Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Інститут інформаційної безпеки, радіоелектроніки та телекомунікацій
Кафедра кібербезпеки та програмного забезпечення

Хименко Михайло Валерійович,
група РФ-171

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Розробка генератора ключових послідовностей на основі клітинних автоматів і бент-функцій багатозначної логіки

Спеціальність:
122 Комп'ютерні науки
Спеціалізація, освітня програма
Комп'ютерні науки та інформаційна безпека

Керівник:
Соколов Артем Вікторович,
к.т.н., доцент

Одеса – 2022

Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Інститут інформаційної безпеки, радіоелектроніки та телекомунікацій
Кафедра кібербезпеки та програмного забезпечення

Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп’ютерні науки
Спеціалізація, освітня програма
Комп'ютерні науки та інформаційна безпека

ЗАТВЕРДЖУЮ
Завідувач кафедри КБПЗ

____

д.т.н.,проф. А.А.Кобозєва
____ ___________2022р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Хименко Михайлу Валерійовичу

1.Тема роботи: *Розробка генератора ключових послідовностей на основі клітинних автоматів і бент-функцій багатозначної логіки*, керівник роботи *Соколов Артем Вікторович, к.т.н., доцент*, затверджені наказом ректора від „____”__________ 20___ р. №_______ .

2.Зміст роботи: *аналіз предметної області, методи вирішення задач та аналіз технологій оцінювання результатів, реалізація програмного продукту.*

3.Перелік ілюстративного матеріалу: б*лок-схема генератора випадкових послідовностей, резгістр зсуву зі зворотним зв'язком, РЗЛЗЗ Фібоначчі на 16 біт, блок-схема алгоритму з використанням дуальних бент-функцій, гістограма нормального розподілу, графік автокореляції, демонстрація часу, затраченого на створення послідовності в 100Кб, графік залежності часу від розміру вхідних даних у Matlab R2013a, результат проходження NIST-тестів алгоритмом першої еволюції, демонстрація часу, затраченого на створення послідовності в 100Кб алгоритмом другої еволюції, демонстрація часу, затраченого на створення послідовності в 1Мб алгоритмом другої еволюції, результат проходження NIST-тестів алгоритмом другої еволюції, результат проходження NIST-тестів*

алгоритмом третьої еволюції, приклад сітки алгоритма NESW, блок-схема алгоритму фінальної еволюції, демонстрація часу, затраченого на створення послідовності в 100Кб агоритмом фінальної еволюції, результат проходження NIST-тестів алгоритмом фінальної еволюції, графічний інтерфейс додатку дл аналізу властивостей бент-функцій.

4. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Дата видачі завдання “______”_____________2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	*Аналіз джерел з теми випускної кваліфікаційної роботи*	*05-10-2022*	*виконано*
2	*Обґрунтування вибору рішення. Збір даних*	*15-10-2022*	*виконано*
3	*Аналіз технологій необхідних для реалізації продукту*	*21-10-2022*	*виконано*
4	*Аналіз технологій оцінювання результатів*	*22-10-2022*	*виконано*
5	*Удосконалення алгоритму*	*11-11-2022*	*виконано*
6	*Розробка програмного додатка*	*18-11-2022*	*виконано*
7	*Підготовка тексту роботи*	*21-11-2022*	*виконано*
8	*Підготовка презентації та доповіді*	*22-11-2022*	*виконано*
9	*Попередній захист*	*02-12-2022*	*виконано*
10	*Нормоконтроль, рецензування*	*19-12-2022*	*виконано*
11	*Занесення роботи в електронний архів*	*20-12-2022*	*виконано*
12	*Допуск до захисту у завідувача кафедри*	*21-12-2022*	*виконано*

Здобувач вищої освіти _______________ *Хименко М.В.*

Керівник роботи _______________ *Соколов А.В.*

АНОТАЦІЯ

Кваліфікаційна робота на тему «Розробка генератора ключових послідовностей на основі клітинних автоматів і бент-функцій багатозначної логіки» на здобуття другого (магістерського) рівня вищої освіти за спеціальністю 122 – Комп’ютерні науки, освітня програма: Комп'ютерні науки та інформаційна безпека, містить 21 рисунок, 1 таблицю, 2 додатки, 20 літературних джерел за переліком посилань. Робота виконана на 57 сторінках загального тексту і 45 сторінках основного тексту.

Метою роботи є удосконалення алгоритму формування ключових послідовностей на основі бент-функцій та регістрів зсуву з лінійним зворотнім зв’язком.

В результаті виконання роботи був розроблений графічний додаток мовою Python, що використовує удосконалений алгоритм формування випадкових послідовностей на основі четвіркових бент-функцій, схем розміщення NESW та порівняльного тестування результатів за допомогою використання NIST-тестів для оцінки сформованих послідовностях.

Результати даної роботи можна використовувати для дослідження залежності пар 4-бент-функцій на вихідну псевдовипадкову послідовність та для подальшого її тестування серійними статистичними NIST-тестами.

ГЕНЕРАТОР ПЕРЕШКОД, ПСЕВДОВИПАДКОВІ ПОСЛІДОВНОСТІ, ГЕНЕРАЦІЯ КЛЮЧОВИХ ПОСЛІДОВНОСТЕЙ, БЕНТ-ФУНКЦІЇ, РЕГІСТР ЗСУВУ.

ANNOTATION

Qualification work on the topic "Development of a key sequence generator based on cellular automata and bent functions of multi-valued logic" for obtaining second (master's) level of higher education in the specialty 122 Computer science, educational program: Computer science and information security, contains 21 pictures, 1 table, 2 appendices, 20 literary sources according to the list of references. The work is completed on 57 pages of the general text and 45 pages of the main text.

The aim of the work is to improve the algorithm for forming key sequences based on bend functions and shift registers with linear feedback.

As a result of the work, a graphical application was developed in Python, which uses an improved algorithm for generating random sequences based on quadruple bend functions, NESW placement schemes and comparative testing of results using NIST tests to evaluate the generated sequences.

The results of this work can be used to study the dependence of pairs of 4-bent functions on the original pseudo-random sequence and to further test it with serial NIST statistical tests.

NOISE GENERATOR, PSEUDORANDOM SEQUENCES, GENERATION OF KEY SEQUENCES, BENT FUNCTIONS, SHIFT REGISTER.

ЗМІСТ

ВСТУП.. 8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 11
1.1 Класичний генератор послідовностей.. 11
1.2 Лінійний зсувний регістр зі зворотним зв'язком 14
1.3 Лінійний конгруентний генератор 15
1.4 Нелінійний конгруентний генератор 18
2 МЕТОДИ ВИРІШЕННЯ ЗАДАЧ ТА АНАЛІЗ ТЕХНОЛОГІЙ ОЦІНЮВАННЯ РЕЗУЛЬТАТІВ.. 20
2.1 Вибір мови для написання алгоритму та його складових............................. 21
2.1.1 Написання алгоритму 21
2.1.2 Написання графічного інтерфейсу 23
2.2 Тестування псевдовипадкових послідовностей 24
2.2.1 Графічні тести... 24
2.2.2 Статистичні тести.. 26
3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ 34
3.1 Аналіз недоліків алгоритму 34
3.1.1 Удосконалення швидкодії.. 36
3.1.2 Удосконалення результатів тестування 39
3.2 Створення графічного інтерфейсу.. 45
ВИСНОВОК.. 47
ПЕРЕЛІК ПОСИЛАНЬ 49
Додаток А. Лістинг програми мовою Python ... **Ошибка! Закладка не определена.**
Додаток Б. Лістинг програмою мовою Matlab . **Ошибка! Закладка не определена.**

ВСТУП

Зі зростанням кількості користувачів мережі інтернет та з розростанням впливу даної мережі на життя людей стає все більше індивідів, які зацікавлені у отриманні приватної інформації незаконним шляхом. Для захисту такого роду інформації користуються безліччю методів, деякі з яких існують уже не перший десяток років, а інші – створюються та покращуються щоденно.

Під час дослідження знань та навичок учнів шкіл про захист інформації було припущено, що можливість доступу до мережі інтернет за допомогою мобільних телефонів та інших засобів зв'язку іноді призводить до втрати приватних даних через невміння відчувати загрозу зі сторони лиходіїв. Звісно такі випадки поодинокі, але це завдяки тому, що програми, які використовуються людьми мають власні засоби захисту інформації такі, як шифрування. Завдяки ньому навіть за умови витоку даних, отримання корисних знань унеможливлюється відсутністю можливості розшифрувати інформацію.

Перед написанням даної роботи було проведено дослідження уже існуючих алгоритмів створення ключових послідовностей для шифрування даних та розроблено удосконалення генератора, який був представлений на захисті бакалаврської роботи[1]. Тим не менш, існуючі алгоритми або характеризуються низькою криптостійкістю, або значною обчислювальною складністю та є слабо адаптованим для програмної реалізації. Означенне гальмує їх застосування на практиці та ставить задачу розробки більш досконалих схем криптографічно захищенних генераторів псевдовипадкових ключових послідовностей. Як показують дослідження, ця задача може бути вирішена за рахунок застосування клітинних автоматів та четвіркових бент-послідовностей. Розроблений в результаті генератор характеризується повною відповідністю статистичним NIST-тестам [2]. Також було вирішено змінити алгоритм запису послідовності у файл, що значно вплинуло на швидкість формування вихідної послідовності. Цей фактор значно зменшив навантаження на обчислювальну машину, що стало поштовхом для випробування алгоритму створенням послідовностей набагато

більшого розміру, які, в свою чергу, дають більш точні результати при дослідженні стохастичних властивостей. Далі у роботі будуть представлені результати порівняння алгоритмів, їх швидкодії та стохастичних властивостей створених ними вихідних послідовностей.

Метою даної роботи є удосконалення алгоритму формування ключових послідовностей на основі бент-функцій та регістрів зсуву з лінійним зворотнім зв'язком.

Після аналізу ринку та технологій, що найшвидше розвиваються, станом на 01.09.2022 було прийнято рішення вибрати за об'єкт дослідження формування псевдовипадкових послідовностей.

Після постановки задачі можна виокремити предмет дослідження яким вважатимемо алгоритми використання бент-функцій багатозначної логіки для генерації ключових послідовностей.

В результаті роботи удосконалено алгоритм формування ключових послідовностей, який відрізняється від існуючих використанням четвіркових бент-функцій, що дозволило скоротити обчислювальні витрати та отримати на виході більш випадкову послідовність.

Практичне значення отриманих результатів полягає в тому, що удосконалений алгоритм генерації псевдовипадкових послідовностей на основі четвіркових бент-функцій та додаток з графічним інтерфейсом, що був створений у результаті написання роботи, можна в подальшому використовувати для дослідження бент-функцій багатозначної логіки та пошуку залежності використаних функцій на результати тестування вихідної послідовності на випадковість за допомогою NIST-тестів.

Завдання, які потрібно розв'язати для створення програмного продукту наступні:

- аналіз предметної області та існуючих програмних рішень для виконання подібних задач;
- перевід існуючого алгоритму на бент-функції багатозначної логіки для усунення можливості проведення атаки Берлекемпа-Мессі[3]

- оцінка швидкодії алгоритму та стохастичних властивостей вихідної послідовності
- удосконалення алгоритму
- розробка та тестування програмного застосунку для генерації ключових послідовностей.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Шифрування є найпоширенішим способом захисту інформації. Криптографічні алгоритми зазвичай використовують випадкові числа, щоб приховати корисну інформацію у випадково згенерованих послідовностях. Як випливає з назви, такі послідовності виробляються генераторами.

Якщо згенерована послідовність чисел виглядає випадковою, генератор називається генератором псевдовипадкових чисел. Іншими словами, статистичний тест на випадковість не може виявити, що числова послідовність була створена навмисно.

