Харківський національний університет імені В.Н. Каразіна
Факультет комп'ютерних наук
Безпека інформаційних систем і технологій

«Допущено до захисту»
Зав. кафедрою БІСТ
Рассомахін С.Г.

_____
« » 2021р.

Пояснювальна записка

до кваліфікаційної роботи бакалавра
спеціальність: 125 - Кібербезпека

на тему: «Моделі та методи стеганографічних перетворень із застосуванням векторних зображень»

оцінка « » Керівник проф. Кузнецов О. О.
Голова ЕК Рецензент проф. Толстолузька О. Г.
Доценко С.І. _____ Виконавець : студентка групи КБ-41
Кононченко А. В.

Харків – 2021

РЕФЕРАТ

Кваліфікаційна робота бакалавра, 57 с., 31 рис., 6 табл., 41 джерело.

Робота містить вступ, 4 розділи, висновки, список джерел інформації та 4 додатки.

У роботі проведено дослідження методів приховування даних у векторні зображення, особливостей їх математичних моделей, які обумовлюють можливість виконання стеганографічних перетворень із вбудовування та вилучення інформаційних послідовностей. Виявлено такі етапи проведення вбудовування як попередня обробка контейнера, розбиття кривих Без'є третього ступеня відповідно до значень елементів вхідних двійкових даних, збереження стеганоключів та зміненого зображення. При вилученні виконуються такі кроки як обробка вмісту стегоконтейнера та відтворення повідомлення шляхом об'єднання отриманих при вбудовуванні сегментів у початкову криву згідно до значень стеганоключів.

Мета роботи полягала в аналізі існуючих методів приховування даних у геометричні елементи векторної графіки, зокрема у криві Без'є третього ступеня, власна програмна реалізація мовою програмування JavaScript та їх експериментальні дослідження щодо таких характеристик як швидкість вбудовування та вилучення, коефіцієнти збільшення стегоконтейнерів та їх спотворення, а також показники стійкості до афінних перетворень. Розроблена програмна реалізація дозволяє виконувати операції приховування та відтворення повідомлень у файлах формату SVG.

Ключові слова: ПРИХОВУВАННЯ ІНФОРМАЦІЇ, СТЕГАНОГРАФІЯ, ВЕКТОРНА ГРАФІКА, КРИВІ БЕЗ'Є, JAVASCRIPT, БЕКЕНД

ABSTRACT

Qualifying work of a bachelor, 57 pages, 31 images, 6 tables, 41 references.

The work consists of the introduction, 4 sections, conclusions, a list of references and 4 applications.

The work conducted a study of methods of concealing data into vector images, the features of their mathematical models, which make it possible to perform steganographic transformations for concealing and extracting information sequences. Such stages of data hiding as pre-processing of the container, splitting the Bezier curves of third-order according to the values of the elements of the input binary data, saving the stegokeys and the modified image were discovered. During extracting, such steps as processing the contents of the stegocontainer and restoring the message by combining the segments into the initial curve according to the values of the stegokeys are performed.

The purpose of the work is to analyze existing methods of data concealing into geometric elements of vector graphics, in particular the Bezier curves of the third order, the own programming implementation in the programming language JavaScript, and their experimental studies regarding such characteristics as speed of concealing and extraction, stegocontainer size growth, and distortion factors, as well as the resistance to the affine transformations. Developed software implementation allows performing the operations of hiding and extracting messages in SVG files.

Key words: INFORMATION HIDING, STEGANOGRAPHY, VECTOR GRAPHICS, BEZIER CURVES, JAVASCRIPT, BACKEND

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕННЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

JPEG – Joint Photographic Experts Group
GIF – Graphics Interchange Format
PNG – Portable Network Graphics
SVG – Scalable Vector Graphic
AI – Adobe Illustrator
FH – Macromedia Freehand
CDR – Coreldraw Vector
EPS – Encapsulated PostScript
XML – Extensible Markup Language
HTML – Hypertext Markup Language
DOM – Document Object Model
JS – JavaScript
IDE – Integrated Development Environment

ВСТУП

На сьогоднішній день актуальною є задача розробки методів для прихованої передачі інформації. Важливість такої передачі полягає у факті неможливості забезпечення 100% захищеності каналу зв’язку, тобто необхідно впроваджувати способи комунікації, які б ускладнювали отримання повідомлення несанкціонованим користувачем.

Вивченням таких методів займається стеганографія і найбільш поширеними способами її застосування стали файли мультимедіа, особливо у час стрімкого технологічного розвитку суспільства. Так, широкого відомими є методи кодування інформації у цифрових зображеннях растрової графіки, де одиницею аналізу є піксель. Однак більшість методів вразливі до великої кількості атак, що приводять до руйнування секретного повідомлення.

Вирішенням такої проблеми можуть стати методи вбудовування інформації у векторні зображення, які будуть стійкі до різного роду афінних перетворень. У зв’язку з тим, що одиницею векторної графіки є геометричні елементи, можливим є проведення стеганографічних перетворень із вбудовування та вилучення секретної інформації.

Об’єктом дослідження є процес приховування інформації у векторні зображення. Предметом дослідження є методи стеганографічних перетворень у точково-задані криві, які здатні забезпечувати стійкість до різного роду афінних перетворень.

Метою роботи є дослідження моделей та методів стеганографічних перетворень елементів векторних зображень, їх програмна реалізація та експериментальні дослідження. Мета досягається шляхом аналізу існуючих методів вбудовування секретних даних у елементи векторної графіки.

1 АНАЛІЗ ТА ДОСЛІДЖЕННЯ ФОРМАТІВ ВЕКТОРНИХ ЗОБРАЖЕНЬ, АЛГОРИТМІВ ПРИХОВУВАННЯ ІНФОРМАЦІЇ У ВЕКТОРНОМУ ЗОБРАЖЕННІ

Усі цифрові зображення поділяються на два основних типи: растрові цифрові зображення та векторні цифрові зображення. Перші представляють собою зображення, які визначаються через упорядковану сітку пікселів. Найбільш поширеними є растровими розширеннями є Joint Photographic Experts Group (JPEG), Graphics Interchange Format (GIF) і Portable Network Graphics (PNG). Другий же тип зображень будується на основі математичних форм, які визначають такі геометричні елементи як прямокутники, криві, відрізки, окружності та баготокутники. Окрім цього елементи доповнюються такими параметрами як градієнти, колір, відтінки, товщина тощо [1]. Найбільш відомим форматом векторної графіки є SVG, а також AI, FH, CDR, EPS та інші [2].

1.1 Аналіз форматів векторної графіки, обґрунтування показників та критеріїв для їх оцінки

На відміну від растрової, векторна графіка складається з траєкторій, які визначаються початковою та кінцевою точками, а також іншими точками, кривими та кутами впродовж них [3]. Такими траєкторіями можуть бути: лінія, квадрат, трикутник або криволінійна форма. Шляхи, визначені досить невеликим переліком геометричних елементів, можуть представляти собою як прості креслення, так і найскладніші схеми.

Оскільки векторні зображення не складаються з деякої визначеної кількості точок, вони можуть бути масштабовані до більшого розміру без втрати якості. Так, якщо над растровим зображенням провести зумування або стиснення, то воно набути блокового або «пікселеризованого» виду. У випадку з векторною графікою при аналогічних операціях краї кожного геометричного об’єкту залишаються гладкими. Поширені типи векторної графіки включають файли таких редакторів як

Adobe Illustrator, Macromedia Freehand, CorelDraw, Adobe (EPS, PDF), Inkscape та інші.

Векторний формат файлів редакторів Adobe Illustrator (.ai) та CorelDraw (.cdr) забезпечує надзвичайно високу якість створюваних зображень, однак низка параметрів, що використовується виключно даною реалізацією, значно понижує сумісність з іншими програмами. Окрім цього бренди Adobe та CorelDraw мають велику кількість версій своїх графічних редакторів, створювані проекти яких зовсім не сумісні з попередніми виданнями.

Обидва формати належать до закритого типу, звідси – обмежений функціонал з іншими редакторами: деякі налаштування елементів векторного зображення та ефекти втрачаються під час конвертації; можливий надто довгий процес обробки файлів з надто громіздким для програми, що використовується, розширенням тощо.

Використання файлів AI та CDR рекомендовано при створенні рекламних макетів, логотипів, візиток, схем та креслень для власного використання або для друкування. Вбудовування ж такого формату в HTML-сторінки неможливе.

Macromedia Freehand – це подібний до Adobe Illustrator пакет для редагування векторної графіки з форматом .FHx (де x – номер версії). Сам редактор має досить стандартний функціонал з оберненою підтримкою версій проектів та файлів інших пакетів графіки, однак його файли не можуть бути використані у веб-застосунках.

Ще одним форматом векторної графіки вважається відкритий стандарт ISO 32000 PDF з підтримкою різних шрифтів, зображень та даних мультимедіа. Файли PDF мають невеликий розмір, зручні для друкування без спотворень майже на будь-якій апаратурі. Основною проблемою є неможливість повноцінного редагування та складність створення нових об’єктів.

EPS-файли часто застосовуються як стандартні формати векторних зображень з можливістю застосування скриптів PostScript та з широкою підтримкою всіх векторних редакторів і використовуються для друкування або проміжних робіт з проектування макетів та інших об’єктів мистецтва. Такий

формат успішно оброблює великі об’єми даних, однак цим і обумовлений значний розмір файлів. Поширеною є практика створення проектів у EPS-форматі та подальше їх конвертування в об’єкти растрової графіки (PNG, JPEG, GIF). Вбудовування же у код веб-сторінки неможливе.

Найбільш гнучким рішенням є застосування SVG-файлів. По суті, зображення у SVG являю собою основану на XML мову розмітки, а також набір пов’язаних інтерфейсів графічних сценаріїв [4]. Саме завдяки своїй структурі можлива просто інтеграція таких векторних зображень у веб-сторінки, адже у такому випадку зменшується кількість звернень до серверу , що веде до збільшення швидкості обробки даних сторінки. На рис. 1.1 зображено підтримку різними браузерами SVG формату в якості елементу embed або object у HTML-коді [5].

