazymyrov.github.io/assets/attachments/theses/2014/daaa5576d6c6ecb227c6ae888cdce3.pdf (дата звернення: 28.09.2022).
5. Dragan Lambić and Miodrag Živković. Comparison of random s-box generation methods. 2013. URL: http://www.doiserbia.nb.rs/img/doi/0350-1302/2013/0350-13021307109L.pdf (дата звернення: 30.09.2022).
6. A.J. Clark, Optimisation heuristics for cryptology, phd, Queensland University of Technology, 1998. URL: https://eprints.qut.edu.au/15777/ (дата звернення: 30.09.2022).
7. ZHANG Yu-An FENG Deng-Guo. Equivalent Generation of the S-box of RIJNDAEL. 2004. URL: http://cjc.ict.ac.cn/eng/qwjse/view.asp?id=1648 (дата звернення: 30.09.2022).
8. T. M. Shami, A. A. El-Saleh, M. Alswaitti, Q. Al-Tashi, M. A. Summakieh and S. Mirjalili. Particle Swarm Optimization: A Comprehensive Survey. 2022. URL:

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9680690 (дата звернення: 04.10.2022).

9. Y. Xiaojing, J. Qingju, L. Xinke. Center Particle Swarm Optimization Algorithm. 2019. URL: https://doi.org/10.1109/ITNEC.2019.8729510 (дата звернення: 04.10.2022).
10. A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, P.C. van Oorschot, S.A. Vanstone. Handbook of Applied Cryptography. 2018. URL: https://doi.org/10.1201/9780429466335 (дата звернення: 07.10.2022).
11. José Antonio Álvarez-Cubero. Vector Boolean Functions: applications in symmetric cryptography. 2015. URL: https://www.researchgate.net/publication/325428613_Vector_Boolean_Functions_applications_in_symmetric_cryptography (дата звернення: 07.10.2022).
12. J. McLaughlin. Applications of search techniques to cryptanalysis and the construction of cipher components. 2012. URL: http://etheses.whiterose.ac.uk/3674/ (дата звернення: 07.10.2022).
13. M. Ahmad, I.A. Khaja, A. Baz, H. Alhakami, W. Alhakami. Particle Swarm Optimization Based Highly Nonlinear Substitution-Boxes Generation for Security Applications. 2020. URL: https://ieeexplore.ieee.org/document/9123381 (дата звернення: 10.10.2022).
14. A. Kuznetsov, Y. Derevianko, V. Myroshnychenko and O. Bulhakova. Research of the Particle Swarm Method for Generating Nonlinear Substitutions. 2021. URL: https://ieeexplore.ieee.org/document/9716660 (дата звернення: 10.10.2022).
15. Kuznetsov, A.; Derevianko, Y.; Poluyanenko, N.; Bagmut, O. Particle Swarm Optimization based on S-Boxes Generation. 2021. URL: http://ceur-ws.org/Vol-3188/paper12.pdf (дата звернення: 10.10.2022).
16. Дерев'янко Я.А., Горбенко І.Д., Кузнецов О.О. Метод рою часток для генерації нелінійних підстановок. 2021. URL: http://www.viti.edu.ua/files/zbk/2021/c_2021.pdf (дата звернення: 10.10.2022).
17. J. Kennedy, R. Eberhart. Particle swarm optimization. 1995. URL: https://ieeexplore.ieee.org/document/488968 (дата звернення: 10.10.2022).