1.1 Класичний генератор послідовностей

Класичним представником генераторів випадкової послідовності (далі ГПВ) є генератори випадкової послідовності, засновані на фізичних явищах. Фізичний ГПВ – це пристрій, який генерує випадкові послідовності на основі таких фізичних процесів:

- Фотоелектричний ефект.
- Тепловий шум;
- Квантові явища, тощо.

Показання таких процесів є абсолютно непередбачуваними величинами. Апаратне забезпечення, яке записує фізичні явища, називається фізичним джерелом шуму. У більшості випадків значення отримують шляхом відцифрування (дискретизації) аналогового сигналу від джерела шуму. Дискретизовані значення з фізичного джерела під час обробки перетворюються у випадкову серію значень. Ця обробка відбувається для зменшення похибки розподілу ймовірності появи відповідних значень дискретизованої величини. Також існує можливість подання дискретизованиого значення фізичного джерела шуму безпосередньо у вихідний блок не виконуючи додаткові операції. Тобто, в

такому випадку, результативна випадкова послідовність повністю відповідає дискретизованій послідовності значень від джерела шуму.

Для отримання кінцевої випадкової послідовності наборів після кожної операції дискретизації(або декількох операцій при збереженні значень в пам`яті) використовуються сигнал синхронізації(безперервний або аперіодичний). Ентропія випадкової послідовності, що видається джерелом шуму, підвищується з генеруванням кожного символу випадкової послідовності.

Генератори випадкової послідовності можуть бути побудовані не тільки на основі фізичних явищ, але і в поєднанні з додатковими програмними продуктами та алгоритмами. На рисунку 1.1 показана функціональна блок-схема ГВП, визначена в міжнародному стандарті ISO/IEC 18031:2011[6].

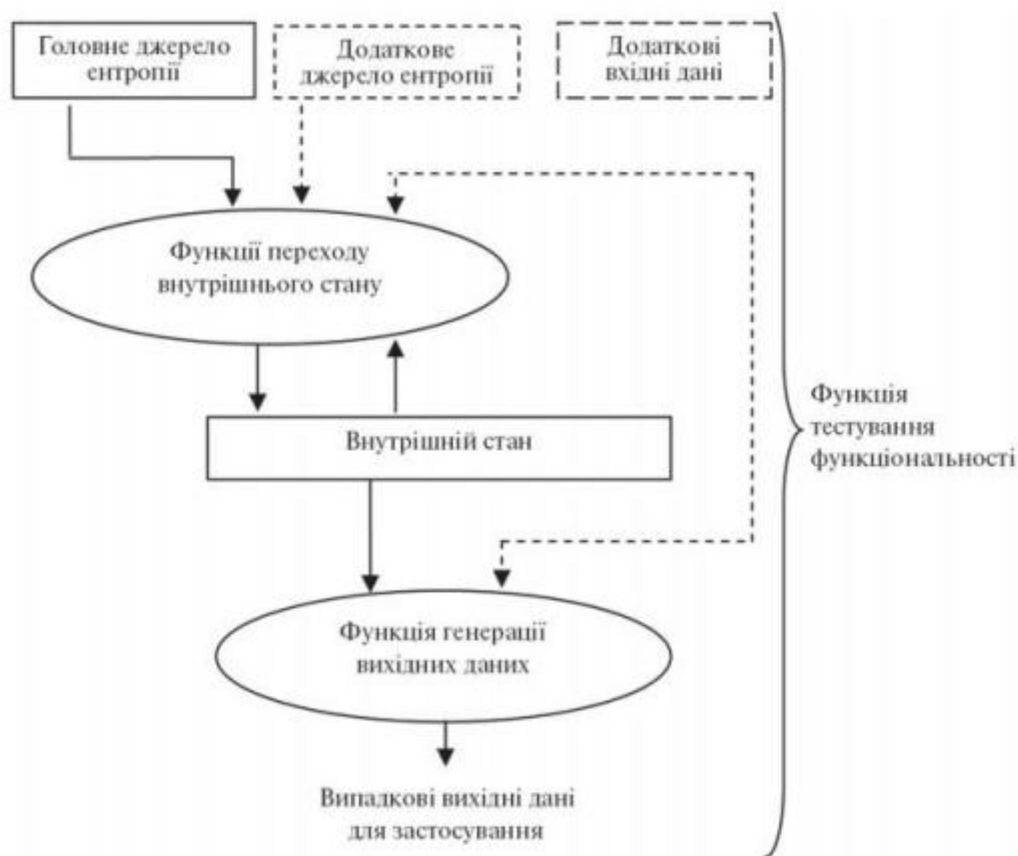

Рисунок 1.1 - Блок-схема генератора випадкових послідовностей

Залежно від багатьох факторів деякі складові частини можуть бути присутніми або відсутніми в складі ГВП. Такі компоненти позначені на блок-схемі пунктирними лініями.

Вимоги встановлюються для кожного компонента стандарту. Приклади фізичних явищ включають:

- Видалення даних про температуру з діодів пристрою.
- фіксовані характеристики радіоактивного розпаду
- нестабільність частоти коливань у режимі вільних коливань.
- конденсатор, який повністю заряджається за заданий проміжок часу.

Для реалізації генератора на основі перших двох явищ потрібен зовнішній пристрій. Однак зловмисники можуть швидко контролювати такі пристрої та зловмисно маніпулювати ними. З цієї причини має сенс використовувати осцилятори та конденсатори як фізичні джерела шуму, оскільки стійкий стан можна реалізувати за допомогою дуже великих інтегральних схем. Такі схеми захищені від несанкціонованого втручання.

При розгляді ГВП в програмних продуктах на користувацьких девайсах як джерела шуму використовуються:

- Час, що минув між натисканнями клавіш або рухами миші.
- системний годинник комп'ютера;
- вміст буфера введення/виведення;
- завантаження системи;
- статистичні дані мережі;
- інші параметри операційної системи.

Більшість фізичних ГВП мають низку серйозних недоліків і послідовностей, які заважають їх використанню. Найважливіші з них:

- неприпустимо короткі або неповний період повторення послідовності
- Погана ідентифікація. Це дозволяє легко передбачити як продовження послідовності, так і її попередні значення;
- Такі характеристики, як випадковість, рівноймовірність, незалежність і однорідність, які не відповідають вимогам.

1.2 Лінійний зсувний регістр зі зворотним зв'язком

Регістр зсуву - це серія тригерів, з'єднаних послідовно. Приклад одного з таких генераторів показано на рисунку 1.2. Основним режимом роботи таких генераторів є логічне зсув значень, записаних в кожному регістрі тригера. Тобто після сигналу синхронізації значення кожного попереднього регістрового тригера записується до наступного регістрового тригера в ланцюжку тригерів по порядку. Набір значень, що зберігаються в регістрах, зміщується на одну позицію вліво або на одну позицію вправо кожного циклу.

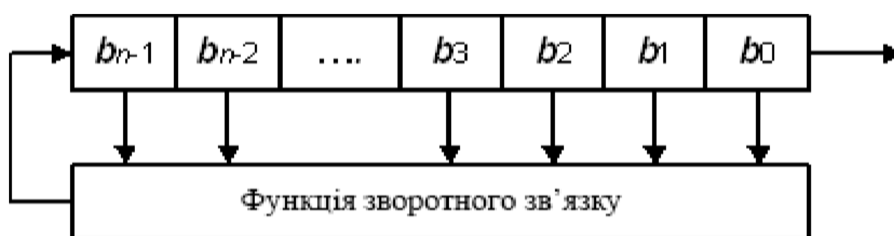

Рисунок 1.2 - Регістр зсуву зі зворотним зв'язком

Один з видів регістру зсуву з лінійним зворотним зв’язком (далі РЗЛЗЗ) представлено на рисунку 1.3. Зазвичай РЗЛЗЗ складаються з наступних компонентів:

- сам регістр зсуву.
- Функція зворотного зв'язку.

Головна відмінність між звичайними регістрами зсуву і РЗЛЗЗ в тому, що при зміщенні бітів деякі позиції змінюють не тільки своє положення, але і значення відповідно до функції, яка прописана в алгоритмі. Зазвичай цією функцією є виключна диз'юнкція (XOR)[4] по відношенню до іншого біта послідовності або іншої функції.

Довжина регістра зсуву - це кількість бітів (тригерів). Коли біт потрібно «витягнути», усі біти в регістрі зсуву зсуваються праворуч або ліворуч на одну позицію. Нове значення крайнього біта визначається функцією, аргументом якої є

інший біт регістра. У більшості випадків ефективний біт вихідного регістра є молодшим бітом. Період регістра зсуву - це довжина отриманої послідовності до того, як вона почне повторюватися.

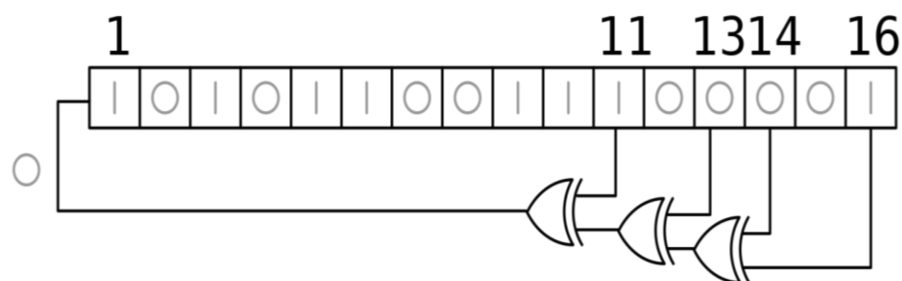

Рисунок 1.3 - РЗЛЗЗ Фібоначчі на 16 біт

ЗЛЗЗ може перебувати у таких внутрішніх станах, де n є довжиною регістра зсуву. Така умова є правильною, оскільки якщо всі біти в регістрі зсуву мають значення 0, то лише 0 бітів є дійсними у виході в результаті такої умови (оскільки xor використовується як варіант зворотного зв’язку). Теоретично довжина послідовності псевдовипадкових чисел відповідає внутрішньому стану вихідного регістра, тому РЗЛЗЗ може генерувати послідовність довжиною біта 2n-1.

Регістр зсуву з лінійним зворотнім зв’язком генерує всі можливі комбінації вихідних речень лише для певних послідовностей.

Можна виділити наступні переваги генератора на основі регістрів зсуву:

- Ефективна апаратна реалізація.
- швидкість дії;

Натомість недоліком регістрових генераторів є те, що вони генерують лише циклічні послідовності чисел. Щоб отримати ациклічну послідовність, потрібно підключити до виходу генератора перетворювач комбінаційного коду. Але при цьому значно погіршується швидкодія.

1.3 Лінійний конгруентний генератор

Лінійний конгруентний генератор (далі ЛКГ) — це алгоритм, який генерує послідовність псевдовипадкових чисел, отриманих за допомогою розривних кусково-лінійних рівнянь[5]. Ця процедура є одним із найстаріших і

найвідоміших алгоритмів генерації псевдовипадкових чисел. Теорію, що лежить в їх основі, відносно легко зрозуміти та швидко реалізувати, особливо на комп'ютерному обладнанні, яке може забезпечувати модульні операції за рахунок зменшення бітів пам'яті.