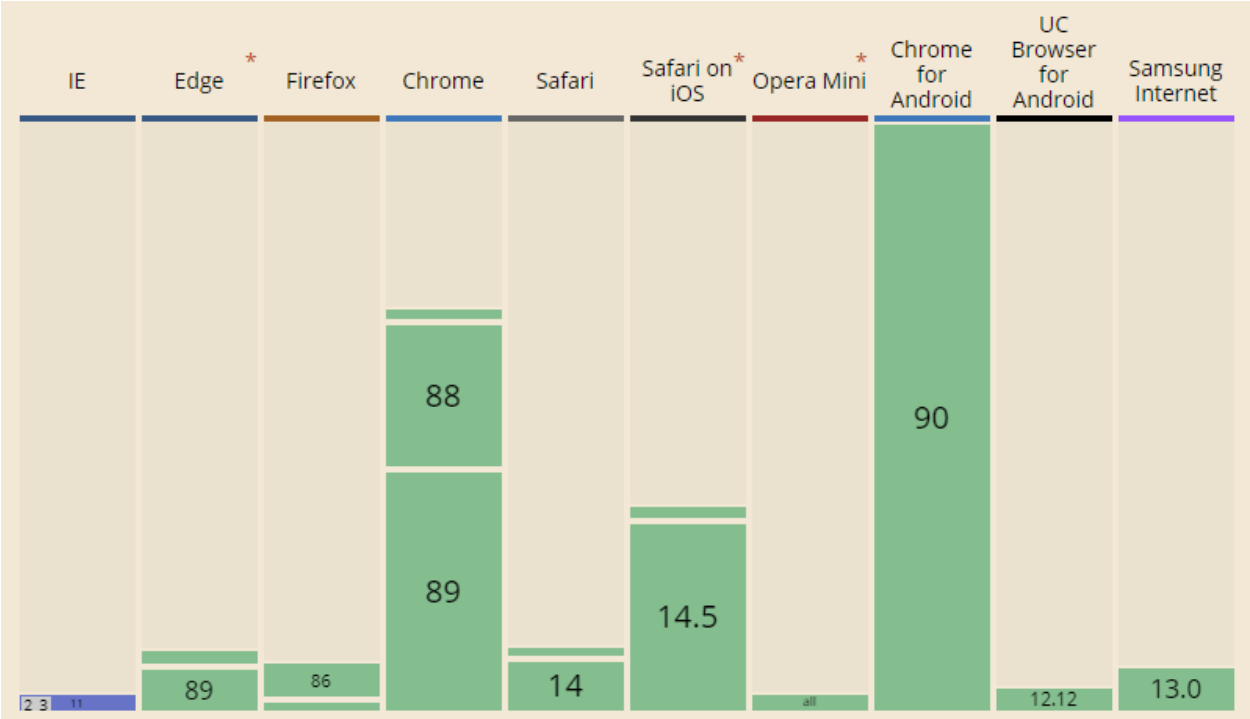

Рисунок 1.1 – Підтримка формату SVG різними браузерами

Широка підтримка формату SVG браузерами та його популярність відіграють важливу роль при спробі досягнення головного завдання стеганографії: приховування самого факту існування секретних даних при їх передачі, зберіганні або обробці [6].

1.2 Аналіз методів приховування інформації у векторних зображеннях

Методи приховування інформації в нерухомих зображеннях вивчаються вже давно і існує багато наукових публікацій в цій галузі, наприклад, [7-10]. Втім, більшість відомих робіт стосуються обробки растрової графіки.

Дослідження методів приховування інформації в векторних зображеннях проведено в роботах [11-16]. У роботі [11] автори досліджують та розвивають запропоновані техніки приховування інформації в ламані лінії. Зокрема, досліджуються питання оптимізації відповідних детекторів цифрових водяних знаків.

У [13] досліджуються цифрові водяні знаки у векторних зображеннях. При цьому приховування відбувається шляхом змін відносних відстаней між координатами окремих точок. Також авторами вивчаються різні метрики для виміру відстаней, зокрема, в роботі застосовано манхеттенські відстані. Такі дослідження продовжено в роботах [14-15], в яких відповідно для приховування інформації також застосовуються відстані між координатами окремих точок та додатково розглядаються технології шифрування векторної графіки.

Роботи [17-18] пропонують методи приховування інформації, використовуючи можливість створення додаткових точок, які певним чином корелюють з основними точками зображення. Наукові праці [19-22] розглядають спектр векторного зображення для вбудовування секретних даних, використовуючи дискретно косинусне перетворення для зміни частотних коефіцієнтів відповідно до обраних констант.

Найбільш ґрунтовною останньою роботою з дослідження технологій приховування інформації в векторні зображення є стаття [23], та відповідна дисертація О. Кінзерявого [24]. У цих роботах досліджено особливості векторної графіки, вивчено та порівняно різні методи приховування інформації. Крім того, запропоновано два методи (побітовий та на основі шаблонів), які при певному удосконаленні можуть бути застосовані до широкого кола математичних об’єктів векторної графіки. Саме ці методи і досліджуються у даній роботі.

1.3 Обґрунтування вибору тематики роботи

Афінні перетворення найбільш імовірно можуть бути використані для спроб руйнування прихованого повідомлення. Також необхідно враховувати можливість набуття зображенням шуму природнім чи зловмисним способами. Так, у загальному випадку афінне перетворення відбувається наступним чином [25-26]:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}. \quad (1.1)$$

При відповідних коефіцієнтах над координатами точок, що піддаються операції, відбувається перетворення, що призводить до зміни положення геометричного об’єкта на полотні. Так, основними афінними перетвореннями є перенесення, поворот, зсув та масштабування (рис. 1.2). При цьому зсув може відбуватися за осями абцис та ординат, а масштабування може бути пропорційним та непропорційним, зі стисненням та розширенням.

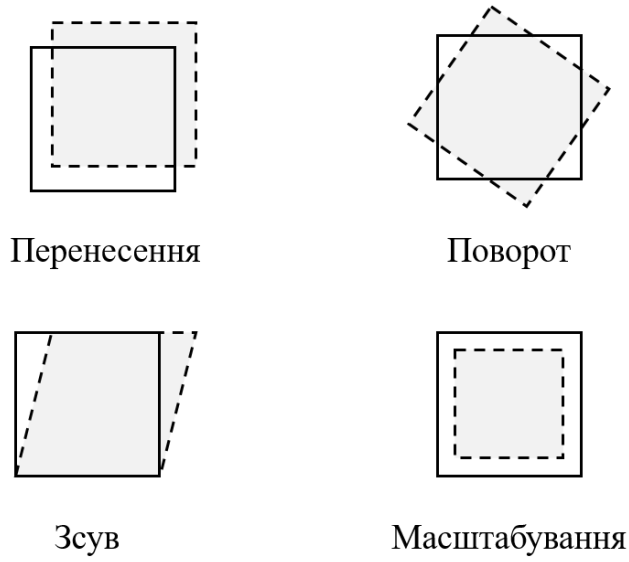

Рисунок 1.2 – Основні види афінних перетворень

Вираз (1.1) для різних випадків може бути записаний наступним чином [27, 28]:

- операція перенесення:

$$\begin{bmatrix} 1 & 0 & c \\ 0 & 1 & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x+c \\ y+f \\ 1 \end{bmatrix}; \tag{1.2}$$

- операція повороту:

$$\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos(\alpha) - y\sin(\alpha) \\ x\sin(\alpha) + y\cos(\alpha) \\ 1 \end{bmatrix}; \tag{1.3}$$

- операція зсуву за віссю абцис:

$$\begin{bmatrix} 1 & b & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x+by \\ y \\ 1 \end{bmatrix}; \tag{1.4}$$

- операція зсуву за віссю ординат:

$$\begin{bmatrix} 1 & 0 & 0 \\ d & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ dx+y \\ 1 \end{bmatrix}; \tag{1.5}$$

- операція масштабування:

$$\begin{bmatrix} a & 0 & 0 \\ 0 & e & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax \\ ey \\ 1 \end{bmatrix}; \tag{1.6}$$

- операція майже афінного перетворення з додаванням шуму:

$$\begin{bmatrix} a+n_1 & b+n_2 & 0 \\ d+n_3 & e+n_4 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x+n_5 \\ y+n_6 \\ 1 \end{bmatrix} = \begin{bmatrix} (a+n_1)(x+n_5)+(b+n_2)(y+n_6) \\ (d+n_3)(x+n_5)+(e+n_4)(y+n_6) \\ 1 \end{bmatrix}. \tag{1.7}$$

Приведені у попередньому підрозділі наукові праці з виконання стеганографічних перетворень над векторними зображеннями у своїй більшості не забезпечують стійкості до поширених видів афінних перетворень при їх багаторазовому застосуванні на контейнері, що містить приховану інформацію. Приведені результати досліджень Кінзяревого [23, 24] у свою чергу пропонують

стійкість до афінних перетворень при їх повторному накладанні. Саму тому тематикою роботи є аналіз запропонованих методів та їх дослідження.

1.4 Висновки до першого розділу

Найбільш вдалим для проведення стеганографічних перетворень є формат векторної графіки SVG, який завдяки своїй структурі дозволяє легко маніпулювати об’єктами, з яких складається. Його широка підтримка різними платформами також дозволяє підвищити рівень скритності при проведенні передачі секретних даних шляхом передачі звичайних на перший погляд файлів медіа.

Існуючі наукові публікації з методів приховування інформаційних повідомлень стосуються растрової графіки, а роботи щодо векторної в основному полягають у кодуванні відстаней між геометричними об’єктами, що містяться у зображенні, або створенні додаткових точок. Усі методи вбудовування у векторні зображення мають вразливість до атак афінного перетворення.

Найпоширенішими видами афінних перетворень є операції перенесення, повороту, зсуву та масштабування з можливими варіаціями (зсуву за осями абцис та ординат, масштабування пропорційне та непропорційне, зі стисненням та із розширенням).

Більшість методів вбудовування інформації у векторні зображення забезпечують одноразову стійкість до афінних перетворень, при цьому при повторному накладенні операцій зміни положення об’єктів, повідомлення може зруйнуватися взагалі. Запропоновані О. Кінзяревим способи приховування даних можуть реалізувати більший рівень стійкості до різного роду перетворень при їх багаторазовому проведенні.

2 РОЗРОБКА МЕТОДУ ТА ОБЧИСЛЮВАЛЬНИХ АЛГОРИТМІВ ПРИХОВУВАННЯ ТА ВИЛУЧЕННЯ ІНФОРМАЦІЇ З ВЕКТОРНИХ ЗОБРАЖЕНЬ

У векторній графіці можна виділити наступні математичні основи: точка, пряма лінія, криві першого та другого порядку, криві Без'є, а також вузли, дотичні лінії та керуючі точки [29-31]. Усі основні векторні елементи будуються за сукупністю точок і будуються згідно з аналітичною формулою із відповідними параметрами. Для того, аби створювати плавні криві найчастіше використовується алгоритм інтерполяції Без'є.

2.1 Математичні моделі стеганографічних перетворень із застосуванням векторної графіки

Криві Без'є вважаються особливим випадком поліноміальних плоских кривих з одним параметром та математично задаються наступним чином [29-31, 32]:

$$P_n(t) = \sum_{i=0}^{n} B_n^i(t) P_i,\ t = t + \Delta t,\ \ t \in [0,1], \tag{2.1}$$

де n – ступінь кривої,

$B_n^i(t)$ – поліном Бернштейна,

t – параметр побудови кривої, який змінюється згідно з кроком Δt,

P_i – це опорні точки кривої ($i \in \overline{0,n}$).