18. Дерев’янко Я.А. Реалізація методу Particle Swarm Optimization для роботи з S-блоками. 2021. URL: https://github.com/DereviankoYaroslav/SBoxDereviankoC11 (дата звернення: 10.10.2022).
19. J. Daemen, V. Rijmen. Rijndael/AES. 2005. URL: https://link.springer.com/referenceworkentry/10.1007/0-387-23483-7_35 (дата звернення: 10.10.2022).
20. O. Kazymyrov, V. Kazymyrova, R. Oliynykov. A Method For Generation Of High-Nonlinear S-Boxes Based On Gradient Descent. 2013. URL: https://eprint.iacr.org/2013/578 (дата звернення: 10.10.2022).
21. Я.А. Дерев’янко, Ю.І. Горбенко, О.О. Кузнецов. Факторіальна система числення для генерації нелінійних підстановок. 2022. URL: https://doi.org/10.30837/rt.2022.2.209.04 (дата звернення: 17.10.2022).
22. Donald E. Knuth. The Art of Computer Programming: *Sorting and Searching*. 1998. URL: https://seriouscomputerist.atariverse.com/media/pdf/book/Art%20of%20Computer%20Programming%20-%20Volume%203%20(Sorting%20&%20Searching).pdf (дата звернення: 17.10.2022).
23. Дерев’янко Я.А. Реалізація алгоритмів для роботи з факторіальною системою числення. 2022. URL: https://github.com/DereviankoYaroslav/SBoxDereviankoFactorial (дата звернення: 17.10.2022).
24. NTL: A Library for doing Number Theory. 2022. URL: https://libntl.org/ (дата звернення: 17.10.2022).
25. Дерев’янко Я.А. Реалізація методу Particle Swarm Optimization з використанням факторіальної системи числення. 2022. URL: https://github.com/DereviankoYaroslav/FactorialPSO-NL (дата звернення: 17.10.2022).
26. John A Clark, Jeremy L Jacob, and Susan Stepney. Searching for cost functions. 2004. URL:

https://www.researchgate.net/publication/4090079_Searching_for_cost_functions (дата звернення: 17.10.2022).

27. Alejandro Freyre-Echevarr´ıa, Ismel Mart´ınez-D´ıaz. A new cost function to improve nonlinearity of bijective S-boxes. 2020. URL: https://www.researchgate.net/publication/343699912_A_new_cost_function_to_improve_nonlinearity_of_bijective_S-boxes (дата звернення: 17.10.2022).
28. Stjepan Picek, Marko Čupić, Leon Rotim. A New Cost Function for Evolution of S-boxes. 2016. URL: https://www.researchgate.net/publication/305801738_A_New_Cost_Function_for_Evolution_of_S-boxes (дата звернення: 17.10.2022).
29. Дерев'янко Я.А. Реалізація цінових функцій WHS, WCF, PCF. 2021. URL: https://github.com/DereviankoYaroslav/costFuncs (дата звернення: 17.10.2022).
30. Результати застосування PSO з використанням цінової функції WHS. 2022. URL: https://drive.google.com/drive/u/0/folders/15WfuMY6dT-LHBoIdCbLrJXSctRCJki4P (дата звернення: 17.10.2022).
31. Результати застосування PSO з використанням цінової функції WCF. 2022. URL: https://drive.google.com/drive/u/2/folders/1scDiZjAkcCiz_a3IkxvuNwfAIWP6d6P- (дата звернення: 17.10.2022).

ДОДАТОК А

Псевдокод модифікованого алгоритму PSO

```
Вхідні значення:
N ← число блоків в популяції
maxItr ← максимальна кількість ітерацій
PSO_mode ← вибір режиму алгоритму (0 – алгоритм зі статті, 1 – модифікований)
_____________________________________________________
Генерація початкової популяції S-box'ів:
srand(time(NULL));
int check = rand()%size;
swarm ← zeros(2 × N, 256) // 256 для 8 × 8 S-box'ів
for i ← 1 to N
        [sbox, xr] ← gen_sbox(i+ check)
Генерація відбувається з використанням rand() та перемішування Фішера-Йейтса
        swarm [i] ← sbox
end for
swarm [1] ← aes_sbox
_____________________________________________________
Обчислення нелінійності для кожної з часток:
for i ← 1 to N do:
NL[i] ← нелінійність(swarm[i])
end for
NL_відсортований ← sort(NL) // сортування у порядку спадання
gBest ← swarm[1]
pBesti ← swarm [i]
swarmVel ← zeros(N, 256)
Встановлення інерційної ваги w
```

Рисунок А.1 – Псевдокод модифікованого алгоритму PSO

```
_____________________________________________
Початок фази покращення:
While (max_itr > 0) do:
        Застосування формули (2.5) для розрахунку ваги
        c_1 ← 2*rand() [0,1]
        c_2 ← 2*rand() [0,1]
        r_1 ← rand() [0,1]
        r_2 ← rand() [0,1]
        NL ← NL_відсортований
        for i ← 1 to N do:
                for j = 1 to 256 do:
                        if (PSO_mode == 1)
                        swarmVel[i][j] ← ceil(w*swarmVel[i][j] + c_1*r_1*(gBest[i][j] -
                        swarm[i][j]) + c_2*r_2*(gBest[j]- swarm[i][j]))
                        end if
                        if (PSO_mode == 0)
                        swarmVel[i][j] ← ceil(w*swarmVel[i][j] + c_1*r_1*(pBest[i][j] -
                        swarm[i][j]) + c_2*r_2*(gBest[j]- swarm[i][j]))
                        end if
                        if (swarmVel[i][j] < 0)
                                swarmVel[i][j] ← (swarmVel[i][j] + 256)mod(256)
                        end if
                        X[i, j] ← int(swarm[i][j] + swarmVel[i][j])mod(256)
                        temp_sbox[j] ← X[i, j]
                end for
```