Просту схему генератора можна представити за допомогою наступного ітераційного рівняння

$$x_{n+1} = (ax_n + c) \, mod \, m, \qquad (1.1)$$

де виконуються наступні умови:

- x – послідовність псевдовипадкових значень
- $m>0$, де m – модуль
- $m>a>0$, де a – множник
- $m>c>0$, де c – інкремент

Період такого генератора завжди менший за m. Якщо параметри правильно підібрані, то такий генератор є генератором максимальної довжини і лише у цьому випадку має період рівний m. Для того, щоб це сталося потрібне виконання наступних умов:

- Числа c та m взаємно прості
- a-1 кратно p для кожного простого p, що є дільником m
- Якщо m кратно 4, то a-1 також кратно 4

Якщо с дорівнює нулю, то генератор приймає наступний вигляд

$$x_{n+1} = ax_n (mod \, m) \qquad (1.2)$$

Якщо правильно підібрати константи, то a може приймати значення 75 і 16807, а m - $2^{31} - 1$, отримаємо генератор з максимальним періодом повторення. Ці константи були запропоновані Парком та Міллером, тому генератор з цими константами називається генератором Парка-Міллера.[6]

Основною перевагою лінійних конгруентних генераторів є їх швидкість, за рахунок малої кількості операцій на байт і простота реалізації.

Для того, щоб статистичні властивості наведеного вище генератора відповідали вимогам потрібен критерій для визначення коефіцієнтів. Одним із

критеріїв є значення змінної c. Її значення має бути достатньо великим, адже період послідовності не перевищує с елементів.

Проблема заключається в тому, що для отримання результату операції визначення залишку використовується найповільніша арифметична операція — ділення. Тому на комп'ютері з двійковою системою числення має сенс вибрати $c = 2n$. У цьому випадку команда, яку комп'ютер запитує про залишок від ділення, зводиться до логічної операції AND.

Один із способів визначення значення c є використання простого числа. Такі прості числа повинні бути меншими за $2n$. У цьому випадку в технічній літературі доведено, що молодші розряди результуючого x_{n+1}-го випадкового числа однаковий випадковий характер, що й старші розряди. Це позначається позитивно на всю послідовність псевдовипадкових чисел.

Як приклад, ми можемо взяти одне з чисел Мерсенна[7], що дорівнює 511 (512 - 1), тому $c = 511$.

Збільшення довжини періоду псевдовипадкових послідовностей є однією з основних вимог до лінійних конгруентних послідовностей. Довжина періоду такої послідовності залежить від значень c і n.

У таблиці 1.1 наведені переліки констант, що забезпечують роботу генератора з найбільшим періодом[8]. Таблиця складена так, що значення упорядковані за максимальним добутком, який не викликає переповнення в слові даної довжини.

Таблиця 1.1 - Добрі константи для лінійних конгруентних генераторів

Переповнюється при	a	c	m
2^{20}	106	1283	6075
2^{21}	211	1663	7875
2^{22}	421	1663	7875
2^{23}	430	2531	11979
2^{23}	936	1399	6655
2^{23}	1366	1283	6075
2^{24}	171	11213	53125
2^{24}	859	2531	11979
2^{24}	419	6173	29282
2^{24}	967	3041	14406

Таблиця 1.1 наведена в скороченому вигляді, першоджерело таблиці можна отримати з книги «Прикладна криптографія»[9].

Швидкість є основною перевагою лінійних конгруентних генераторів. Це тому, що псевдовипадкові послідовності мають менше операцій на біт. Недоліком таких генераторів є передбачуваність послідовностей. На даний момент лінійні конгруентні генератори не використовуються для завдань, пов'язаних з криптографією. Натомість їх використовують для завдань моделювання. Прикладом такого завдання є моделювання випадкової поведінки. Після генерації така псевдовипадкова числова послідовність при тестуванні значень статистичних властивостей показує одну з найменших похибок серед інших генераторів. Ці результати спостерігаються при виконанні більшості емпіричних тестів.

1.4 Нелінійний конгруентний генератор

Найбільше розповсюдження здобули квадратичні та кубічні конгруентні генератори. Квадратичний генератор працює за наступною формулою:

$$x_{i+1} = \left(ax_i^2 + bx_i + c\right) \bmod m \qquad (1.3)$$

Де x_{i+1} – наступне число, x_i - попереднє число, a , b, c , m – константи.

Квадратична конгруентна послідовність має найбільший період якщо задовольняються всі умови:

- c та m - взаємно прості числа
- a та b кратні p (будь-який простий дільник m)
- a – парне число

Кубічний же генератор має наступний вигляд:

$$X_n = (aX_{n-1}^3 + bX_{n-1}^2 + cX_{n-1} + d)\, mod\, m \qquad (1.4)$$

Лінійні конгруентні методи генерують статистично хороші псевдовипадкові послідовності, але вони не є криптографічно надійними. Генератори на основі лінійної конгруентності є передбачуваними і тому не можуть використовуватися для криптографії. Генератори, засновані на лінійній конгруентності, були вперше

зламані Джимом Рідом, а потім Джоан Бояр. Вона також успішно виявила недоліки квадратних і кубічних генераторів. Інші дослідники розширили ідеї Бояр, розробивши методи виявлення помилок у генераторах довільних поліномів. Це довело, що генератори на основі конгруентності непридатні для використання в криптографії. Однак генератори на основі лінійної конгруентності зберігають свою корисність для некриптографічних програм, таких як моделювання. Вони ефективні та демонструють хороші статистичні властивості в найбільш часто використовуваних емпіричних тестах.

2 МЕТОДИ ВИРІШЕННЯ ЗАДАЧ ТА АНАЛІЗ ТЕХНОЛОГІЙ ОЦІНЮВАННЯ РЕЗУЛЬТАТІВ

Основною задачею роботи було прийнято удосконалення існуючого алгоритму на основі дуальних бент-функцій, представленого на рисунку 2.1. Даний алгоритм було розроблено під час написання статті[10] для використання його у пристрої генерації перешкод методом створення шуму на певній радіочастоті.

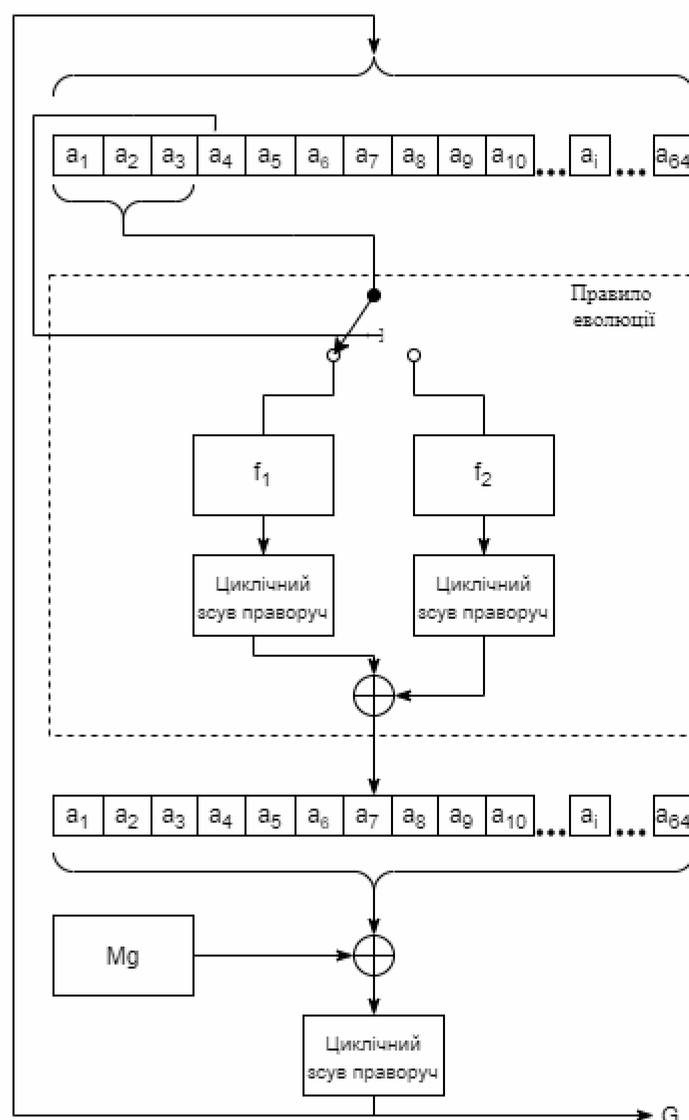

Рисунок 2.1 – Блок-схема алгоритму з використанням дуальних бент-функцій

Потреба в удосконаленні була заснована на кількості бент-функцій, що приймають участь у алгоритмі генерації ключової послідовності, швидкодії

роботи алгоритму та якості створеної послідовності. У статті[11] описана потреба та важливість створення бент-функцій з потужністю алфавіту 4. Попередній алгоритм використовує дуальні бент-функції, що означає потужність алфавіту 2.

2.1 Вибір мови для написання алгоритму та його складових

2.1.1 Написання алгоритму

Через те, що алгоритм розробляється не з нуля, то було вирішено удосконалювати його в уже існуючому середовищі, яким виявився Matlab.

MatLab — одна з найстаріших, добре розроблених і перевірених часом автоматизованих математичних обчислювальних систем, побудованих на високорівневому представленні та застосуванні матричних операцій. Звідси і назва системи - MATrix LAboratory, що означає Матрична лабораторія.

Matlab надається як комплекс Matlab + Simulink + Toolbox + Blockset, де частини Toolbox і Blockset пакетів розширення викликів MathWorks Corporation для систем Matlab і Simulink відповідно.

Однією з головних місій системи Matlab завжди було надання користувачам потужної мови програмування, орієнтованої на технічні та математичні обчислення та здатної перевершити можливості традиційних мов програмування, які використовувалися для реалізації чисельних методів протягом багатьох років. При цьому особлива увага приділялася збільшенню швидкості обчислень і адаптації системи до вирішення найрізноманітніших завдань користувачів.

Для MATLAB можна створювати спеціальні набори інструментів (панелі інструментів англійською), які розширюють його можливості. Набори інструментів — це набори функцій, написаних у MATLAB, які вирішують певні класи задач. Mathworks пропонує набори інструментів для багатьох областей, зокрема:

- Цифрова обробка сигналів, зображень і даних: DSP Toolbox, Image Processing Toolbox, Wavelet Toolbox, Communication Toolbox, Filter Design Toolbox - Набір

функцій, який вирішує широкий спектр завдань з обробки сигналів, зображень, проектування цифрових фільтрів і систем зв'язку. .

- Системи керування: Control System Toolbox, μ-Analysis and Synthesis Toolbox, Robust Control Toolbox, System Identification Toolbox, LMI Control Toolbox, Model Predictive Control Toolbox, Model-Based Calibration Toolbox - допомагає з динамічним функціональним набором для системного аналізу та синтезу, проектування, моделювання та ідентифікація систем керування, у тому числі сучасних алгоритмів, таких як робастної управління, Н ∞-управління, ЛМН-синтез, μ-синтез та інші.
- Фінансовий аналіз: GARCH Toolbox, Fixed Income Toolbox, Financial Time Series Toolbox, Financial Derivatives Toolbox, Financial Toolbox, Datafeed Toolbox - набір, який дозволяє швидко та ефективно збирати, обробляти та передавати різні функції фінансової інформації.
- Включає аналіз і синтез тривимірних географічних карт: Mapping Toolbox.
- Збір і аналіз експериментальних даних: Data Acquisition Toolbox, Image Acquisition Toolbox, Instrument Control Toolbox, Code Composer Studio Link - Набір функцій, які дозволяють зберігати та обробляти дані, отримані під час експериментів, включаючи живі дані. Підтримує широкий спектр наукових та інженерних вимірювальних пристроїв.
- Візуалізація та представлення даних: Virtual Reality Toolbox - Створюйте інтерактивні світи та візуалізуйте наукову інформацію за допомогою технології віртуальної реальності та мови VRML.
- Інструменти розробки: MATLAB Builder для COM, MATLAB Builder для Excel, MATLAB Builder для NET, MATLAB Compiler– набори функцій, що дозволяють створювати незалежні програми з середовища MATLAB.
- Взаємодія із зовнішніми програмними продуктами: MATLAB Report Generator, Excel Link, Database Toolbox, MATLAB Web Server, ModelSim Link - Набір функцій, який дозволяє зберігати дані різними способами, щоб інші програми могли їх використовувати.
- База даних: Database Toolbox — Інструменти для роботи з базами даних.