Поліном Бернштейна визначає вагу опорної точки, є базисною функцією кривої Без'є та може бути обрахований за наступним виразом:

$$B_n^i(t) = C_n^i (1-t)^{n-i} t^i, \tag{2.2}$$

де $C_n^i = \dfrac{n!}{(n-i)!i!}$ – біноміальний коефіцієнт.

Зважаючи на те, що поліном Бернштейна фактично є деяким коефіцієнтом ваги опорної точки, то форма кривої формально описується наступним виразом:

$$P_i(t) = P_0 \cdot k_0 + P_1 \cdot k_1 + P_2 \cdot k_2 + P_3 \cdot k_3, \tag{2.3}$$

де P_0, P_1, P_2, P_3 – опорні точки кубічної кривої Без'є,

k_0, k_1, k_2, k_3 – скалярні коефіцієнти, які зважують вклад опорної точки.

Знаючи, що коефіцієнти – це поліноми Бернштейна, які визначаються згідно з виразом (2.2) і залежать від значення параметру побудови кривої, вони можуть бути записані у вигляді (2.4):

$$\begin{aligned} k_0(t) &= \binom{3}{0} \cdot (1-t)^3 \cdot t^0 = (1-t)^3 \\ k_1(t) &= \binom{3}{1} \cdot (1-t)^2 \cdot t^1 = 3(1-t)^3 t \\ k_2(t) &= \binom{3}{2} \cdot (1-t)^1 \cdot t^2 = 3(1-t)t^2 \\ k_3(t) &= \binom{3}{3} \cdot (1-t)^0 \cdot t^3 = t^3 \end{aligned} \tag{2.4}$$

Таким чином, визначення опорних точок для кривої першого, другого та третього ступенів буде описуватися відповідно виразами 2.5-2.7:

$$P = (1-t) \cdot P_0 + t \cdot P_1 \tag{2.5}$$

$$P = (1-t)^2 \cdot P_0 + 2 \cdot (1-t) \cdot t \cdot P_1 + t^2 \cdot P_2 \tag{2.6}$$

$$P = (1-t)^3 \cdot P_0 + 3 \cdot (1-t)^2 \cdot t \cdot P_1 + 3 \cdot (1-t) \cdot t^2 \cdot P_2 + t^3 \cdot P_3 \tag{2.7}$$

Очевидно, що вищезазначені формули визначення форми кривої Без'є за опорними точками являють собою досить трудомісткий процес, особливо у випадку, коли крок Δt зміни параметру побудови траєкторії досить невеликий. Існує більш стабільний алгоритм обрахунку позиції точки на кривій для будь якого параметру t, на відміну від методу прямого використання параметричної форми кривої.

Так, принцип алгоритму де Кастельжо полягає у рекурсивному обчисленні проміжної позиції кожного із сегментів, утворених шляхом умовного з'єднання сусідніх точок, з використанням лінійної інтерполяції контрольних точок для заданого параметру t [33].

Нехай задано 4 контрольні точки P_0, P_1, P_2, P_3, які послідовно з'єднані умовними лініями:

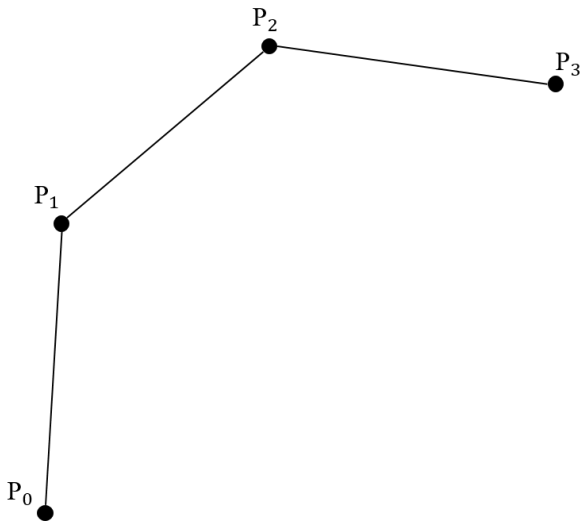

Рисунок 2.1 – Задані опорні точки кривої Без'є третього ступеня

Тоді для визначення траєкторії, яка задана контрольними точками, необхідно, аби параметр побудови t поступово (з кроком Δt) набув значення від 0 до 1. При цьому постійно обчислюються точки на попередньо умовно створених сегментах, позиція яких залежить від t.

Припустимо, що $\Delta t = 0.2$, а $t_1 = t_0 + \Delta t = 0.2$. Звідси на відрізках P_0P_1, P_1P_2, P_2P_3 відкладаються точки, які знаходиться на відстані, пропорційній значенню t (при $t = 0.2$ відстань становить 20% відрізка), та з'єднуються між собою (відрізки $P_0^1P_0^2, P_0^2P_0^3$ на рис. 2.2). На новоутворених відрізках знову розташовують точки на відстані t, які утворюють відрізок $P_1^1P_1^2$. На позиції, відкладеній на відстань 20%, на останньому відрізку і буде знаходитись точка кривої P_2^1.

На рис. 2.3 у графічному вигляді зображено знаходження точки при параметрі побудови $t = 0.8$.

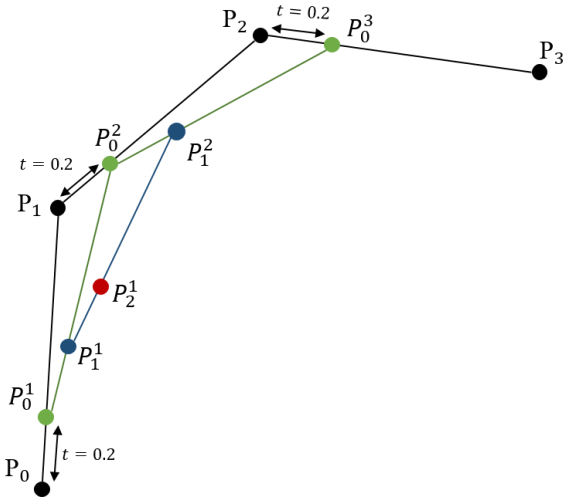

Рисунок 2.2 – Визначення точки кривої при параметрі побудови $t = 0.2$

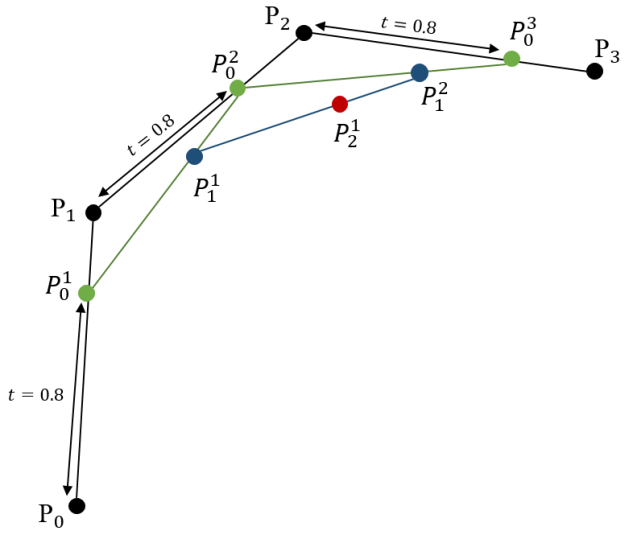

Рисунок 2.3 – Визначення точки кривої при параметрі побудови $t = 0.8$

При проходженні всіх значень параметра побудови t із заданим кроком, відтворюється крива:

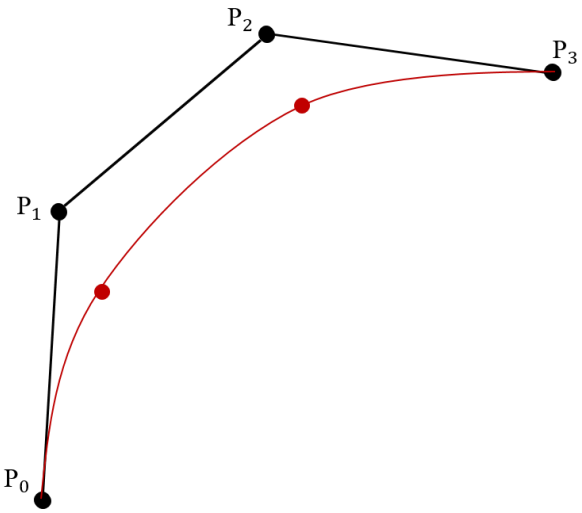

Рисунок 2.4 – Побудована кубічна крива за опорними точками P_0, P_1, P_2, P_3

У кривих Без'є перша та остання точки співпадають з першою та останньою контрольними точками, а лінії P_0P_1 та P_2P_3 є дотичними до них. Така властивість говорить про те, що перехід між двома кривими буде непомітним, якщо дотична P_2P_3 першої кривої буде колінеарна до дотичної першої опорної точки другої (перехід до якої виконується). Ця властивість обумовлює існування складених

кубічних кривих Без'є. Така крива може бути неперервною і складатись із n елементарних кривих, в яких остання контрольна точка буде першою для наступного сегмента:

$$c = c_1 \cup c_2 \cup ... \cup c_n \tag{2.8}$$

Завдяки такій властивості неперервності та параметричній залежності кривих Без'є припустиме використання параметру побудови t для приховування двійкових даних шляхом розбиття сегменту у відповідних позиціях на два сегменти. Здійснення переходу від однієї кривої з n опорними точками до сукупності сегментів можливе через рекурсивний алгоритм де Кастельжо. Розбиття кубічної кривої можна схематично відобразити наступним чином:

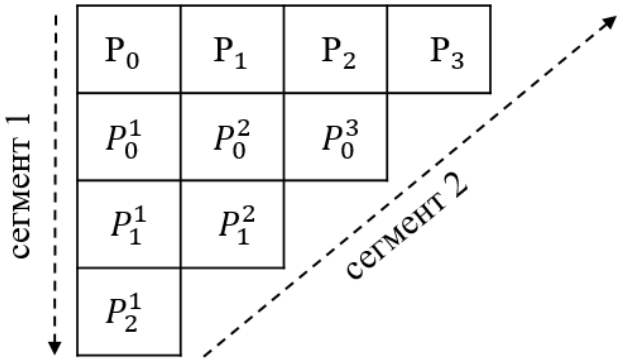

Рисунок 2.5 – Схема розбиття кубічної кривої за алгоритмом де Кастельжо

Відповідні проміжні точки розраховуються за формулами:

P_0	P_1	P_2	P_3
$P_0^1 = P_0 + (P_1 - P_0)t$	$P_0^2 = P_1 + (P_2 - P_1)t$	$P_0^3 = P_2 + (P_3 - P_2)t$	
$P_1^1 = P_0^1 + (P_0^2 - P_0^1)t$	$P_1^2 = P_0^2 + (P_0^3 - P_0^2)t$		
$P_2^1 = P_1^1 + (P_1^2 - P_1^1)t$			

Рисунок 2.6 – Формули розрахунку точок за алгоритмом де Кастельжо

Або у вигляді виразів (2.9) для точок першого рівня та (2.10) – для інших:

$$P_0^j = P_{i-1} + (P_{i+1} - P_{i-1})t \tag{2.9}$$

$$P_i^j = P_{i-1}^j + (P_{i-1}^{j+1} - P_{i-1}^j)t, i = \overline{1, n-1} \tag{2.10}$$

Схема на рис. 2.5 називають матрицею де Кастельжо і вона наглядно демонструє, які точки служитимуть опорними для сегментів, що утворюються. Так, результатом розбиття кубічної кривої Без'є з контрольними точками P_0, P_1, P_2, P_3 у точці t будуть два сегменти з опорними точками відповідно P_0, P_0^1, P_1^1, P_2^1 та P_2^1, P_1^2, P_0^3, P_3. Наприклад, приведена на рис. 2.4 крива у випадку розбиття в точці $t = 0.8$ матиме вигляд як на рис. 2.7 і являтиме собою складену неперервну криву Без'є.