Рисунок А.2 – Продовження псевдокоду модифікованого алгоритму PSO

```
                Застосування наведеного вище алгоритму для збереження бієктивності
                if (i == 0)
                        int LAT = LATMax(temporarySbox, sbox_size, count);
                        int NL = raiseToPower(2, count - 1) - LAT;
                        if (NL > 98)
                                for (int k = 0; k < 15; ++k)
                                        srand(temporarySbox[k] * (curIter * k) % 256);
                                        int coeff1 = rand() % 50;
                                        printf("coeff1 %d", coeff1);
                                        int coeff2 = rand() % 256;
                                        printf("coeff2 %d", coeff2);
                                        int temp = tempSbox[coeff1];
                                        temporarySbox[coeff1] = temporarySbox[coeff2];
                                        temporarySbox[coeff2] = temp;
                                end for
                        end if
                end if
                swarm[N + i] ← temp_sbox
        end for
```

Рисунок А.3 – Продовження псевдокоду модифікованого алгоритму PSO

```
        for i ← 1 to N do:
                NL_відсортований[N + i] ← нелінійність(swarm[N + i])
        end for
        NL_відсортований ← sort(NL_відсортований)
        Відкидання зайвих у популяції елементів згідно з нелінійністю
        if (current_Iter == 0)
                        for (int q = 0; q < size; ++q)
                                gBest[q] = swarm[0][q];
                end for
                for (int i = 1; i < N; ++i)
                                for (int j = 0; j < size; ++j)
                                        pBest[i - 1][j] = swarm[i][j];
                                end for
                end for
        end if
        else
                for (int q = 0; q < size; ++q)
                                gBest[q] = swarm[1][q];
                end for
                for (int i = 2; i < N; ++i)
                                for (int j = 0; j < size; ++j)
                                        pBest[i - 1][j] = swarm[i][j];
                                end for
                end for
        end else
        Оцінка всіх блоків в популяції на відповідність необхідним параметрам
        Зміна gBest, якщо такий знайдено
        maxItr ← maxItr – 1
        PSO_mode = 0
        ++current_Iter
end while
Запис кінцевої популяції блоків у файл
```

Рисунок А.4 – Продовження псевдокоду модифікованого алгоритму PSO

ДОДАТОК Б

Результати експериментальних досліджень руху популяції

Таблиця Б.1 – Рух однієї з частинок популяції за 20 ітерацій

Номер блоку	Iter	NL
6375481545267114037838733209081408394422188113065421098753296981247581221085292063779432687751212781839080706608816322516365880536515386994290527304724911378099784764768708515458634530730396194872559968972396480836284245580507728943503908112405664989958404574737634114386843419126633228223574095361916460808983077643816142719075547416894932806109050882470385024231224193319647371925292885922682951433968953204734788429607416265779235535390250724021161414788131845766151295329887050812303284077541967782172276615224972748278536204129340813864094061322886526584460452307727098089187762702019812183716423083698040766963547137125842419307210882850589013032431089475582013529882937965300018726151348748643637431136859996152668107314706014667101090542402743395570310191209095135218805833972102976237983335786495339279756131858226602244582286964840708010237130364619632879696797722479804618681670024787651979105744303902308986434338457451216253661928813852937513627934928242345006432449656440654759818779467233823	1	94
5129936652325249727482785362041293408138640940613228865265844604523077270980891877627020198121837164230836980407669635471371258424193072108828505890130324310894755820135298829379653000187261513487486436374311368599961526681073147060146671010905424027433955703101912090951352188058339721029762379833357864953392797561318582266022445822869648407080102371303646196328796967977224798046186816700247876519791057443039023089864343384574512162536619288138529375136279349282423450064324496564406547598187794672338231	2	94