- Наукові та математичні пакети: Bioinformatics Toolbox, Curve Fitting Toolbox, Fixed Point Toolbox, Fuzzy Logic Toolbox, Genetic Algorithms and Direct Search Toolbox, OPC Toolbox, Optimization Toolbox, Partial Differential Equations Toolbox, Spline Toolbox, Statistics Toolbox, RF Toolbox - набір інструментів спеціалізовані математичні функції для вирішення широкого кола наукових та інженерних проблем, включаючи розробку генетичних алгоритмів, розв'язання задач у приватних похідних, цілочисельних задач, оптимізації системи тощо.
- Нейронні мережі: Neural Network Toolbox - інструменти для синтезу та аналізу нейронних мереж.
- Symbolic Mathematics: Symbolic Mathematics Toolbox - інструменти символічної математики, які можуть взаємодіяти з символьним процесором програм Maple [12]. Окрім перерахованих вище, існують тисячі наборів інструментів MATLAB, написаних іншими компаніями та любителями.

2.1.2 Написання графічного інтерфейсу

Наступним кроком після вибору мови алгоритму є вибір мови написання графічного інтерфейсу та набору тестів для перевірки ключової послідовності. В пункті роботи 2.2.3 описано процес пошуку і вибору алгоритму для перевірки результуючої послідовності на випадковість.

Так, як програмна реалізація тестів не є метою даної роботи, було прийнято рішення використати уже готовий набір статистичних алгоритмів, що допоможуть оцінити випадковість результатів роботи алгоритму. Серед готових рішень стояв вибір між TEST-U01, реалізацію якого можна знайти мовою C і NIST-тестами, реалізація яких доступна мовою Python.

Головними критеріями вибору були:

- швидкодія;
- можливість взаємодіяти з алгоритмом для автоматизації процесів;
- простота реалізації.

В даному випадку мова С перемагає в швидкості взаємодії з процесором обчислювальної машини.

Відносно можливості взаємодіяти з програмним кодом написаним мовою Matlab обидві мови Python і С тримають паритет. Matlab розроблений таким чином, що його функції можуть однаково легко викликатись обома запропонованими мовами.

Відносно ж простоти реалізації мало хто може посперечатись з тим, що програмний код мовою Python набагато більш простий для сприйняття, набагато більш зрозумілий і легше пишеться. Маючи уже велику кількість досвіду з ним було прийнято рішення вибрати як основну мову написання графічного інтерфейсу – Python.

2.2 Тестування псевдовипадкових послідовностей

2.2.1 Графічні тести

Одним з видів оцінки результатів готових ключових послідовностей створених алгоритмами формування псевдовипадкових величин є графічний метод. Існує велика кількість графічного представлення величин.

З найбільш популярних і часто використовуваних можна виділити наступні графічні методи:

- гістограма це спосіб побудови табличних даних, наближене зображення розподілу числових даних. Фігура, що складається з квадратів без пропусків. Кількісні співвідношення деяких показників відображаються у вигляді прямокутників із пропорційними площами. У більшості випадків для простоти розуміння ширини прямокутників вважаються рівними, а висота визначає пропорції відображуваних параметрів. На рисунку 2.2 показано гістограму так званого «нормального розподілу».

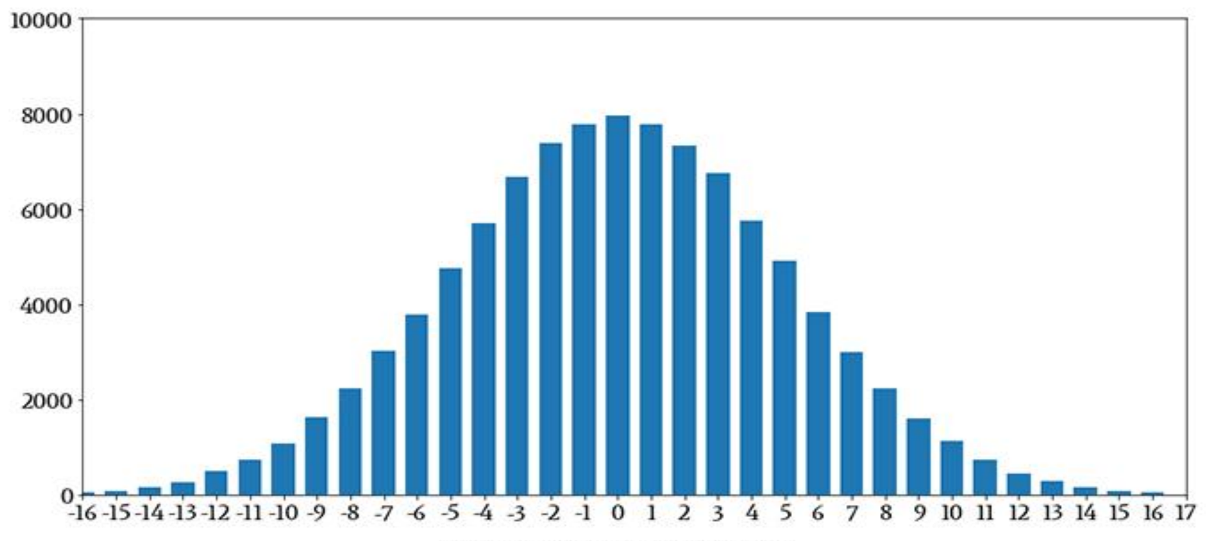

Рисунок 2.2 – Гістограма нормального розподілу.

- автокореляція - це кореляція Пірсона між значеннями цього процесу в різні моменти часу як функція від двох моментів часу, або від відставання в часі. Якщо переносити це визначення на ситуацію оцінювання псевдовипадкової послідовності, то автокореляція - це оцінка між множиною наборів послідовності і окремими множинами цією ж послідовності приклад якої приведений на рисунку 2.3.

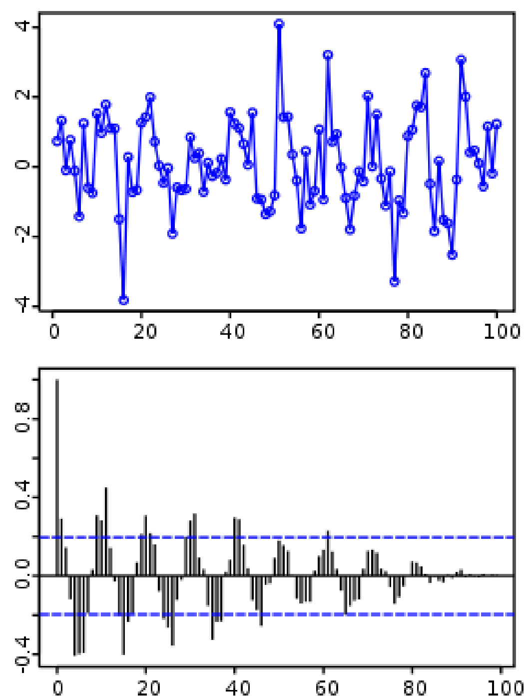

Рисунок 2.3 – Графік автокореляції

- графічний спектральний тест (Метою цього тесту є перевірка однорідності розподілу нулів і одиниць в досліджуваній послідовності на основі аналізу величини випромінювання Фур'є. Нехай маємо двійкову послідовність довжини N. Потім потрібно застосувати дискретне перетворення Фур'є до послідовності, щоб отримати набір гармонік $S = DTF(x)$. Для послідовностей, властивості яких наближаються до властивостей справді випадкової послідовності, кількість гармонік, довжина яких значно перевищує середню гармонічну довжину хвилі, має наближатися до нуля. В іншому випадку порядок не є випадковим
- перевірка на монотонність. Монотонною є послідовність, в якій кожен член x_n є меншим (більшим, не меншим, не більшим) за член послідовності x_{n-1}. Якщо послідовність є монотонною, то це показує на наявність прямих зв'язків між її елементами.

Головним недоліком графічних тестів є те, що фінальний вирок виносить людина, яка проводить візуальну оцінку. Результатом роботи алгоритму буде послідовність символів utf-8[13], що може нараховувати сотні тисяч знаків. Саме через людський фактор не можна вважати графічне оцінювання результатів достатньо коректним.

2.2.2 Статистичні тести

Головною перевагою статистичних тестів над графічними є те, що результатом статистичної оцінки послідовності є числова величина, за якою можна судити про те, яка псевдовипадкова послідовність більш наближена до випадкової.

Серед найпопулярніших серій тестів можна виділити:

а) TEST-U01. Програмна бібліотека, реалізована мовою ANSI C, яка надає набір утиліт для емпіричної перевірки випадковості генераторів випадкових чисел (ГСЧ) [14]. Ця бібліотека була вперше опублікована в 2007 р. П'єром Л'Екуйє та Річардом Сімаром з Університету Монреаля [15]. Ці тести можна застосовувати до зумовлених бібліотечних генераторів, генераторів користувача і потоків випадкових чисел, що зберігаються у файлах. Також доступні спеціальні набори

тестів для однорідних послідовностей випадкових чисел або послідовностей бітів 0-1. Він також пропонує основні інструменти для побудови точкових векторів, згенерованих генераторами.

b) Тести NIST – набір статистичних тестів, розроблених Лабораторією інформаційних технологій, головним дослідницьким підрозділом Національного інституту стандартів і технологій. Він містить 15 статистичних тестів, призначених для визначення того, наскільки випадковою є двійкова послідовність, згенерована апаратним або програмним генератором випадкових чисел. Ці тести базуються на різних статистичних властивостях, унікальних для випадкових послідовностей. У комплекс входять такі тести:

1) Частотний бітовий тест. Суть цього тесту полягає у визначенні частки нулів і одиниць у всій двійковій послідовності. Мета полягає в тому, щоб з’ясувати, чи насправді кількість нулів і одиниць у послідовності приблизно однакова. Це очікувано для двійкових випадкових послідовностей. Цей тест оцінює, наскільки одиничний коефіцієнт близький до 0,5. Тому нулів і одиниць має бути приблизно однакова кількість. Якщо розраховане значення ймовірності під час тестування $p < 0{,}01$, то двійкова послідовність не є випадковою. В іншому випадку порядок буде випадковим. Усі наступні тести запускаються за умовою проходження цього тесту.

2) Тест частотного блоку. Суть тесту полягає у визначенні частки комірок у блоці довжиною m біт. Мета полягає в тому, щоб з’ясувати, чи насправді частота повторення комірок у блоці довжиною m біт становить близько m/2, як і очікувалося б для абсолютно випадкової послідовності. Розраховане під час випробування значення ймовірності p не повинно бути менше 0,01. В іншому випадку ($p < 0{,}01$) двійкова послідовність не є абсолютно випадковою. Якщо припустити, що $m = 1$, тест продовжується тестом №1 (частотний бітовий тест).

3) Серія ідентичних бітових тестів. Суть полягає в тому, щоб підрахувати загальну кількість рядків у вихідній послідовності. Тут слово «рядок» означає безперервну підпослідовність ідентичних бітів. Ланцюжок довжиною k бітів складається з k однакових бітів, починаючи і закінчуючи бітами, що містять

протилежні значення. Мета цього тесту — перевірити, чи дійсно кількість рядків одиниць і нулів різної довжини відповідає числам у випадковому порядку. Зокрема, він визначає, швидко чи повільно чергуються 1 і 0 у вихідній послідовності. Якщо розраховане значення ймовірності під час тестування $p < 0{,}01$, то двійкова послідовність не є випадковою. В іншому випадку замовлення вважається випадковим.