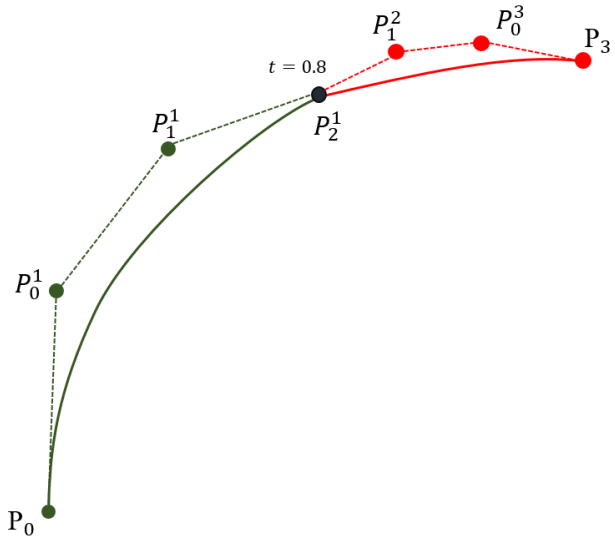

Рисунок 2.7 – Розбиття кубічної кривої на два сегменти в точці $t = 0.8$

Окрім розділення кривих Без'є алгоритм Кастельжо дозволяє провести обернену операцію – відтворення сукупності сегментів у вигляді єдиної кривої. Ґрунтуючись на даній можливості, можна виконувати стеганографічні перетворення заданих кривих на сукупність сегментів визначеним чином (за допомогою параметра побудови кривої) для вбудовування та зворотне перетворення сегментів у криву для вилучення секретних даних.

2.2 Обчислювальний алгоритм приховування даних у векторних зображеннях

Згідно з математичною моделлю стеганографічного перетворення очевидною стає можливість реалізації алгоритму вбудовування секретних даних у векторні зображення, які містять криві Без'є. Оскільки за допомогою алгоритму Кастельжо реалізовується розбиття кривої на сегменти, які напряму залежать від параметру побудови t, можна встановити, що при вхідному біту (наприклад при «1»)

здійснюватиметься поділ кривої на два сегменти в точці t_i, а при «0» – поділ не відбуватиметься [34-35]. Оскільки стеганографічне перетворення відбувається в залежності від значення одного біта, то такий спосіб приховування має назву побітового методу.

Так, на першому етапі необхідно представити двійкове повідомлення m у вигляді блоків з m_l біт. Оптимальним приймається розмір блоку повідомлення у $m_l = 8$ біт.

Другий етап знаменує процес приховування кожного блоку повідомлення у криву. Для цього визначається крок зміни параметру побудови Δt, який має бути менший за величину $\frac{1}{m_l}$ задля виконання більш рівномірного поділу кривої. Також необхідно обрати довільне початкове значення t_0, яке задовольняє умові $0 < t_0 < 1 - m_l \cdot \Delta t$.

На вхід подається крива C^i та відповідно блок двійкових даних m^i і починається поступове збільшення параметра побудови кривої $t_i = t_{i-1} + \Delta t$:

- аналізується вхідне значення біта m_j^i: якщо від дорівнює двійковій одиниці відбувається розбиття кривої в точці t_i та утворення двох сегментів $C^i = c_k^i \cup c_{k+1}^i$, а наступне вбудовування виконуватиметься у сегмент c_{k+1}^i;
- аналізується вхідне значення біта m_j^i: якщо від дорівнює двійковому нулю розбиття кривої в точці t_i не відбувається.

Процес вбудовування блоку m^i в C^i завершується заміною початкової кривої сукупністю сегментів $C^i = c_1^i \cup c_2^i \cup ... \cup c_{n \le m_l + 1}^i$.

Так, наприклад, якщо векторне зображення містить одну криву, в яку необхідно вбудувати восьми бітне повідомлення $m = 11001011$, то після розбиття

крива буде представлена шістьома сегментами (рис. 2.8) відповідно з сьома вузлами (при прихованні одиниці та перша й остання опорні точки).

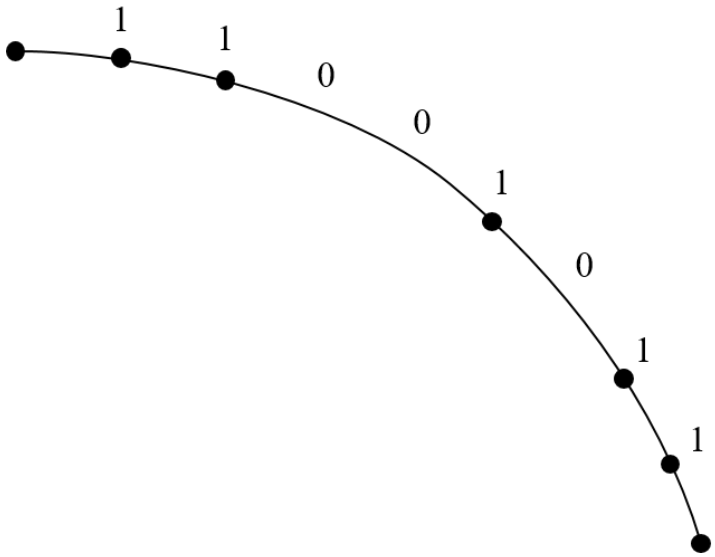

Рисунок 2.8 – Приклад розбиття кривої побітовим методом

Так як сегменти розбитої кривої являють собою повноцінні криві того ж порядку, то вихідна неперервна крива містить значну кількість точок, яка може бути визначена наступним чином:

$$Q = q_{вуз} + q_{сег} \cdot q_{кер}, \qquad (2.11)$$

де $q_{вуз}$ – кількість вузлів,

$q_{сег}$ – кількість сегментів,

$q_{кер}$ – кількість керуючих точок.

У граничному випадку вбудовування повідомлення з восьми одиниць одну складену криву необхідно буде представити згідно з (2.11) $Q = 10 + 9 \cdot 2 = 28$ точками. У цей час крива, представлена на рис. 2.8, описується $Q = 7 + 6 \cdot 2 = 19$ точками.

Для досягнення меншого об’єму даних, потрібних для описання кривої з вбудованим секретним повідомленням, необхідно менше раз виконувати розбиття кривої. Це можна забезпечити у випадку, коли блоки однакових двійкових даних будуть зіставлені з певним кроком зміни параметра побудови t. Таким чином деякі

бітові паттерни з визначеною довжиною вбудовуватимуться в криву декілька раз замість приховання кожного біта окремо [36-37].

Для такого методу паттернів необхідно зберігати таблицю зіставлення блоків бінарних даних та відповідних значень Δt. Приклад набору паттернів та їх кроків зміни представлений у таблиці 2.1. Так, при довжині паттерну $p = 4$ кількість значень блоків визначатиметься наступним чином

$$w = 2^p \tag{2.12}$$

і дорівнюватиме $w = 2^4 = 16$. Максимальне значення Δt не має перевищувати відношення довжини паттерну та максимальної кількості приховуваних у криву бітів, а також бути унікальним серед переліку кроків для різних паттернів.

Таблиця 2.1 – Приклад таблиці паттернів

w	Паттерн	Крок зміни Δt_w
0	0000	0.0095
1	0001	0.009
2	0010	0.0085
3	0011	0.008
4	0100	0.0075
5	0101	0.007
6	0110	0.0065
7	0111	0.006
8	1000	0.0055
9	1001	0.005
10	1010	0.0045
11	1011	0.004
12	1100	0.0035
13	1101	0.003
14	1110	0.0025
15	1111	0.002

Для вбудовування методом паттерну на першому етапі необхідно представити двійкове повідомлення m у вигляді блоків з m_l біт. Оптимальним приймається розмір блоку повідомлення у $m_l = 8$ біт.

На другому етапі відбувається приховування кожного блоку в криву. На вхід подається крива C^i, послідовність двійкових даних m^i, яка ділиться на блоки m_j^i довжиною p, та починається зміна параметра $t_i = t_{i-1} + \Delta t_w$ (початкове значення t_0 – довільне та менше максимального значення Δt). Ґрунтуючись на значенні m_j^i, з таблиці паттернів вилучається відповідне значення Δt_w та у точці $t_i = t_{i-1} + \Delta t_w$ відбувається розбиття кривої $C^i = c_k^i \cup c_{k+1}^i$. Наступний блок буде приховуватися у сегмент c_{k+1}^i та його крок зміни Δt_w додаватиметься до попереднього значення t.

Процес вбудовування блоку m^i в C^i завершується заміною початкової кривої сукупністю сегментів $C^i = c_1^i \cup c_2^i \cup ... \cup c_n^i$, де n – кількість блоків, які вбудовуються в криву. Після завершення приховування секретних даних в усі доступні у векторному зображенні криві, необхідно зберігати таблицю паттернів, а також значення останнього параметру t. Ці дані служать стеганоключем, без якого вилучення інформації неможливе.

Так, якщо попереднє повідомлення $m = 11001011$ вбудовувати методом паттернів, то після розбиття крива буде представлена трьома сегментами (рис. 2.9) та чотирма вузлами.

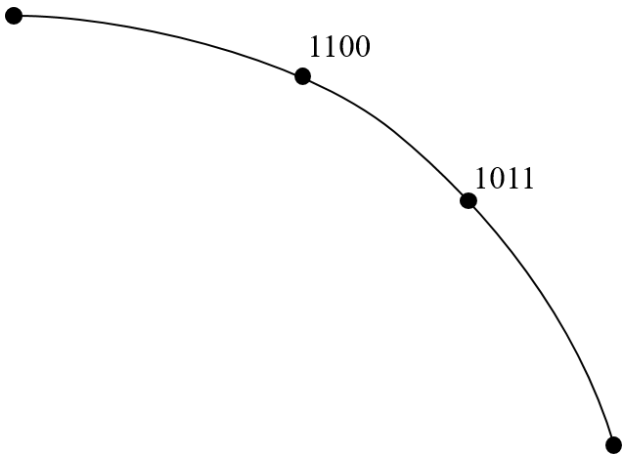

Рисунок 2.9 – Приклад розбиття кривої методом паттернів

Тоді, згідно з виразом (2.11) крива на рис. 2.9 описуватиметься $Q = 4 + 3 \cdot 2 = 10$ точками на відміну від кривої, в яку було виконано побітове вбудовування з тим самим повідомленням та отриманням на виході 19 точок.