Продовження таблиці Б.1 – Рух однієї з частинок популяції за 20 ітерацій

Номер блоку	Iter	NL
512993665232524972748278536204129340813864094061322886526584460452307727098089187762702019812183716423083698040766963547137125842419307210882850589013032431089475582013529882937965300018726151348748643637431136859996152668107314706014667101090542402743395570310191209095135218805833972102976237983335786495339279756131858226602244582286964840708010237130364619632879696797722479804618681670024787651979105744303902308986434338457451216253661928813852937513627934928242345006432449656440654759818779467233823	3	94
512993665232524972748278536204129340813864094061322886526584460452307727098089187762702019812183716423083698040766963547137125842419307210882850589013032431089475582013529882937965300018726151348748643637431136859996152668107314706014667101090542402743395570310191209095135218805833972102976237983335786495339279756131858226602244582286964840708010237130364619632879696797722479804618681670024787651979105744303902308986434338457451216253661928813852937513627934928242345006432449656440654759818779467233823	4	96
512993665232524972748278536204129340813864094061322886526584460452307727098089187762702019812183716423083698040766963547137125842419307210882850589013032431089475582013529882937965300018726151348748643637431136859996152668107314706014667101090542402743395570310191209095135218805833972102976237983335786495339279756131858226602244582286964840708010237130364619632879696797722479804618681670024787651979105744303902308986434338457451216253661928813852937513627934928242345006432449656440654759818779467233823	5	96

Продовження таблиці Б.1 – Рух однієї з частинок популяції за 20 ітерацій

Номер блоку	Iter	NL
5129936652325249727482785362041293408138640940613228865265844604523077270980891877627020198121837164230836980407669635471371258424193072108828505890130324310894755820135298829379653000187261513487486436374311368599961526681073147060146671010905424027433955703101912090951352188058339721029762379833357864953392797561318582266022445822869648407080102371303646196328796967977224798046186816700247876519791057443039023089864343384574512162536619288138529375136279349282423450064324496564406547598187794672338235033273047946281296915532549549822596246485866688655739789291825307076827450499853784666961857343352824193021208399705555400581583849875760418895344090504484693319615821849966114386499874488099357132719682103902736151152474981803026973007012270580505475912508040821514462275857176727302506489682025544998907395497458894619315210432025727317102766574427201285996449461999934395977477775245144604415489376669300352858978410820089929485916072887374663935556556596498384851524858267988158816027945468633625743235033273047946281296915532549549822596246485866688655739789291825307076827450499853784666961857343352824193021208399705555400581583849875760418895344090504484693319615821849966114386499874488099357132719682103902736151152474981803026973007012270580505475912508040821514462275857176727302506489682025544998907395497458894619315210432025727317102766574427201285996449461999934395977477775245144604415489376669300352858978410820089929485916072887374663935556556596498384851524858267988158816027945468633625743235

Продовження таблиці Б.1 – Рух однієї з частинок популяції за 20 ітерацій

Номер блоку	Iter	NL
503327304794628129691553254954982259624648586668865573978 929182530707682745049985378466696185734335282419302120839 970555540058158384987576041889534409050448469331961582184 996611438649987448809935713271968210390273615115247498180 302697300701227058050547591250804082151446227585717672730 250648968202554499890739549745889461931521043202572731710 276657442720128599644946199993439597747777524514460441548 937666930035285897841082008992948591607288737466393555655 659649838485152485826798815881602794546863362574323	9	96
503327304794628129691553254954982259624648586668865573978 929182530707682745049985378466696185734335282419302120839 970555540058158384987576041889534409050448469331961582184 996611438649987448809935713271968210390273615115247498180 302697300701227058050547591250804082151446227585717672730 250648968202554499890739549745889461931521043202572731710 276657442720128599644946199993439597747777524514460441548 937666930035285897841082008992948591607288737466393555655 659649838485152485826798815881602794546863362574323	10	96
503327304794628129691553254954982259624648586668865573978 929182530707682745049985378466696185734335282419302120839 970555540058158384987576041889534409050448469331961582184 996611438649987448809935713271968210390273615115247498180 302697300701227058050547591250804082151446227585717672730 250648968202554499890739549745889461931521043202572731710 276657442720128599644946199993439597747777524514460441548 937666930035285897841082008992948591607288737466393555655 659649838485152485826798815881602794546863362574323	11	96