4) Перевірка найдовшої послідовності одиниць у блоці. Цей тест визначає найдовшу послідовність одиниць у блоці довжиною m біт. Мета полягає в тому, щоб перевірити, чи дійсно довжина такої послідовності в абсолютно випадковій послідовності відповідає очікуваній довжині найдовшої одиничної послідовності. Вихідна послідовність вважалася невипадковою, якщо розраховане значення ймовірності під час тестування становило $p < 0{,}01$. В іншому випадку передбачається його випадковість. Зауважимо, що цей тест дає той самий результат при розгляді найдовшої послідовності нулів, припускаючи, що одиниці та 0 зустрічаються приблизно однаково. Тому для вимірювання можна використовувати лише одиниці.

5) Ранговий тест для біноміальних матриць. Тут обчислюються ранги непересічних підматриць, побудованих з вихідних двійкових послідовностей. Метою цього тесту є перевірка лінійної залежності підрядків фіксованої довжини, які складають початкову послідовність. Якщо тест обчислює значення ймовірності $p < 0{,}01$, він робить висновок, що вхідна послідовність бітів не є випадковою. Інакше це вважається абсолютно випадковим. Цей тест також входить до пакету DIEHARD.

6) Тест спектру. Суть тесту полягає в оцінці висоти піку дискретного перетворення Фур'є вихідної послідовності. Мета полягає в виявленні періодичних особливостей у вхідній послідовності, таких як близько розташовані повторювані області. Тому це чітко демонструє відхід від випадковості досліджуваної послідовності. Ідея полягає в тому, що кількість піків вище 95% діапазону відсічення значно перевищує 5%. Якщо значення ймовірності,

розраховане під час тесту, становить $p < 0{,}01$, то двійкова послідовність не є повністю випадковою, інакше вона є випадковою.

7) Тест порівняння зразків, що не перекриваються. Цей тест підраховує кількість визначених шаблонів, знайдених у вихідній послідовності. Мета полягає в тому, щоб ідентифікувати генератори випадкових або псевдовипадкових чисел, які, як правило, слідують заданому аперіодичному шаблону. Подібно до тесту порівняння шаблонів, що перекриваються, вікно довжиною m використовується для пошуку конкретного шаблону довжиною m біт. Якщо шаблон не знайдено, вікно зсувається на 1 біт. Якщо шаблон знайдено, вікно перейде до наступної цифри знайденого шаблону та продовжить пошук. Розраховане під час випробування значення ймовірності p не повинно бути менше 0,01. В іншому випадку ($p < 0{,}01$) двійкова послідовність не є абсолютно випадковою.

8) Тест порівняння повторюваних зразків. Суть цього тесту полягає в підрахунку кількості попередньо визначених шаблонів, знайдених у вихідній послідовності. Подібно до тесту 7 для зіставлення шаблонів без перекриття, вікно довжиною m використовується для пошуку конкретного шаблону довжиною m біт. Аналогічним чином виконується і сам пошук. Якщо шаблон не знайдено, вікно зсувається на 1 біт. Єдина відмінність між цим тестом і тестом 7 полягає в тому, що якщо шаблон знайдено, вікно переходить на одиницю вперед і пошук продовжується. Розраховане під час випробування значення ймовірності p не повинно бути менше 0,01. В іншому випадку ($p < 0{,}01$) двійкова послідовність не є абсолютно випадковою.

9) Загальні статистичні критерії Маурера. Тут визначається кількість бітів між ідентичними шаблонами у вихідній послідовності (міра, безпосередньо пов’язана з довжиною стиснутої послідовності). Мета цього тесту — побачити, чи можна цю послідовність значно стиснути без втрати інформації. Як результат тесту обчислюється значення ймовірності p. Якщо $p < 0{,}01$, вихідна послідовність вважалася невипадковою. В іншому випадку передбачається його випадковість.

10) Лінійний тест складності. Тест базується на принципі роботи регістра зсуву з лінійним зворотним зв’язком (LSF). Мета полягає в тому, щоб з'ясувати,

чи вхідна послідовність є достатньо складною, щоб вважатися абсолютно випадковою. Абсолютно випадкова послідовність характеризується довгим лінійним регістром зсуву зі зворотним зв'язком. Якщо такі регістри занадто короткі, послідовність не вважається абсолютно випадковою. Як результат тесту обчислюється значення ймовірності p. Якщо $p < 0,01$, вихідна послідовність вважалася невипадковою. В іншому випадку передбачається його випадковість.

11) Тест на періодичність. Цей тест підраховує частоту всіх можливих дублікатів m-бітових шаблонів у вихідній послідовності бітів. Мета полягає в тому, щоб визначити, чи насправді 2m повторюваних шаблонів довжиною m бітів зустрічаються приблизно так само часто, як біти в абсолютно випадковій вхідній послідовності. Останній, як відомо, рівномірний. Тобто кожен m-бітовий шаблон має однакову ймовірність появи в послідовності. Якщо значення ймовірності, обчислене за допомогою тесту, становить $p < 0,01$, то ця двійкова послідовність не є повністю випадковою. інакше це випадково. Зауважте, що коли $m = 1$, перевірка періодичності стає перевіркою бітової швидкості.

12) Наближений ентропійний тест. Подібно до тесту на періодичність, цей тест зосереджений на підрахунку частоти всіх можливих накладених m-бітових шаблонів у вихідній послідовності бітів. Метою цього тесту є порівняння кількості перекривань двох послідовних блоків оригінальних послідовностей довжиною m і m+1 з показниками подібних блоків повністю випадкових послідовностей. Розраховане під час тестування значення ймовірності p повинно бути не менше 0,01. В іншому випадку ($p < 0,01$) двійкова послідовність не є абсолютно випадковою.

13) Тест кумулятивних сум. Перевіркою є максимальне відхилення (від нуля) протягом будь-якого раунду, визначене кумулятивною сумою вказаних (-1, +1) цифр у послідовності. Метою цього тесту є визначення того, чи є кумулятивна сума підпослідовностей, що виникають у вхідній послідовності, занадто великою або занадто малою порівняно з очікуваною поведінкою такої суми для повністю випадкової послідовності введення. Тому накопичення можна розглядати як довільний обхід. Для випадкових послідовностей відхилення від випадковості

повинно бути близьким до нуля. Для деяких типів послідовностей, які не є абсолютно випадковими, відхилення від нуля під час випадкового пошуку є дуже важливим. Якщо значення ймовірності, обчислене під час тестування, становить $p < 0{,}01$, то вхідна двійкова послідовність не є абсолютно випадковою.

14) Тест на випадкові відхилення. Суть цього тесту полягає в підрахунку кількості циклів з k рівним входам при випадковому повторенні кумулятивної суми. Випадкові ітерації кумулятивних сум починаються з часткових сум після перетворення послідовності (0,1) у відповідну послідовність (-1,+1). Випадковий цикл обходу складається із серії кроків одиничної довжини, які виконуються у випадковому порядку. Причому такі прогони починаються і закінчуються на одному елементі. Мета цього тесту полягає в тому, щоб визначити, чи відрізняється кількість знаходження певного стану в циклі від подібного числа при абсолютно випадковій послідовності введення. Цей тест насправді є набором із 8 тестів, які виконуються для кожного з 8 станів циклу (-4, -3, -2, -1 і +1, +2, +3, +4). Такі тести визначають ступінь випадковості вихідної послідовності за такими правилами: Якщо значення ймовірності, обчислене під час тестування, становить $p < 0{,}01$, то вхідна двійкова послідовність не є абсолютно випадковою.

15) Ще один тест на випадкове відхилення. Цей тест підраховує загальну кількість входів певного стану шляхом випадкового повторення кумулятивної суми. Мета полягає в тому, щоб визначити відхилення від очікуваної кількості відвідувань різних станів під час випадкового обходу. Насправді цей тест складається з 18 тестів для кожної умови (-9, -8, …, -1 і +1, +2, …, +9). На кожному етапі робиться висновок про випадковість вхідної послідовності. Якщо значення ймовірності, обчислене під час тестування, становить $p < 0{,}01$, то вхідна двійкова послідовність не є абсолютно випадковою.

c) DIEHARD. Тест Diehard — це набір статистичних тестів, призначених для вимірювання якості набору випадкових чисел. Вони були розроблені Джорджем Магсалією протягом шести років і вперше були випущені на компакт-диску з випадковими числами в 1995 році. Ці тести:

1) відстань дня народження. Вибираються випадкові точки з великими інтервалами. Відстань між точками має бути розподілена за Пуассоном. Цей тест названо на честь парадоксу дня народження.

2) Перестановки, що перетинаються. Аналізується послідовність із п'яти випадкових чисел. 120 перестановок мають відбутися з рівною ймовірністю.

3) Ранги матриць. Вибирається певна кількість бітів із певної кількості випадкових чисел, щоб сформувати матрицю з 0 і 1, і обчислюється ранг матриці.

4) Мавпячий тест. Певна послідовність бітів сприймається як про «слово». Рахується кількість повторюваних слів. Кількість слів, які не зустрічаються, має задовольняти відомому розподілу. Назва тесту походить від теорему про нескінченну мавпу.

5) Обчислення одиниць. Рахується кількість одиничних бітів у кожному байті. Перетворіть ці числа на «літери» та обчислюється кількість «слів» із 5 літер.

6) тест на паркування. Довільно розміщують одиничні кола в квадраті 100x100. Коло успішно припарковано, якщо воно не перекриває наявне успішно припарковане коло. Після 12 000 випробувань кількість успішно запаркованих кіл має відповідати нормальному розподілу.

7) Тест на мінімальну відстань Розміщують випадковим чином 8000 крапок у квадраті 10000x10000 і знаходять мінімальну відстань між парами. Квадрат цієї відстані має бути експоненціально розподілений із певним середнім.

8) випадковий тест м'яча. Випадково вибирають 4000 точок у кубі з 1000 ребрами. У кожній точці розміщують сферу, радіус якої дорівнює мінімальній відстані до іншої точки. Об'єм мінімальної кулі повинен бути експоненціально розподілений з певним середнім.

9) Тест на стиснення. Число 231 множать на випадкове значення з плаваючою комою (0,1), поки воно не дорівнюватиме 1. Повторюють це 100 000 разів. Кількість значень (0,1), необхідних для досягнення 1, має відповідати певному розподілу.

10) Дублікат контрольної суми. Генерується довга послідовність випадкових значень з плаваючою точкою в (0,1). Додається послідовність із 100 послідовних чисел із плаваючою комою. Суми повинні бути нормально розподілені з характерним середнім і дисперсією.

11) Runs test. Генерується довга послідовність випадкових значень з плаваючою точкою в (0,1). Рахуються висхідні та низхідні послідовності. Кількості мають відповідати певному розподілу.

12) The craps test. Запускаються 200 000 ігор у craps[16], рахуються виграші та кидки за гру. Кожен підрахунок має відповідати певному розподілу.

13) Більшість тестів у DIEHARD повертають значення p, яке має бути від нуля до одиниці. Якщо вхідний файл містить справді незалежні випадкові біти. Ці p-значення отримують за допомогою $p = F(X)$, де F є припущеним розподілом вибіркової випадкової змінної X – часто нормальним.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Аналіз недоліків алгоритму

Після проведеного тестування готового алгоритму було виявлено наступні недоліки:

- Використання дуальних бент-функцій;
- Швидкодія алгоритму доволі низька;
- Низька якість проходження тестування.

На рисунку 3.1 представлені початкові результати алгоритму. Алгоритм створює ключову послідовність певної довжини, обчислює час, що був витрачений і записує дану послідовність в файл для подальшої перевірки її тестами. Для приведення всіх еволюцій алгоритму до певного «спільного знаменника» вважатимемо довжину ключової вихідної послідовності такою, що після перекодування її у формат utf-8 на виході отримуємо файл розміром 1Мб[17].