При втіленні методів необхідно притримуватись точності обрахунку точок для побудови сегментів для уникнення надто довгих та помітних дробових частин у точок, а також для можливості реалізації вилучення даних.

2.3 Обчислювальний алгоритм вилучення даних з векторних зображень

Вилучення даних із стеганоконтейнерів відбувається шляхом відтворення початкової кривої із сукупності сегментів за заданим значенням параметра побудови t [34]. У випадку з побітовим методом на вхід подається крива, яка складається із сегментів $C^i = c_1^i \cup c_2^i \cup ... \cup c_{n \le m_l + 1}^i$. Об'єднання сегментів починається з кінця, значення параметру побудови або передається захищеним каналом, або визначається програмно:

$$t_{вил} = \Delta t \cdot \left(q_{біт} + 1 \right), \tag{2.13}$$

де $q_{біт}$ – максимальна кількість бітів, що вбудовуються в одну криву.

Послідовно визначається значення параметра $t_i = t_{i+1} - \Delta t$, яке використовуються для знаходження деякої контрольної точки P_d' за опорними точками останніх двох сегментів. Точка P_d' необхідна для обчислення додаткових точок P_{d1}' та P_{d2}', які беруть участь у вилучення біта інформації шляхом виконання або невиконання умови

$$\left| P_{d1}' - P_{d2}' \right| \le A, \tag{2.14}$$

де A – максимально допустима похибка, значення якої залежить від визначеної точності координат точок при вбудовуванні даних.

Так, за координатами опорних точок отриманих на вході двох сегментів кривої $\left\{ P_0, \mathrm{P}_0^1, \mathrm{P}_1^1, \mathrm{P}_2^1 \right\} \left\{ \mathrm{P}_2^1, \mathrm{P}_1^2, \mathrm{P}_0^3, P_3 \right\}$ визначаються точки:

$$P_1^t = \frac{\left(P_2^1 - tP_1^2\right)}{\left(1-t\right)}, P_2^t = \frac{\left(P_1^2 - tP_0^3\right)}{\left(1-t\right)}, P_3^t = \left(1-t\right)P_0^1 + tP_2^t \quad (2.14.1)$$

А умова вилучення за їх допомогою виглядає наступним чином:

$$\left|P_1^t - P_3^t\right| \le A, \left|P_1^1 - P_1^t\right| \le A, \left|P_1^1 - P_3^t\right| \le A, \quad (2.14.2)$$

Шукані точки геометрично виглядають наступним чином:

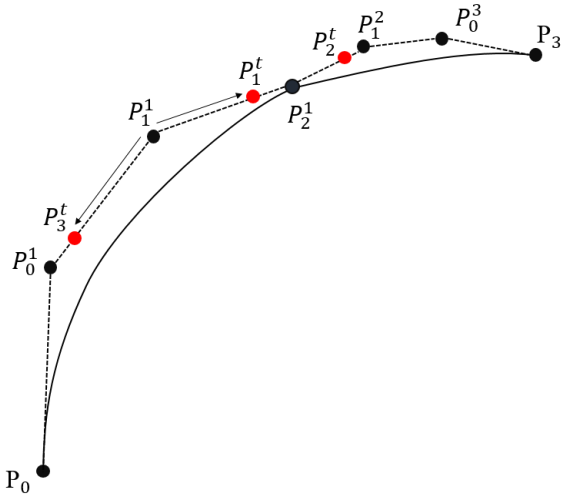

Рисунок 2.10 – Геометричне розташування додаткових точок

У випадку дотримання умови (2.14) виконується об’єднання останніх двох сегментів та вилучення двійкової одиниці (або нуля, якщо він був вибраний в якості критерію поділу кривих). Якщо ж (2.14) не виконується – вилучається двійковий нуль, а сегменти не об’єднуються.

При вилучення даних із стеганоконтейнера, отриманого після операції приховування методом паттернів, необхідно володіти вхідними даними: стеганоключем із таблицею співвідношення бінарних блоків та їх кроками зміни параметра побудови кривої, а також сукупністю фінальних параметрів $t_{фін}$ для кожної кривої [37].

На першому етапі відбувається об’єднання останніх двох сегментів відповідно до значення $t_{фін}$ в один сегмент. Далі необхідно виконати перебір кроків Δt_w з метою знаходження такого $t_i = t_{i+1} - \Delta t_w$, аби обрахування точок P_d', P_{d1}' та P_{d2}' виконувалась умова (2.11). Коли таке значення t_i буде знайдене, останні два

сегменти об’єднаються в одну криву. Для вилучення останнього блоку виконується перебір Δt_w до тих пір, поки $t_{i+1} - \Delta t_w = t_0$.

2.4 Висновки до другого розділу

Математична модель кривих Без’є дозволяє виконувати над ними стеганографічні перетворення вбудовування та вилучення даних. Рекурсивний алгоритм де Кастельжо забезпечує можливість здійснення операцій розбиття кривої на сукупність сегментів відповідно до значення параметру побудови кривої. Параметрична залежність може бути використана задля реалізації функцій вилучення секретної інформації, базуючись на розрахунку додаткових точок та значення похибки.

Вбудовування секретних даних можливо реалізувати двома шляхами:

- розбиттям кривої при кожному біті з інформаційної послідовності (в залежності від того, який обрано – «0» чи «1») відповідно до поточного значення параметра побудови;
- розбиттям кривої при кожному двійковому блоку відповідно до співвіднесеного з ним кроку зміни параметра побудови у таблиці паттернів.

Вилучення прихованого повідомлення відбувається в наслідок відтворення початкової кривої, поступово об’єднуючи сегменти та перевіряючи, чи задовольняють певній умові додатково обраховані точки. У випадку методу паттернів виконується додаткова операція пошуку кроку зміни параметра побудови в таблиці паттернів при вилученні кожного блоку даних.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ СТЕГАНОГРАФІЧНИХ ПЕРЕТВОРЕНЬ ІЗ ЗАСТОСУВАННЯМ ВЕКТОРНИХ ЗОБРАЖЕНЬ

3.1 Обґрунтування вибору мови програмування та середовища розробки

В останній час все більшої популярності набувають застосунки, які функціонують в браузерах, через їхню широку підтримку різними пристроями (від персональних комп'ютерів до мобільних телефонів). Відомо, що ні один веб-застосунок не буде ефективним без належної обробки даних та забезпечення інтерактивної взаємодії з користувачем. Так, однією із засадничих мов програмування є JavaScript, яка на сьогоднішній день займає передові місця із використання (рис. 3.1.1) [38].

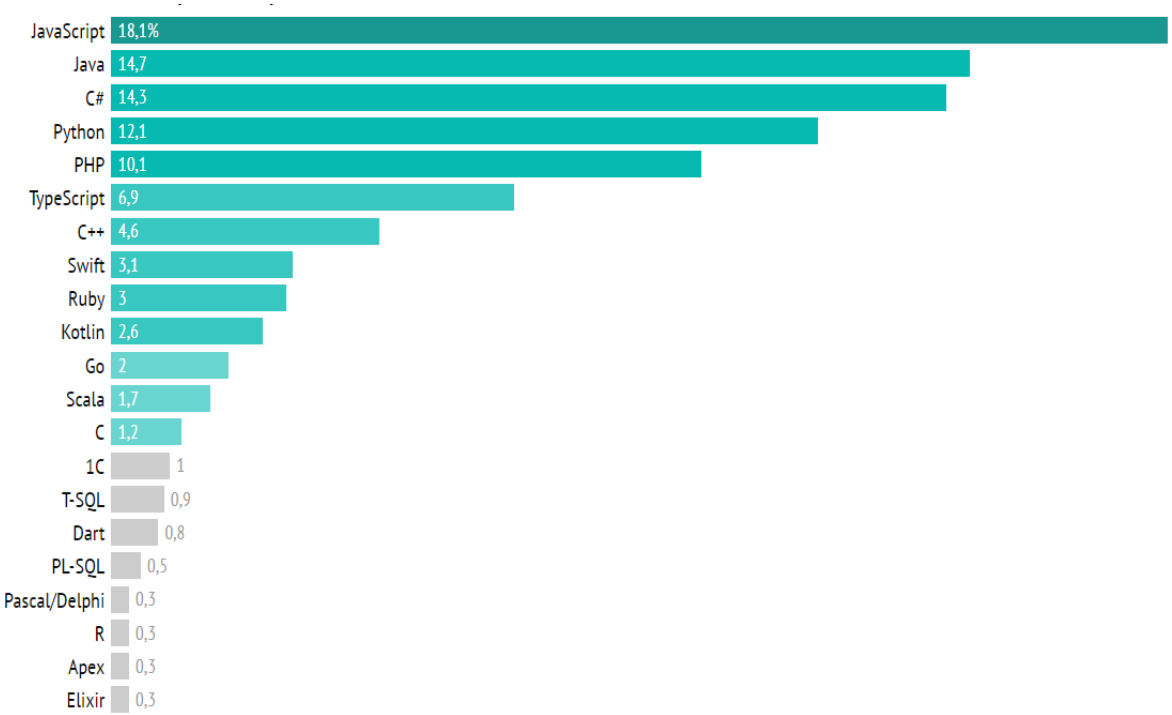

Рисунок 3.1 – Статистика використання різних мов програмування

Хоча оригінально JavaScript планувалася мовою веб-клієнтських застосунків, на даний момент вона перетворилася у потужний інструмент для написання бекенду складних та професіональних задач [39]. При цьому мова є незалежною від платформи, тобто скрипти можуть бути запущені на всіх типах платформ та

браузерів. Конкуренцію JS у бекенді складають головним чином Ruby (on Rails), .NET (ASP), PHP (Laravel), Python (Django) і Java.