Продовження таблиці Б.1 – Рух однієї з частинок популяції за 20 ітерацій

Номер блоку	Iter	NL
503327304794628129691553254954982259624648586668865573978 929182530707682745049985378466696185734335282419302120839 970555540058158384987576041889534409050448469331961582184 996611438649987448809935713271968210390273615115247498180 302697300701227058050547591250804082151446227585717672730 250648968202554499890739549745889461931521043202572731710 276657442720128599644946199993439597747777524514460441548 937666930035285897841082008992948591607288737466393555655 659649838485152485826798815881602794546863362574323	12	96
503327304794628129691553254954982259624648586668865573978 929182530707682745049985378466696185734335282419302120839 970555540058158384987576041889534409050448469331961582184 996611438649987448809935713271968210390273615115247498180 302697300701227058050547591250804082151446227585717672730 250648968202554499890739549745889461931521043202572731710 276657442720128599644946199993439597747777524514460441548 937666930035285897841082008992948591607288737466393555655 659649838485152485826798815881602794546863362574323	13	96
503327304794628129691553254954982259624648586668865573978 929182530707682745049985378466696185734335282419302120839 970555540058158384987576041889534409050448469331961582184 996611438649987448809935713271968210390273615115247498180 302697300701227058050547591250804082151446227585717672730 250648968202554499890739549745889461931521043202572731710 276657442720128599644946199993439597747777524514460441548 937666930035285897841082008992948591607288737466393555655 659649838485152485826798815881602794546863362574323	14	96

Продовження таблиці Б.1 – Рух однієї з частинок популяції за 20 ітерацій

Номер блоку	Iter	NL
4907602820766044260332130797687377232671215028937673307949782967000949331612865034664813658692569167292918641382867152131441244102598379901161243030643258732747937676027870227219483808467807692197133989299182133743742608450556174533241861608182533485168504700011161742369128974242093881001422435301090352641618640951167378967753904389786060527863816292978465773577201194632981397854604011675116620401052137007011949228226063041823310753396270830277183746791311046545995875950320722179215954468559544753224621	15	98
4907602820766044260332130797687377232671215028937673307949782967000949331612865034664813658692569167292918641382867152131441244102598379901161243030643258732747937676027870227219483808467807692197133989299182133743742608450556174533241861608182533485168504700011161742369128974242093881001422435301090352641618640951167378967753904389786060527863816292978465773577201194632981397854604011675116620401052137007011949228226063041823310753396270830277183746791311046545995875950320722179215954468559544753224621	16	98
4907602820766044260332130797687377232671215028937673307949782967000949331612865034664813658692569167292918641382867152131441244102598379901161243030643258732747937676027870227219483808467807692197133989299182133743742608450556174533241861608182533485168504700011161742369128974242093881001422435301090352641618640951167378967753904389786060527863816292978465773577201194632981397854604011675116620401052137007011949228226063041823310753396270830277183746791311046545995875950320722179215954468559544753224621	17	98

Продовження таблиці Б.1 – Рух однієї з частинок популяції за 20 ітерацій

Номер блоку	Iter	NL
4907602820766044260332130797687377232671215028937673307949782967000949331612865034664813658692569167292918641382867152131441244102598379901161243030643258732747937676027870227219483808467807692197133989299182133743742608450556174533241861608182533485168504700011161742369128974242093881001422435301090352641618640951167378967753904389786060527863816292978465773577201194632981397854604011675116620401052137007011949228226063041823310753396270830277187467913110465459958759503207221792159544685595447532246 21	18	98
4907602820766044260332130797687377232671215028937673307949782967000949331612865034664813658692569167292918641382867152131441244102598379901161243030643258732747937676027870227219483808467807692197133989299182133743742608450556174533241861608182533485168504700011161742369128974242093881001422435301090352641618640951167378967753904389786060527863816292978465773577201194632981397854604011675116620401052137007011949228226063041823310753396270830277187467913110465459958759503207221792159544685595447532246 21	19	98
4907602820766044260332130797687377232671215028937673307949782967000949331612865034664813658692569167292918641382867152131441244102598379901161243030643258732747937676027870227219483808467807692197133989299182133743742608450556174533241861608182533485168504700011161742369128974242093881001422435301090352641618640951167378967753904389786060527863816292978465773577201194632981397854604011675116620401052137007011949228226063041823310753396270830277187467913110465459958759503207221792159544685595447532246 21	20	98