На рисунку 3.1 представлена перша еволюція алгоритму. Потрібно зробити зауваження, що на рисунку можемо бачити, що час, витрачений на формування послідовності складає 704.4 секунди. Справа в тому, що це час витрачений на формування послідовності, що перекодована в файл, який склав розміром 100Кб, що у 10.24 рази менше за розмір, установлений в умові вище.

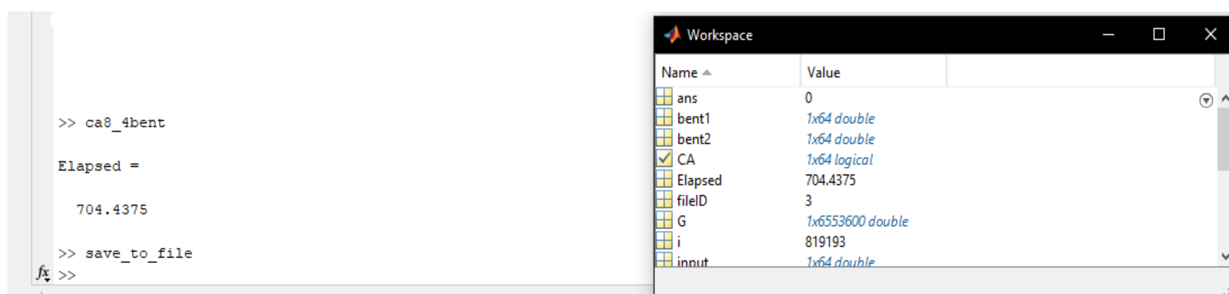

Рисунок 3.1 - Демонстрація часу, затраченого на створення послідовності в 100Кб

На жаль, алгоритм першої еволюції працює доволі повільно, тому провести повноцінний тест не виявляється можливим. Найбільш проблемним місцем можна вважати місце запису послідовності у змінну. В алгоритмі використовується

наступний варіант запису послідовності в змінну `G=[G, CA]`.Такий вид запису не є проблемою якщо значення G досить малі (до 1*104) але даний алгоритм має справу з послідовністю, що значно перевищує це значення. На рисунку 3.2 показаний графік залежності часу, що потрібний на виконання однієї такої операції, від розміру вхідних значень G (на графіку y) для Matlab версії від 2008 року. В той же час якщо брати графік, що показує ідентичну залежність, але на більш широкому проміжку і в більш сучасній версії Matlab, то можна побачити, що час, який потрібен на додавання одного елемента в список таким способом зростає набагато швидше і раптовіше. Приклад такого графіку продемонстрований на рисунку 3.3.

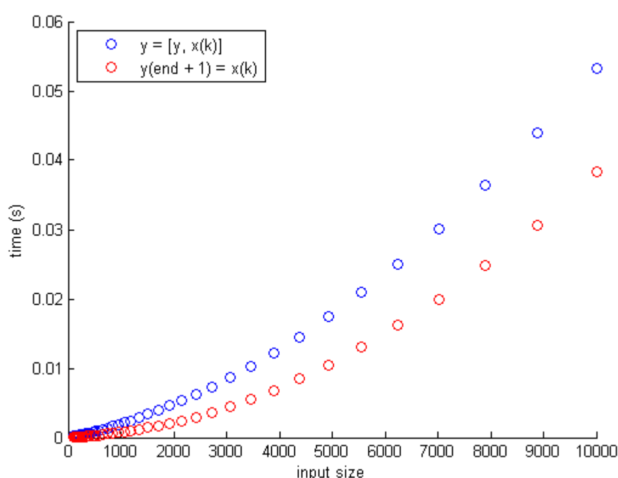

Рисунок 3.2 - Графік залежності часу від розміру вхідних даних у Matlab R2008a

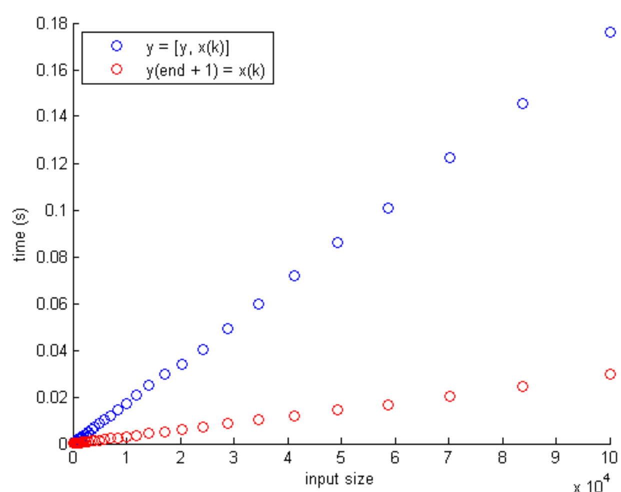

Рисунок 3.3 - Графік залежності часу від розміру вхідних даних у Matlab R2013a

Якщо оцінювати час, який би був затрачений на створення файлу з послідовністю розміром 1Мб, то можна висловити припущення, що це зайняло б більше трьох годин. Час, витрачений на створення послідовності, обчислено, потрібно обчислити кількість пройдених тестів. Як було сказано раніше, для оцінки використовується серія з 15 тестів, створених Національним інститутом стандартів і технологій. Чим більше тестів пройдено, тим більше послідовність схожа на дійсно випадкову. На рисунку 3.4 представлений результат в 11 пройдених тестів.

```
P=0.5934183066645216
P=0.5496515157342672
P=0.7117337778485726
P=0.9165372878377065
P=0.9614981113906811
P=0.921513814921687
P=0.7657615804670187
P=0.712951503143521
P=0.2838654827560125
P=0.007066952874102466
P=0.035114528882050286
P=0.030527076611293356
P=0.021506990727283324
P=0.03881766702431297
P=0.04014393690286134
P=0.031326097224965536
P=0.03305776169374059
P=0.04603644133955436

SUMMARY
-------
monobit_test                              1.0                  PASS
frequency_within_block_test               0.11732636796593947 PASS
runs_test                                 0.1800422361867566 PASS
longest_run_ones_in_a_block_test          0.43661792984736286 PASS
binary_matrix_rank_test                   0.6574813684575579 PASS
dft_test                                  0.1642761080545318 PASS
non_overlapping_template_matching_test    1.00000000892819   PASS
overlapping_template_matching_test        0.0                  FAIL
maurers_universal_test                    0.863346658591012  PASS
linear_complexity_test                    0.0                  FAIL
serial_test                               0.14709301287071835 PASS
approximate_entropy_test                  0.29117468024431653 PASS
cumulative_sums_test                      0.6249402192404576 PASS
random_excursion_test                     0.0018043707332569784 FAIL
random_excursion_variant_test             0.007066952874102466 FAIL

Total count of passed tests = 11

C:\Users\hate_win\Desktop\diploma\nist>
```

Рисунок 3.4 - Результат проходження NIST-тестів алгоритмом першої еволюції

Немає певної шкали оцінювання, яка б показала скільки пройдених тестів відповідають не випадковій послідовності, тому надалі результати будуть сприйматись таким чином, що дозволяє оцінити яка з послідовностей краще, що означає кращий алгоритм її генерації. Кількість в 11 пройдених тестів є середнім значенням, тому далі будемо відштовхуватись від неї.

3.1.1 Удосконалення швидкодії

Після проведеного аналізу першим за важливістю недоліком вирішено вважати швидкодію алгоритму. Як було зазначено раніше, найбільше часу витрачається саме на моменті запису вихідної послідовності *G* у змінну для збереження.

Варто зазначити, що змінна G це масив з {0,1}. На кожній ітерації алгоритму до неї в кінець записується список довжиною, яка дорівнює довжині бент-функції, що використовується в алгоритмі.

На рисунках 3.2 та 3.3 було представлено порівняння двох методів додавання елементів до одновимірного масиву даних.

```
G = [G, CA];
```

Такий запис використовується в алгоритмі на даний момент. Він досить повільний, якщо розмова йде про великі розміри змінних.

На графіках було проведено порівняння з набагато більш швидким методом додавання елементів до масивів. Додавання елемента в кінець списку не потребує рекурсивного звернення до існуючої змінної, що сильно розвантажує пам’ять та зменшує час на виконання команди. Якщо на графіках указана загальна форма запису такої команди, то для існуючого алгоритму найкращим варіантом запису можна вважати:

```
for i=1:8:length(CA)
    fwrite(fid,uint8(bi2de(CA(i:i+7))));
end
```

Також слід зазначити, що даний запис несе в собі дуже важливе удосконалення іншої частини алгоритму. Раніше завантаження послідовності в файл відбувалось після отримання готової послідовності, але якщо змінити цю форму запису на паралельний з обчисленням блоку CA, що містить частину фінальної послідовності на даному кроці алгоритму, то можна автоматизувати процес збереження при цьому не сповільнюючи алгоритм.

На рисунку 3.5 продемонстрований результат після заміни методу запису результатів у файл. Можна побачити, що для фінального файлу вагою у 100Кб знадобилось всього 6.28 секунд, що майже у 113 разів швидше, ніж у стандартного алгоритму.

```
>> ca8_4bent

Elapsed =

    6.2812

>>
```

Name	Value
ans	0
bent1	1x64 double
bent2	1x64 double
CA	1x64 logical
Elapsed	6.2813
fid	3
G	[]

Рисунок 3.5 - Демонстрація часу, затраченого на створення послідовності в 100Кб алгоритмом другої еволюції

Тепер для отримання вихідного файлу розміром 1Мб потрібно значно менше часу, тому надалі алгоритм буде використовуватись для створення файлу розміром 1Мб і подальшою його перевіркою за допомогою статистичних тестів. На рисунку 3.6 продемонстрована кількість часу, яка потрібна алгоритму другої еволюції для отримання файлу розміром 1Мб.

```
>> ca8_4bent

Elapsed =

   46.8906

>>
```

Workspace

Name	Value
ans	0
bent1	1x64 double
bent2	1x64 double
CA	1x64 logical
Elapsed	46.8906
fid	4
G	[]

Рисунок 3.6 - Демонстрація часу, затраченого на створення послідовності в 1Мб алгоритмом другої еволюції

Файл заданого розміру був створений всього за 46.89 секунд, що навіть в порівняння не йде з очікуваними трьома годинами раніше.

Після переводу алгоритму генерації ключових послідовностей на 4-бент-функції були отримані наступні результати, представлені на рисунку 3.7. Можемо бачити, що завдяки тому, що збільшилась вихідна послідовність тепер тестам набагато легше визначити чи є послідовність, справді, випадковою.

```
P=0.14275535308788428
P=0.108735767636993
P=0.20982043672527104
P=0.23104516241393896
P=0.24807234800645714
P=0.5125653142151565
P=0.5205228832757727
P=0.44945582731421674
P=0.799846105662473З
P=0.3748571426833401
P=0.1432349075246697
P=0.05390255716938722
P=0.044171344908442614
P=0.04860657067343573
P=0.01122886027995977
P=0.002309478490420638
P=0.0003452106153455428
P=1.389104578055706e-05

SUMMARY
-------
monobit_test                            7.7889705277772879e-11 FAIL
frequency_within_block_test             0.0014312174691939082 FAIL
runs_test                               0.0                   FAIL
longest_run_ones_in_a_block_test        0.5074880512485193 PASS
binary_matrix_rank_test                 0.8022999412026713 PASS
dft_test                                0.19074266687346952 PASS
non_overlapping_template_matching_test  1.0000466213966237 PASS
overlapping_template_matching_test      0.20028521464493465 PASS
maurers_universal_test                  0.10995012470799276 PASS
linear_complexity_test                  0.7408361379488465 PASS
serial_test                             5.271094618243967e-30 FAIL
approximate_entropy_test                1.0450280622894797e-29 FAIL
cumulative_sums_test                    5.2388315907592187e-11 FAIL
random_excursion_test                   0.1590963215116591 PASS
random_excursion_variant_test           1.389104578055706e-05 FAIL

Total count of passed tests = 8

C:\Users\hate_win\Desktop\diploma\nist>
```

Рисунок 3.7 - Результат проходження NIST-тестів алгоритмом другої еволюції

Значення у 8 пройдених тестів це доволі низький результат, що означає, що наступним кроком буде покращення алгоритму генерації. Таке різке падіння кількості пройдених тестів можна пояснити тим, що тепер у тестів є більше даних, які вказують на випадковість послідовності.