Популярність JavaScript обумовлена тим фактом, що одноразове написання коду приводить до отримання стабільно працюючого застосунку, сумісного з великою кількістю популярних мобільних платформ (iOS, Android) та беззаперечною підтримкою десктопних браузерів. Останній стандарт JavaScript ECMAScript 2015 (ES6) підтримується наступним переліком браузерів [40]:

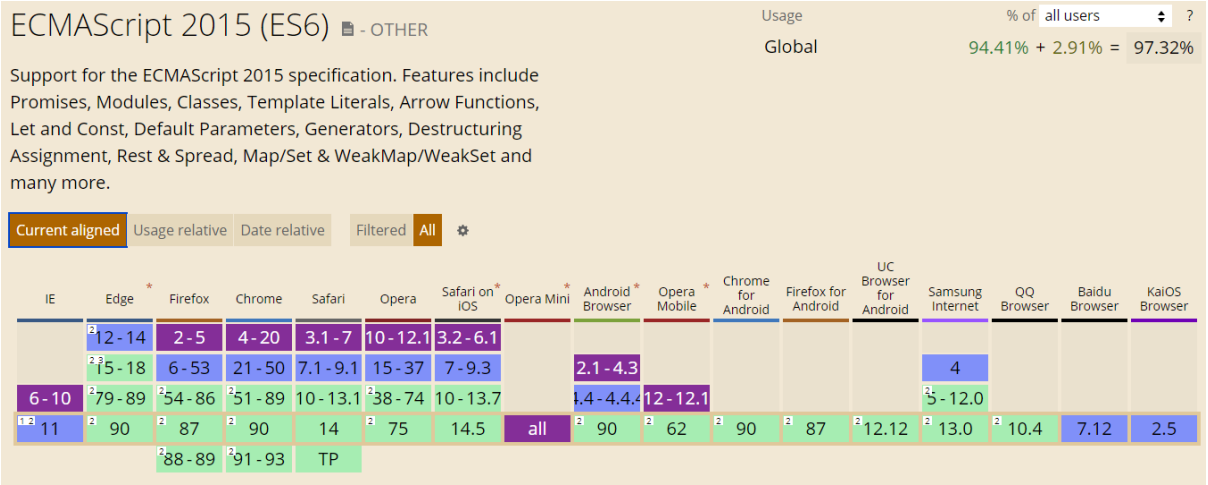

Рисунок 3.2 – Підтримка JavaScript ES6 браузерами

Будь-який застосунок, написаний на нативному JavaScript може бути у майбутньому доповнений функціоналом таких фреймворків як React, Angular, Node, Vue та інші. Тобто завжди існує можливість покращення роботи програми, особливо враховуючи зростаючу поширеність застосунків на мовах екосистеми JavaScript у всіх галузях професійної діяльності.

Окрім популярності та легкості запуску скриптів створеної програми вибір мови програмування обумовлений і специфікою теми, що досліджується. Так, векторні зображення у форматі SVG можуть бути легко представлені об’єктами DOM структури веб-сторінки. Як відомо, SVG є мовою розмітки, тобто доступ до її окремих елементів просто досягається інструментами JavaScript. Також можливим є перегляд контейнеру векторних елементів у режимі роботи зі сторінкою через консоль розробника.

Вибір середовища розробки при програмування на JavaScript досить широкий. Обране IDE повинно забезпечувати комфортний процес розробки та надавати можливість роботи з допоміжними системами та інтуїтивну навігацію у структурі проекту. Серед найбільш популярних виділяються такі редактори як Visual Studio, WebStorm, Atom, Brackets, Comodo Edit, Sublime Text та інші [41].

Обраним середовищем розробки є повнофункціональне IDE WebStorm від JetBrains зі студентською ліцензією. Серед переліку якісних редакторів WebStorm виокремлюється широким функціоналом, попередньо встановленим пакетом додаткових плагінів, підтримкою великої кількості форматів файлів. Внутрішня система контролю версій не дозволяє в аварійних ситуаціях втратити написані скрипти. У рамках теми, що досліджується, WebStorm забезпечує зручний паралельний перегляд SVG-файлів (код розмітки та векторне зображення, що ним описується).

3.2 Розробка програмної реалізації обробки контейнера

Для реалізації операцій приховання та вилучення секретних даних необхідно виконати попередню обробку контейнера. Така обробка необхідна для того, аби представити вхідні дані у єдиному, зручному для виконання перетворень вигляді. Початкові контейнери розроблялися за допомогою графічного редактора Inkscape.

Наприклад, вхідна крива має наступний вигляд:

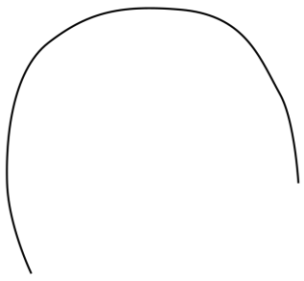

Рисунок 3.3– Приклад вхідної кривої

Крива на рис. 3.3 описується таким атрибутом шляху:

```
d="m 8.6934523,43.845239 c 0,0 -3.0238096,-6.236608 -3.2127975,-11.90625 C 5.2916666,26.269345 6.047619,17.197916 10.961309,13.418155 15.875,9.6383926 20.032738,8.315476 28.537201,9.0714284 c 8.504465,0.7559524 10.352564,6.2891626 13.06867,11.1502976 c 1.945574,3.482079 2.428354,11.717263 2.428354,11.717263"
```

Рисунок 3.4 – Приклад опису кривої

На першому етапі вилучені із зображення криві у строковому вигляді розбиваються за наявністю символу пробілу між ними. Так, початковий опис кривої на рис. 3.4 набуває наступного виду:

```
[ "m", "8.6934523,43.845239", "c", "0,0", "-3.0238096,-6.236608", "-3.2127975,-11.90625", "C", "5.2916666,26.269345", "6.047619,17.197916", "10.961309,13.418155", "15.875,9.6383926", "20.032738,8.315476", "28.537201,9.0714284", "c", "8.504465,0.7559524", "10.352564,6.2891626", "13.06867,11.1502976", "c", "1.945574,3.482079", "2.428354,11.717263", "2.428354,11.717263"]
```

Рисунок 3.5 – Перший етап обробки контейнера

Наступний етап знаменує перетворення строкових даних у вигляд об’єкту кривої. У даній програмній реалізації всі криві перетворюються на об’єкти, які зберігають такі дані як:

- початкова точка кривої та її літерал;
- сегменти кривої, їх літерали та валідність;
- точки, з яких складаються сегменти.

Під валідністю мається на увазі можливість вбудовування у даний сегмент секретних даних, яка залежить від відстані між точками (допустимою приймається відстань, більша за одиницю) та інших виключень, які будуть описані далі. У загальному випадку крива, що перейшла обробку зі строкового виду, має наступний вигляд:

```
Curve {
   Literal: String,
   Start point: Point
   Segments: [segment1, segment2,...,
   segmentN],
}
```

```
segment {
  validness: String,
  literal: String,
  points: [Point, Point, Point, Point]
}
Point { x: number, y: number }
```

Рисунок 3.6 – Загальний вигляд об’єктів кривої та сегмента

Таким чином, крива на рис. 3.3 буде представлена у виді:

```
Curve
        literal: "m"
        segments: Array(4)
                0: segment
                        literal: "c"
                        points: Array(3)
                                0: Point {x: 0, y: 0}
                                1: Point {x: -3.0238096, y: -6.236608}
                                2: Point {x: -3.2127975, y: -11.90625}
                        valid: "none"
                1: segment
                        literal: "C"
                        points: Array(6)
                                0: Point {x: 5.2916666, y: 26.269345}
                                1: Point {x: 6.047619, y: 17.197916}
                                2: Point {x: 10.961309, y: 13.418155}
                                3: Point {x: 15.875, y: 9.6383926}
                                4: Point {x: 20.032738, y: 8.315476}
                                5: Point {x: 28.537201, y: 9.0714284}
                        valid: "none"
                2: segment
                        literal: "c"
                        points: Array(3)
                                0: Point {x: 8.504465, y: 0.7559524}
                                1: Point {x: 10.352564, y: 6.2891626}
                                2: Point {x: 13.06867, y: 11.1502976}
                        valid: "none"
                 3: segment
                        literal: "c"
                        points: Array(3)
                                0: Point {x: 1.945574, y: 3.482079}
                                1: Point {x: 2.428354, y: 11.717263}
                                2: Point {x: 2.428354, y: 11.717263}
                        valid: "none"
        startPoint: Point {x: 8.6934523, y: 43.845239}
```

Рисунок 3.7 – Другий етап обробки контейнера

Третій етап знаменує процес «вилучення» кривих із сегментів. Необхідність цього обумовлюється фактом використання лише кубічних кривих Без’є при описі зображень. Так, більшість досить довгих кривих уже на етапі обробки контейнера є складеними кривими, а їх сегменти можуть складатися з кратної трьом кількості точок. На рис. 3.7 видно, що другий сегмент складається з шести точок, тобто його можна представити як два сегменти:

```
Curve
        literal: "m"
        segments: Array(5)
                0: segment
                        literal: "c"
                        points: Array(3)
                                0: Point {x: 0, y: 0}
                                1: Point {x: -3.0238096, y: -6.236608}
                                2: Point {x: -3.2127975, y: -11.90625}
                        valid: "none"
                1: segment
                        literal: "C"
                        points: Array(3)
                                0: Point {x: 5.2916666, y: 26.269345}
                                1: Point {x: 6.047619, y: 17.197916}
                                2: Point {x: 10.961309, y: 13.418155}
                        valid: "none"
                2: segment
                        literal: "C"
                        points: Array(3)
                                0: Point {x: 15.875, y: 9.6383926}
                                1: Point {x: 20.032738, y: 8.315476}
                                2: Point {x: 28.537201, y: 9.0714284}
                        valid: "none"
                3: segment
                        literal: "c"
                        points: Array(3)
                                0: Point {x: 8.504465, y: 0.7559524}
                                1: Point {x: 10.352564, y: 6.2891626}
                                2: Point {x: 13.06867, y: 11.1502976}
                        valid: "none"
                4: segment
                        literal: "c"
                        points: Array(3)
                                0: Point {x: 1.945574, y: 3.482079}
                                1: Point {x: 2.428354, y: 11.717263}
                                2: Point {x: 2.428354, y: 11.717263}
                        valid: "none"
        startPoint: Point {x: 8.6934523, y: 43.845239}
```

Рисунок 3.8 – Третій етап обробки контейнера

На четвертому етапі об’єкти сегментів кривої представляються у канонічному вигляді, над яким буде зручно виконувати операції стеганографічних перетворень (рис. 3.9). Канонічним вважається форма представлення кривої P_0, P_1, P_2, P_3, де перша точка є останньою точкою попереднього сегмента (або стартовою точкою кривої, якщо сегмент є початковим).

```
Curve
        literal: "m"
        segments: Array(5)
                0: segment
                        literal: "c"
                        points: Array(4)
                                0: Point {x: 8.6934523, y: 43.845239}
                                1: Point {x: 0, y: 0}
                                2: Point {x: -3.0238096, y: -6.236608}
                                3: Point {x: -3.2127975, y: -11.90625}
                        valid: "none"
                1: segment
                        literal: "C"
                        points: Array(4)
                                0: Point {x: -3.2127975, y: -11.90625}
                                1: Point {x: 5.2916666, y: 26.269345}
                                2: Point {x: 6.047619, y: 17.197916}
                                3: Point {x: 10.961309, y: 13.418155}
                        valid: "none"
                2: segment
                        literal: "C"
                        points: Array(4)
                                0: Point {x: 10.961309, y: 13.418155}
                                1: Point {x: 15.875, y: 9.6383926}
                                2: Point {x: 20.032738, y: 8.315476}
                                3: Point {x: 28.537201, y: 9.0714284}
                        valid: "none"
                3: segment
                        literal: "c"
                        points: Array(4)
                                0: Point {x: 28.537201, y: 9.0714284}
                                1: Point {x: 8.504465, y: 0.7559524}
                                2: Point {x: 10.352564, y: 6.2891626}
                                3: Point {x: 13.06867, y: 11.1502976}
                        valid: "none"
                4: segment
                        literal: "c"
                        points: Array(4)
                                0: Point {x: 13.06867, y: 11.1502976}
                                1: Point {x: 1.945574, y: 3.482079}
                                2: Point {x: 2.428354, y: 11.717263}
                                3: Point {x: 2.428354, y: 11.717263}
                        valid: "none"
        startPoint: Point {x: 8.6934523, y: 43.845239}
```

Рисунок 3.9 – Четвертий етап обробки контейнера

При побудовах кривих графічні редактори часто застосовують комбінації відносних та абсолютних координат. Цей факт не дає провести нормального вбудовування даних без помітного візуального спотворення. Тому необхідно перевіряти криві на наявність обох варіантів координат і перетворити їх на один. У даному випадку, на п’ятому етапі обробки проводиться перехід до абсолютних координат (якщо необхідно), а також перевіряється відстань між опорними точками сегментів кривої і встановлюється значення валідності.