ДОДАТОК В

Результати експериментальних досліджень простору блоків

Таблиця В.1 – Значення cost-функцій для блоків поруч з AES S-box (від'ємний діапазон)

№ відносно AES	NL	Зн. WCF	Зн. WHS	Зн. PCF
S-блоку AES	112	0	423026995376	7491,62
-1	110	11700598210560	430086705510	3848,05
-2	108	44804729733120	430945926406	2001,12
-3	108	52510001725440	428708549140	2004,23
-4	108	30440580710400	443610209288	1979,06
-5	108	35006667816960	435706220520	1979,26
-6	108	46326758768640	429396202398	2003,66
-7	108	43187573882880	423729590896	2000,74
-8	108	47087773286400	443219012938	2000,85
-9	108	47087773286400	441411094412	2001,23
-10	110	17598460723200	439695085512	3902,97
-11	108	28538044416000	444493642016	1972,34
-12	108	29489312563200	432263985516	1971,26
-13	108	40428896256000	438870460546	1996,81
-14	108	42521686179840	450216684318	2002,73

Продовження таблиці В.1 – Значення cost-функцій для блоків поруч з AES S-box (від’ємний діапазон)

№ відносно AES	NL	Зн. WCF	Зн. WHS	Зн. PCF
-15	108	46802392842240	432527810036	2003,9
-16	108	27396522639360	434028474010	1970,32
-17	108	45755997880320	428612686260	1999,98
-18	108	33960272855040	427530724568	1971,05
-19	108	48039041433600	439803811100	1999,53
-20	108	27777029898240	434194758902	1974,82
-21	108	32057736560640	418313803146	1975,61
-22	110	16361812131840	424359996748	3900,94
-23	108	27016015380480	421181586264	1971,95
-24	110	11034710507520	425293347302	3852,16
-25	110	17408207093760	437995892574	3904,26
-26	108	43568081141760	438329754918	1997,31
-27	108	32818751078400	427649368258	1970,42
-28	108	49465943654400	442103563230	2005,69
-29	108	53556396687360	434199574462	2005,33
-30	108	29108805304320	428337021516	1970,45

Таблиця В.2 – Значення cost-функцій для блоків поруч з AES S-box (додатній діапазон)

№ відносно AES	NL	Зн. WCF	Зн. WHS	Зн. PCF
S-блоку AES	112	0	423026995376	7491,62
+1	110	17598460723200	429899107746	3909,85
+2	110	11985978654720	433077518230	3857,61
+3	110	11320090951680	416047451220	3852,37
+4	110	17408207093760	426285571838	3902,36
+5	108	42807066624000	433994526992	1969,83
+6	110	11034710507520	425264372642	3841,33
+7	108	36338443223040	433379771026	1976,81
+8	110	18359475240960	438795558776	3904,74
+9	110	17027699834880	430261458110	3904,83
+10	108	30821087969280	444407400210	1971,69
+11	108	30916214784000	444045049846	1972,28
+12	110	17122826649600	443757904890	3905,77
+13	110	18549728870400	443617891644	3907,39
+14	110	12461612728320	447238568170	3857,38
+15	108	37479964999680	440052113552	1978,98
+16	108	45185236992000	443959935034	1978,69

Продовження таблиці В.2 – Значення cost-функцій для блоків поруч з AES S-box (додатній діапазон)

№ відносно AES	NL	Зн. WCF	Зн. WHS	Зн. PCF
+17	110	17788714352640	421463238970	3900,44
+18	108	35767682334720	429906248364	1974,61
+19	110	12461612728320	433439868594	3857,38
+20	110	18074094796800	444120255254	3909,61
+21	108	50702592245760	432992403418	1983,7
+22	108	38336106332160	435229780684	1980,03
+23	110	16361812131840	428789810030	3902,3
+24	110	11034710507520	434432645168	3852,16
+25	108	28347790786560	428602212266	1977,02
+26	108	35482301890560	444483168022	1979
+27	110	17027699834880	417618338858	3899,93
+28	108	29489312563200	427856459476	1971,7
+29	108	49085436395520	432697631512	1996,44
+30	108	31391848857600	436240563694	1970,98