3.1.2 Удосконалення результатів тестування

Наступним недоліком можна вважати використання дуальних бент-функцій. На даний момент такі функції стають доволі популярними для вирішення певних задач криптографії, тому не можна прогнозувати точно коли алгоритми генерації

випадкових величин на їх основі будуть зламані, але завдяки постійному розвитку з'являються нові їх види, які все ще шукають своє призначення. Одним з таких нових видів можна вважати бент-функції з потужністю алфавіту 4[18].

На відміну від дуальних бент-функцій, де послідовність складається тільки з нулів та одиниць, в 4-бент функціях результат представлений у вигляді послідовностей з {0, 1, 2, 3}, саме цю їх особливість буде використано для покращення алгоритму.

4-бент-функції, які будуть використовуватись надалі мають довжину відмінну від дуальних. Якщо 2-бент-функції мали довжину 64, то 4-бент мають довжину 16 біт. Через зміну довжини функцій в алгоритмі треба змінити розмір Mg послідовності та забезпечити безперебійну роботу методу з новими послідовностями бітів.

На рисунку 3.8 показаний час, витрачений на формування послідовності основаної на 4-бент-функціях та запису її у файл розміром 1Мб.

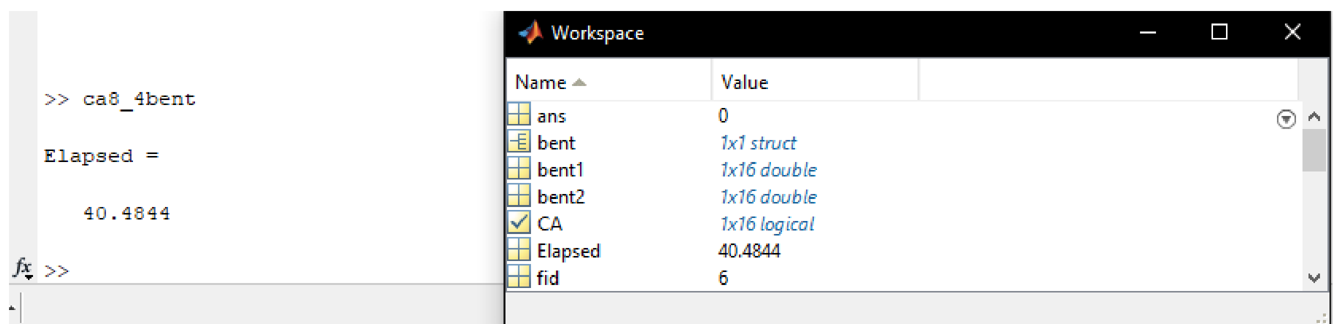

Рисунок 3.8 - Демонстрація часу, затраченого на створення послідовності в 1Мб алгоритмом з використанням 4-бент-функцій

Час в 40.48 секунд свідчить про те, що для створення файлу розміром 1Мб алгоритму на основі 4-бент-функцій потрібно навіть менше часу, ніж ідентичному алгоритму з використанням дуальних функцій. Хоча пришвидшення і незначне, але при створенні більш довгих послідовностей різниця в 1.15 разів може зіграти велику роль.

Якщо говорити про результати тестування, то на рисунку 3.9 представлене коротке резюме проходження файлом з вихідною послідовністю серії з 15 NIST-тестів.

```
P=0.9749544914430657
P=0.7483189418226386
P=0.7142185170646251
P=0.7907026332849842
P=0.822468248973117
P=0.5311521077405561
P=0.5242644670904282
P=0.39422473880260067
P=0.1471184712622803
P=0.08281801631317581
P=0.0325633943197405
P=0.0072296274455730496
P=0.0014719019028213737
P=0.0008651790012684687
P=0.001701369171731529
P=0.002275973169437916
P=0.004992819656684482
P=0.02016661479832961

SUMMARY
-------
monobit_test                             0.8122689192015151 PASS
frequency_within_block_test              -0.0010947863715087033 FAIL
runs_test                                0.385080066351219З PASS
longest_run_ones_in_a_block_test         0.3717306573771816 PASS
binary_matrix_rank_test                  0.2787290142084916 PASS
dft_test                                 2.208916228276721e-232 FAIL
non_overlapping_template_matching_test   0.9999999065323055 PASS
overlapping_template_matching_test       0.0                FAIL
maurers_universal_test                   0.0                FAIL
linear_complexity_test                   0.0                FAIL
serial_test                              0.6900577010384529 PASS
approximate_entropy_test                 0.7746363523824039 PASS
cumulative_sums_test                     0.952495624241742  PASS
random_excursion_test                    0.006940695203899716 FAIL
random_excursion_variant_test            0.0008651790012684687 FAIL

Total count of passed tests = 8

C:\Users\hate_win\Desktop\diploma\nist>_
```

Рисунок 3.9 - Результат проходження NIST-тестів алгоритмом третьої еволюції

З результатів можна побачити, що кількість пройдених тестів не змінилась. При цьому час, що витрачено на побудову послідовності зменшився, хоча і незначно. Як було сказано раніше, результат у 8 пройдених тестів не є задовільним, тому було прийнято рішення додати у алгоритм новий метод, що дозволить покращити цей результат. Для того, щоб не збільшувати обчислювальну складність алгоритму було прийнято рішення не додавати нових

операцій, натомість вирішено змінити поведінку послідовностей шляхом додавання методу розміщення їх елементів у сітці. Після аналізу можливих варіантів було вирішено зупинитись на сітці NESW[19]. Дослівно це розшифровується як North, East, South, West. Суть методу заключається в скручуванні послідовності в матрицю 4*4 заповнюючи її з лівого нижнього кута слідуючи за годинниковою стрілкою (далі спіралізація). Після цього матриця «розгортається», починаючи з випадкового кута і утворює нову послідовність. На рисунку 3.10 представлений приклад заповнення сітки і чотири вихідні послідовності. Для простоти візуалізації було взято матрицю 3*3.

3	4	5	вхідна:123456789
2	9	6	вихідна1:123456789
1	8	7	вихідна2:345678129
			вихідна3:567812349
			вихідна4:781234569

Рисунок 3.10 – Приклад сітки алгоритма NESW

Після підставлення спіралізації у алгоритм він набуває свого фінального вигляду. На рисунку 3.11 представлена блок-схема алгоритму фінальної еволюції.

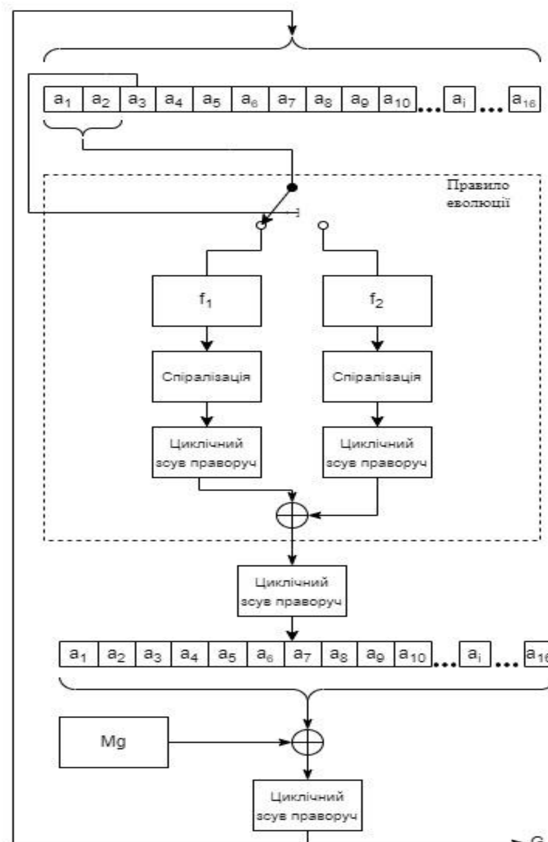

Рисунок 3.11 – Блок-схема алгоритму фінальної еволюції

Даний алгоритм є фінальним результатом удосконалення, тому потрібно перевірити час витрачений на побудову послідовностей і результат проходження тестів.

Для початку створюється послідовність, що при записі в файл дасть розмір 1Мб. На рисунку 3.12 представлений результат того, скільки часу на це знадобилось.

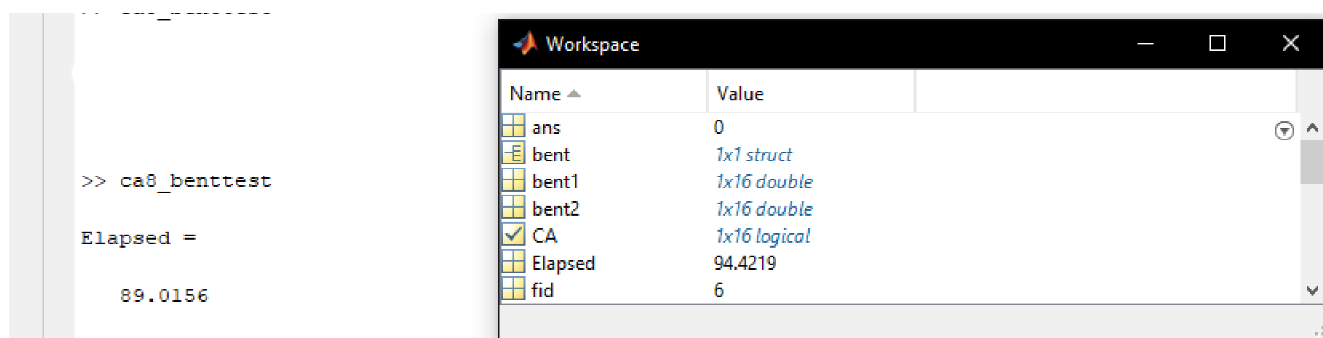

Рисунок 3.12 - Демонстрація часу, затраченого на створення послідовності в 1Мб алгоритмом фінальної еволюції

З рисунку можна судити, що спіралізація сильно погіршила швидкість створення послідовностей. Насправді швидкодія алгоритму зменшилась більше, ніж в два рази. Якщо порівнювати з попередньою еволюцією, то цей результат здається провальним, однак якщо порівнювати з початковою версією алгоритму, то швидкість генератора збільшилась. На рисунку 3.13 представлена кількість часу, яка потрібна фінальному алгоритму для побудови послідовності довжиною в 100Кб (саме такої довжини початковий алгоритм будував послідовність).

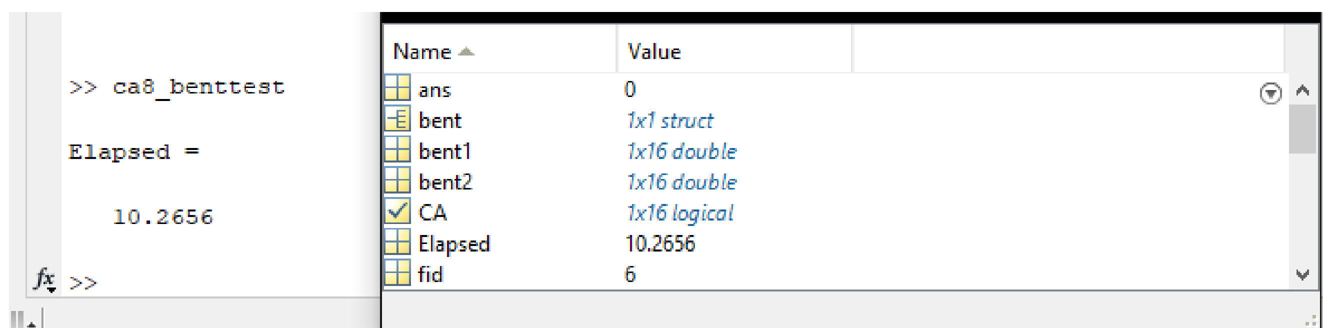

Рисунок 3.13 - Демонстрація часу, затраченого на створення послідовності в 100Кб агоритмом фінальної еволюції

З інформації на рисунку можна судити, що новому алгоритму для побудови послідовності такої ж довжини потрібно майже в 69 разів менше часу.