Крива, що зображена на рис. 3.3, має у своєму складі й абсолютні, й відносні координати, тому над усіма сегментами відбувається операція переходу до абсолютних координат:

```
Curve
        literal: "m"
        segments: Array(5)
                0: segment
                        literal: "C"
                        points: Array(4)
                                0: Point {x: 8.693452, y: 43.845239}
                                1: Point {x: 8.693452, y: 43.845239}
                                2: Point {x: 5.669643, y: 37.608631}
                                3: Point {x: 5.480655, y: 31.938989}
                        valid: "no"
                1: segment
                        literal: "C"
                        points: Array(4)
                                0: Point {x: 5.480655, y: 31.938989}
                                1: Point {x: 5.2916666, y: 26.269345}
                                2: Point {x: 6.047619, y: 17.197916}
                                3: Point {x: 10.961309, y: 13.418155}
                        valid: "yes"
                2: segment
                        literal: "C"
                        points: Array(4)
                                0: Point {x: 10.961309, y: 13.418155}
                                1: Point {x: 15.875, y: 9.6383926}
                                2: Point {x: 20.032738, y: 8.315476}
                                3: Point {x: 28.537201, y: 9.0714284}
                        valid: "yes"
                3: segment
                        literal: "C"
                        points: Array(4)
                                0: Point {x: 28.537201, y: 9.0714284}
                                1: Point {x: 37.041666, y: 9.827381}
                                2: Point {x: 38.889765, y: 15.360591}
                                3: Point {x: 41.605871, y: 20.221726}
                        valid: "yes"
                4: segment
                        literal: "C"
                        points: Array(4)
                                0: Point {x: 41.605871, y: 20.221726}
                                1: Point {x: 43.551445, y: 23.703805}
                                2: Point {x: 44.034225, y: 31.938989}
                                3: Point {x: 44.034225, y: 31.938989}
                        valid: "no"
        startPoint: Point {x: 8.6934523, y: 43.845239}
```

Рисунок 3.10 – П’ятий етап обробки контейнера

Приведена послідовність обробки контейнера у майбутньому може бути доповнена для формування цілого набору правил фільтрації сегментів кривих, над якими неможливо проводити стеганографічні перетворення, що розглядаються. Очевидно, відбувається виключення із процесів приховування та вилучення інших геометричних об’єктів, що можуть бути наявні в зображенні. Однак такі геометричні можуть знаходиться у складі кривої, виступаючи її проміжним

сегментом, і тоді необхідно доповнювати функції парсингу контейнера на відповідну обробку бажано зі збереженням можливості вбудовування даних у сусідні валідні елементи. Подібним «аномальним» випадком може бути наявність подвійної пари координат після команди moveto (m або M), з якої починається кожний об’єкт, при цьому друга точка є неявною командою lineto.

3.3 Розробка програмної реалізації алгоритмів приховування інформації з векторних зображень

Приховування секретних даних шляхом стеганографічних перетворень кривих Без’є можливе завдяки параметричній природі геометричного елементу. Саме вона дозволяє виконувати розбиття кривої за алгоритмом Кастельжо на сукупність сегментів, які пов’язані між собою спільною контрольною точкою.

Для вдалого вбудовування визначаються такі константи:

- $precision$ – точність координат при обчисленні нових точок;
- $maxBitQuantity$ – допустима кількість приховуваних біт в одну криву.

Вбудовування побітовим методом.

При вбудовуванні інформації за побітовим алгоритмом *encodeBit* встановлюється параметр $deltaStep \le \frac{1}{maxBitQuantity}$, який відповідає за зміну параметра побудови кривої t. Повідомлення m для приховування представляється у двійковому вигляді та розбивається на блоки m^i довжиною у $maxBitQuantity$.

Процес вбудовування відбувається завдяки поділу вхідної кривої C^i на сукупність сегментів $C^i = c_1^i \cup c_2^i \cup ... \cup c_{indexOfSegment}^i$, де $indexOfSegment$ – змінна, яка визначає кількість сегментів на певному етапі вбудовування, та на початку дорівнює $indexOfSegment = 0$. Розбиття відбувається у точці $t_i = t_{i-1} + \Delta t$ при t_0 – довільне позитивне значення $t_0 < 1 - maxBitQuantity \cdot \Delta t$. Оскільки вбудовуються бінарні дані, то виникає дві ситуації:

- приховання $m_j^i = 1$ – відбувається поділ сегменту кривої під номером $indexOfSegment$ у точці t_i, будуючи матрицю де Кастельжо згідно з виразами (2.9) та (2.10) та вибираючи ті точки, які будуть служити опорними для новоутворених сегментів. При цьому змінна $indexOfSegment$ інкрементується (наступне вбудовування виконуватиметься у сегмент з індексом $indexOfSegment + 1$) та виконується перехід до m_{j+1}^i;
- приховання $m_j^i = 0$ – поділ сегменту з номером $indexOfSegment$ не реалізовується, виконується перехід до m_{j+1}^i.

Після завершення процесу приховування блоку m^i секретного повідомлення утворені при розбитті сегменти складають нову криву $C^i = c_1^i \cup c_2^i \cup ... \cup c_{indexOfSegment}^i$.

Дані з якими працює функція приховання відображені у табл. 3.1, а псевдокод програмної реалізації виглядає на рис. 3.11:

Таблиця 3.1 – Дані, з якими працює функція *encodeBit*

Визначені константи:	- крок зміни параметру побудови кривої $deltaStep$; - точність розрахунку координат точок $precision$; - кількість біт для приховання в одну криву $maxBitQuantity$.
Вхідні дані:	- сукупність вилучених із зображення кривих $normalizedCurves$; - двійкове повідомлення для вбудовування $input$.
Вихідні дані:	множина нових кривих, утворених при вбудовуванні $encodedCurves$.

```
1. normalizedCurves = selectCurves()
2. input = chunkString(input,maxBitQuantity)
3. For(i = 0;i < normalizedCurves.length;i++)
3.1. For(j = 0; j < normalizedCurves[i].segments.length;i++)
3.2. encodeBit(normalizedCurves[i].segments[j],input[i],deltaStep,precision)
3.2.1. indexOfSegment = 0,splittedSegments = []
3.2.2. For(t = deltaStep,index = 0;index < input[i].length;index++)
3.2.2.1. if (input[i][index] === '1')
3.2.2.1.1. seg = splitCurve(t,normalizedCurves[i].segments[j],precision)
3.2.2.1.2. splittedSegments.push(seg)
3.2.2.1.3. indexOfSegment++
3.2.2.1.4. t = t + deltaStep
3.2.3. encodedCurves.push(splittedSegments)
4. return encodedCurves
```

Рисунок 3.11 – Псевдокод функції приховання *encodeBit*

У вищенаведеному фрагменті псевдокоду функція *selectCurves*() використовується для вилучення кривих з коду розмітки векторного зображення у SVG та їх приведення до нормального виду, зручного для обробки інструментами даної реалізації. Функція *input = chunkString(input,maxBitQuantity)* використовується для розбиття послідовності двійкових даних на блоки довжиною *maxBitQuantity*. Операція *splitCurve(t,normalizedCurves[i].segments[j], precision)* виконує поділ сегменту *normalizedCurves[i].segments[j]* у точці *t* з розрахунком координат з точністю *precision*, дані, якими оперує функція представлені у табл. 3.2, а псевдокод – на рис. 3.12.

Таблиця 3.2 – Дані, з якими працює функція поділу кривої

Вхідні дані:	- параметр побудови кривої *parameter*; - сегмент для поділу *points*; - точність розрахунку *precision*.
Вихідні дані:	масив із двох сегментів

1. $matrix = initMatrix(points.length);\ segment1 = [], segment2 = []$
2. $For(i = 0; i < matrix.length; i++)$
2.1. $For(j = 0; j < matrix[i].length; i++)$
2.1.1. $matrix[0][j] = points[i]$
2.1.2. $matrix[i][j] = matrix[i-1][j] +$
$+(matrix[i-1][j+1] - matrix[i-1][j]) \times parameter$
2.1.3. $matrix[i][j] = round(matrix[i][j], precision)$
3. $For(i = 0; i < matrix.length; i++)$
3.1. $segment1.push(matrix[i][0])$
3.2. $segment1.push(matrix[i][matrix[i].length - 1])$
4. $return\ [segment1, segment2]$

Рисунок 3.12 – Псевдокод операції поділу кривої

У вищенаведеному фрагменті псевдокоду функція *initMatrix(points.length)* створює пусту матрицю де Кастельжо, яка під час виконання поділу заповнюється згідно з виразами (2.9) та (2.10). Після розрахунку сегментів, утворених при поділі кубічної кривої, матриця має такий вигляд:

(index)	0	1	2	3
0	Point	Point	Point	Point
1	Point	Point	Point	
2	Point	Point		
3	Point			

Рисунок 3.13 – Вигляд матриці де Кастельжо при програмній реалізації

Вбудовування методом паттернів.

При вбудовування секретних даних методом паттернів *encodePattern* визначається початкове значення параметра побудови t_0 та довжина блоків даних, що будуть приховуватися, *patternLength*. Повідомлення m для приховування представляється у двійковому вигляді та розбивається на блоки m^i довжиною у *maxBitQuantity*.

Процес вбудовування відбувається завдяки поділу вхідної кривої C^i на сукупність сегментів $C^i = c_1^i \cup c_2^i \cup ... \cup c_{indexOfSegment}^i$, де *indexOfSegment* – змінна, яка визначає кількість сегментів на певному етапі вбудовування, та на початку дорівнює $indexOfSegment = 0$. Таблиця паттернів приховуваних двійкових даних згідно з (2.12) складається із $2^{patternLength}$ записів відповідності блоку даних до кроку Δt. Розбиття відбувається у точці $t_i = t_{i-1} + \Delta t$ при t_0 – довільне позитивне значення $t_0 < max\Delta t$, де $max\Delta t$ – максимальне значення серед визначених кроків зміни параметра побудови кривої у таблиці паттернів.