Залишилось перевірити яку кількість тестів проходить послідовність у 1Мб. На рисунку 3.14 представлений результат проходження послідовністю тестів.

```
P=0.17668591781470055
P=0.0658470894706018
P=0.01734402286945136
P=0.03131757527734218
P=0.15838723067171592
P=0.36830726189545654
P=0.46879758624609663
P=0.44518645840327636
P=0.47722796679685786
P=0.2901067948925675
P=0.24430923022047238
P=0.9941019213254952
P=0.3144842846365921
P=0.09946780726811179
P=0.028315689602325837
P=0.008616839450357594
P=0.0022766643535735626
P=0.005956484648499028

SUMMARY
-------
monobit_test                             0.20024716998628847 PASS
frequency_within_block_test              1.0904022899637447 PASS
runs_test                                0.041157923812754175 PASS
longest_run_ones_in_a_block_test         0.15201183960225634 PASS
binary_matrix_rank_test                  0.5289783498573701 PASS
dft_test                                 0.0                FAIL
non_overlapping_template_matching_test   0.9985888603536105 PASS
overlapping_template_matching_test       0.10591407268673976 PASS
maurers_universal_test                   0.6697816913560224 PASS
linear_complexity_test                   0.5973944857842832 PASS
serial_test                              0.15480887159584994 PASS
approximate_entropy_test                 0.15597533281502798 PASS
cumulative_sums_test                     0.27679906611786276 PASS
random_excursion_test                    0.007048519614385253 FAIL
random_excursion_variant_test            0.0022766643535735626 FAIL

Total count of passed tests = 12

C:\Users\hate_win\Desktop\diploma\nist>sp800_22_retests.py
```

Рисунок 3.14 - Результат проходження NIST-тестів алгоритмом фінальної еволюції

Хоча результат у 12 тестів можна вважати вищим за середній, але кількість пройдених тестів можна ще збільшити. Для цього потрібно дати користувачу можливість самостійно вибирати пари бент-функцій та проводити порівняльний аналіз послідовностей, що були створені на їх основі.

3.2 Створення графічного інтерфейсу

При дослідженні бент-функцій було висловлено припущення, що випадковість вихідної послідовності алгоритму може бути залежною не тільки від вхідної послідовності, але і від бент-функцій, що використовуються. Для того, щоб підтвердити цю гіпотезу було створено додаток з графічним інтерфейсом мовою Python, який дозволяє перебрати пари бент-функцій для пошуку пари, що на виході дасть найбільшу кількість пройдених тестів.

Графічний інтерфейс програми, представленої на рисунку 3.15 складається з наступних елементів:

- Двох полів вводу
- Чотирьох кнопок
- Консолі
- Додаткових полів з текстом

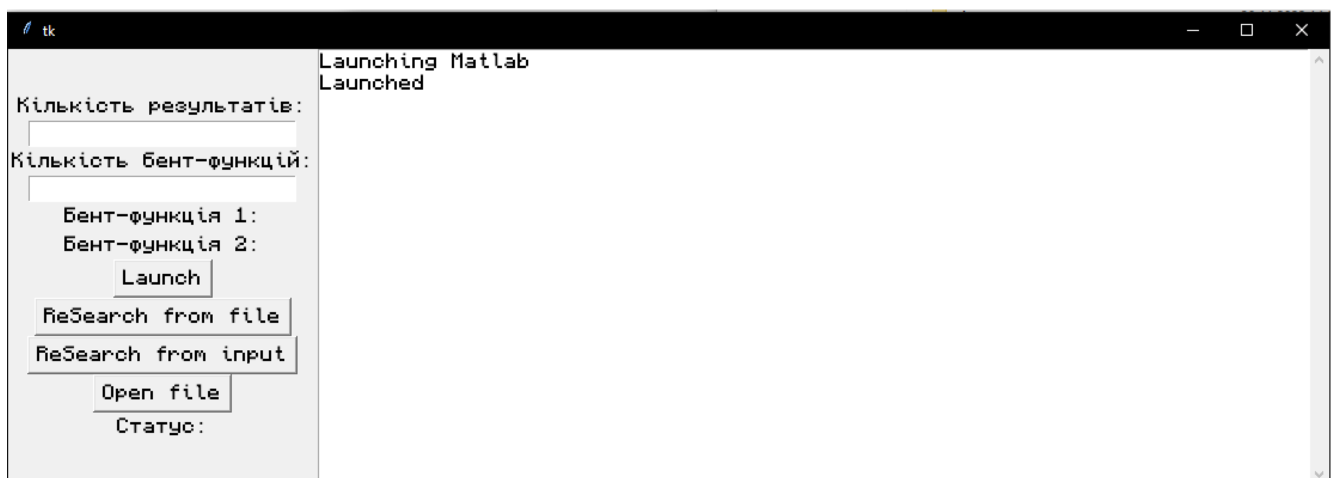

Рисунок 3.15 – Графічний інтерфейс додатку для аналізу властивостей бент-функцій

Перше поле вводу потрібно для того, щоб користувач міг зазначити скільки найкращих результатів після обчислення його цікавить. Друге поле вводу зазначає скільки бент-функцій з пам’яті програми буде взято.

Кнопка «Launch» запускає функцію пошуку. Після натискання клавіші програма зчитує значення полів вводу, бере з пам’яті кількість бент-функцій указану користувачем і формує з них всі можливі пари. Кожна з цих пар

використовується для запуску алгоритму, описаному в попередньому пункті роботи. Після того, як алгоритм закінчить свою роботу на одній парі функцій, одразу проводиться перевірка статистичними тестами. Після перевірки результат та пара бент-функцій записується в змінну. Функція запускається з наступною парою бент-функцій.

Після того, як всі можливі пари були перевірені, створюється файл, в який завантажується та кількість найкращих результатів, яку вказав користувач. Кнопка «Open file» потрібна для перегляду цього файлу.

Кнопка «Research from file» потрібна для більш поглибленого аналізу пар бент-функцій, записаних в файлі. Функція, яка викликається при натисканні кнопки «Launch» створює послідовності розміром в 500Кб для економії часу. В той же час функція, яка викликається при натисканні кнопки «Research from file» створює файли розміром 1Мб.

При натисканні кнопки «ReSearch from input» відкривається нове вікно, що продемонстровано на рисунку Це вікно потрібно для ручного введення та перевірки бент-функцій.

ВИСНОВОК

Результатом виконання дослідницької роботи з теми Розробка генератора ключових послідовностей на основі клітинних автоматів і бент-функцій багатозначної логіки став алгоритм створення псевдовипадкових послідовностей, який проходить в середньому 13-15 статистичних Nist-тестів. Також було створено додаток з графічним інтерфейсом мовою Python, який проводить дослідження пар 4-бент-функцій для пошуку пар, використання яких в алгоритмі призводить до найбільш високої кількості пройдених тестів.

Графічний інтерфейс додатку був створений за допомогою модуля Tkinter[20], що використовується у невеликих користувацьких додатках через свою простоту у використанні та велику кількість літератури для його вивчення та опанування.

Також був використаний стандартний модуль threading, який дозволяє програмі використовувати більше, ніж один потік процесора, для збільшення швидкодії програми.

Під час виконання роботи були вирішені наступні задачі:

Було проведено аналіз предметної області та існуючих програмних рішень для виконання подібних задач;

Переведено існуючий алгоритм на бент-функції багатозначної логіки для усунення можливості проведення атаки Берлекемпа-Мессі

Проведено оцінку швидкодії алгоритму та стохастичних властивостей вихідної послідовності

Виконано удосконалення алгоритму з використанням спіралізації

Проведена розробка та тестування програмного застосунку для генерації ключових послідовностей.

Як підсумок можна зазначити, що удосконалений алгоритм має потенціал ще швидше виконувати обчислення, але така можливість може бути реалізована переходом на більш потужну машину, яка забезпечить багатопотоковість в роботі

системи. Також можна удосконалити графічний інтерфейс та додати більше підказок для користувачів. На даний момент додаток повністю готовий до користування та може принести багато користі для тих, хто вивчає бент-функції та шукає закономірності між парами бент-функцій та результатами, які дають алгоритми з їх використанням.

ПЕРЕЛІК ПОСИЛАНЬ

1. Хименко М. В. Розробка пристрою заглушення сигналів стільникових: кваліфікаційна робота бакалавра. Одеський нац. політехн. ун-т. Одеса, 2020.

2. Elaine Barker, Lawrence Bassham, Alexander Calis, Chris Celi, Tim Hall. Random Bit Generation RBG. URL: https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software

3.Casey E. Berlekamp-Massey Algorithm. URL: https://www-users.cse.umn.edu/~garrett/students/reu/MB_algorithm.pdf

4. Bui H. T., Al-Sheraidah A. K., Wang Y. New 4-transistor XOR and XNOR designs. The 2nd IEEE Asia Pacific Conference. Materials of The 2nd IEEE Asia-Pacific Service Computing Conference 11-14 December 2007.

5. Нефедов В.І. Теорія електрозв’язку Київ: Магнолія, 2018.

6. ISO/IEC 18031:2011. Information technology — Security techniques — Random bit generation. URL: https://www.iso.org/standard/54945.html

7. Мазур К., Сіра І. Perfect numbers and Mersenne numbers. URL: https://dspace.hnpu.edu.ua/handle/123456789/9133

8. Гапак О. М. Період генератора голлманна на основі регістрів зсуву зі зворотним зв’язком за перенесенням. URL: https://visnyk.vntu.edu.ua › article › download

9. Шнайєр Б. Прикладная криптография. Київ: Тріумф, 2002.

10. Соколов А.В., Хименко М.В. Тестирование генераторов ключевых последовательностей на основе совершенных алгебраических конструкций. *Наука та суспільне життя України в епоху глобальних викликів людства у цифрову еру: Матеріали Міжнародної науково-практичної конференції.* Одеса, 2021.

11. Sokolov A.V. Properties of the full class of quaternary bent-functions of two variables. Одеський нац. політехн. ун-т. Одеса, 2022.

12. Попов Б. О. Розв'язування математичних задач у системі комп'ютерної алгебри Maple V. Київ : ViP, 2001. 312 с.

13. Becker J.. Unicode manifesto. URL: http://utf8everywhere.org/

14. L'Ecuyer P., Simard R.. ACM Transactions on Mathematical Software. URL: https://doi.org/10.1145/1268776.1268777

15. Université de Montréal. URL: https://www.umontreal.ca/en/

16. Holmes S.. The Craps principle 10/16. URL: statweb.stanford.edu.

17. International Standard IEC 60027-2. Definitions of the SI units: The binary prefixes. URL: https://physics.nist.gov/cuu/Units/binary.html

18. Mesnager S. Bent Functions Fundamentals and Results. California: Springer, 2016.

19. Mukhamedjanov D.. Improving pseudorandom generator on cellular automata with bent functions. *Conference on Computer Science and Information Systems*. 2018.

20. Python interface to Tcl/Tk. URL: https://docs.python.org/3/library/tkinter.html