Оскільки вбудовуються блоки даних m_j^i, то поділ кривої відбувається у будь-якому випадку, а точка розбиття визначається відповідним значенням послідовності довжини *patternLength* у таблиці паттернів:

- за вхідним значенням m_j^i з таблиці паттернів вилучається значення Δt та обраховується параметр побудови кривої $t_i = t_{i-1} + \Delta t$;
- у точці t_i виконується поділ кривої, кількість сегментів *indexOfSegment* збільшується на один;
- виконується перехід до наступного блока m_{j+1}^i.

Після завершення процесу приховування блоку m^i секретного повідомлення утворені при розбитті сегменти складають нову криву $C^i = c_1^i \cup c_2^i \cup ... \cup c_{indexOfSegment}^i$.

Необхідні дані для роботи методу паттернів відображені у табл. 3.3 а псевдокод програмної реалізації виглядає як на рис. 3.14:

Таблиця 3.3 – Дані, з якими працює функція *encodePattern*

Визначені константи:	- початкове значення параметра побудови кривої *startStep*; - точність розрахунку координат точок *precision*; - кількість біт для приховання в одну криву *maxBitQuantity*; - довжина паттерну *patternLength*.
Вхідні дані:	- сукупність вилучених із зображення кривих *normalizedCurves*; - двійкове повідомлення для вбудовування *input*.
Вихідні дані:	- множина нових кривих, утворених при вбудовуванні *encodedCurves*; - таблиця паттернів *patternTable*; - масив останніх параметрів параметра побудови *finalParameters*.

```
1. normalizedCurves = selectCurves()
2. input = chunkString(input, maxBitQuantity)
3. For(i = 0; i < normalizedCurves.length; i++)
3.1. For(j = 0; j < normalizedCurves[i].segments.length; i++)
3.2. encodePattern(normalizedCurves[i].segments[j], input[i],
     precision, startStep, patternLength, maxBitQuantity)
3.2.1. indexOfSegment = 0, splittedSegments = [], finalParameters = []
3.2.2. patternTable = initTableOfPatterns(patternLength, maxBitQuantity)
3.2.3. inputBlock = chunkString(inputBlock, patternLength)
3.2.3.1 For(t = startStep, index = 0; index < inputBlock.length; index++)
3.2.3.1.1 if (patternTable.find(inputBlock[index]))
3.2.3.1.1.1. curStep = patternTable.find(inputBlock[index])
3.2.3.1.1.2. seg = splitCurve(t + curStep,
                 normalizedCurves[i].segments[j], precision)
3.2.3.1.1.3. splittedSegments.push(seg)
3.2.3.1.1.4. indexOfSegment++
3.2.3.1.1.5. t = t + curStep
3.2.3.1.2. finalParameters.push(t)
3.2.3.2. encodedCurves.push(splittedSegments)
4. return {encodedCurves, patternTable, finalParameters}
```

Рисунок 3.14 – Псевдокод функції приховання *encodePattern*

У фрагменті псевдокоду алгоритму *encodePattern* змінна *inputBlock* ініціалізує розбиття вхідної послідовності m^i на блоки довжиною *patternLength*. Функція *initTableOfPatterns*() створює таблицю паттернів, з якою виконується подальше вбудовування блоків даних. При програмній реалізації таблиця паттернів має вид як на рис. 3.15.

index	pattern	step
1	"0000"	0.0095
2	"0001"	0.009
3	"0010"	0.0085
4	"0011"	0.008
5	"0100"	0.0075
6	"0101"	0.007
7	"0110"	0.0065
8	"0111"	0.006
9	"1000"	0.0055
10	"1001"	0.005
11	"1010"	0.0045
12	"1011"	0.004
13	"1100"	0.0035
14	"1101"	0.003
15	"1110"	0.0025
16	"1111"	0.002

Рисунок 3.15 – Вигляд таблиці паттернів при програмній реалізації

Оскільки для вилучення даних необхідно мати ключ, то після вбудовування даних окрім нових кривих зберігаються фінальні параметри побудови кривих та таблиця паттернів. Такий стеганоключ перетворюється у об’єкт JSON, серіалізується та записується у файл. У загальному випадку структура JSON-файлу має наступний вигляд:


```
"patternTable":[{"index":1,"pattern":"0000","step":0.0095},{"index":2,"pattern":"0001","step":0.009},{"index":3,"pattern":"0010","step":0.0085},{"index":4,"pattern":"0011","step":0.008},{"index":5,"pattern":"0100","step":0.0075},{"index":6,"pattern":"0101","step":0.007},{"index":7,"pattern":"0110","step":0.0065},{"index":8,"pattern":"0111","step":0.006},{"index":9,"pattern":"1000","step":0.0055},{"index":10,"pattern":"1001","step":0.005},{"index":11,"pattern":"1010","step":0.0045},{"index":12,"pattern":"1011","step":0.004},{"index":13,"pattern":"1100","step":0.0035},{"index":14,"pattern":"1101","step":0.003},{"index":15,"pattern":"1110","step":0.0025},{"index":16,"pattern":"1111","step":0.002}],
"finalParameters":[[0.01,0.015,0.0185,0.0095,0.014]]}
```


Рисунок 3.16 – Збережений у файл стеганоключ

3.4 Розробка програмної реалізації алгоритмів вилучення інформації з векторних зображень

Для вилучення секретної інформації із векторного зображення необхідно здійснити об’єднання сегментів складеної кривої, поступово отримуючи приховані дані та відтворюючи початкову криву. За контрольними точками двох останніх

обраховуються додаткові точки, порівняння яких дозволяє виконати вилучення відповідних даних. Порівняння здійснюється завдяки визначеній константі $accuracy$, яка повинна задовольняти умові $accuracy \le 5 \cdot 10^{1-precision}$, де $precision$ – обрана при вбудовуванні точність розрахунку координат.

Вилучення побітовим методом.

При вилученні побітовим методом на вхід подаються криві виду $C^i = c_1^i \cup c_2^i \cup ... \cup c_{indexOfSegment}^i$, де $indexOfSegment$ – кількість сегментів, з якої складається неперервна крива. Секретним ключом є значення параметра побудови кривої, яке було застосовано востаннє при приховуванні даних та передається захищеним каналом зв'язку, або визначено програмно: $steganoKey = (maxBitQuantity + 1) \cdot parameterStep$, де $parameterStep$ – це крок зміни параметра t, який використовувався при вбудовуванні інформації.

Вилучення виконується за останніми двома сегментами $c_{indexOfSegment-1}^i \cup c_{indexOfSegment}^i$ при значенні параметра $t_i = t_{i-1} - parameterStep$: згідно з формулами (2.14.1) обраховуються додаткові точки за координатами опорних точок $c_{indexOfSegment-1}^i, c_{indexOfSegment}^i$ та виконується перевірка умови (2.14.2). У залежності від виконання порівняння відбувається:

- додаткові точки $|P'_{d1} - P'_{d2}| \le accuracy$: сегменти $c_{indexOfSegment-1}^i$, $c_{indexOfSegment}^i$ об'єднуються в одну криву при поточному t_i, $indexOfSegment$ зменшується на одиницю і вилучається $m_j^i = 1$;
- додаткові точки $|P'_{d1} - P'_{d2}| \not\le accuracy$: об'єднання не виконується і вилучається $m_j^i = 0$.

Після завершення процесу вилучення отримується початкова крива та вбудоване повідомлення у двійковому представленні. Дані, необхідні для вилучення відображені у табл. 3.4. а псевдокод операції виглядає як на рис. 3.17:

Таблиця 3.4 – Дані, з якими працює функція вилучення *decodeBit*

Вхідні дані:	- сукупність вилучених із зображення кривих *normalizedCurves*; - крок зміни параметра побудови *deltaStep*; - стеганоключ *steganoKey*; - похибка *accuracy*.
Вихідні дані:	вилучене двійкове повідомлення *decodedMessage*

```
1. normalizedCurves = selectCurves()
2. input = chunkString(input, maxBitQuantity)
3. For(i = 0; i < normalizedCurves.length; i++)
3.1. For(j = 0; j < normalizedCurves[i].segments.length; i++)
3.2. decodeBit(maxBitQuantity, accuracy, normalizedCurves[i].segments[j], deltaStep, precision)
3.2.1. For(i = normalizedCurves[i].segments[j].length − 1;
        t = steganoKey − deltaStep; i ≥ 0; t = t − deltaStep)
3.2.1.1. segment1 = normalizedCurves[i].segments[j − 1]
3.2.1.2. segment2 = normalizedCurves[i].segments[j]
3.2.1.3. adP1 = getAdditionalPoint(t, segment2[0], segment2[1])
3.2.1.3. adP2 = getAdditionalPoint(t, segment2[1], segment2[2])
3.2.1.3. adP3 = getCurvePoint(t, segment1[1], adP2)
3.2.1.4. if (isDifferenceSmaller(adP1, adP3, accuracy) ||
            isDifferenceSmaller(segment2[2], adP1, accuracy) ||
            isDifferenceSmaller(segment2[2], adP3, accuracy))
3.2.1.4.1. decodedMessage.push('1')
3.2.1.4.2. mergedSegment = mergeSegments(t, segment1, segment2)
3.2.1.5. else decodedMessage.push('0')
4. return decodedMessage
```

Рисунок 3.17 – Псевдокод функції вилучення *decodeBit*

За перевірку умови (2.14) відповідає функція $isDifferenceSmaller()$. Функція $mergeSegments(t, segment1, segment2)$ реалізує операцію об'єднання двох сегментів $c^i_{indexOfSegment-1} = \left\{P_0, \mathrm{P}^1_0, \mathrm{P}^1_1, \mathrm{P}^1_2\right\}$ і $c^i_{indexOfSegment} = \left\{\mathrm{P}^1_2, \mathrm{P}^2_1, \mathrm{P}^3_0, P_3\right\}$ шляхом обрахунку опорних точок сегменту, над яким проводилось розбиття. Таким чином, при поточному t_i необхідно:

1) Обчислити точку $P_2 = \frac{\left(P^3_0 - t_i P_3\right)}{\left(1 - t_i\right)}$.

2) Обчислити додаткову точку $P_{\text{додат}} = \frac{\left(P^2_1 - t_i P^3_0\right)}{\left(1 - t_i\right)}$.

3) Обчислити точку $P_1 = \frac{\left(P_{\text{дод}} - t_i P_0\right)}{\left(1 - t_i\right)}$.

У результаті отримується початковий сегмент $\left\{P_0, P_1, P_2, P_3\right\}$.

Вилучення методом паттернів.

При вилученні прихованого повідомлення методом паттернів на вхід подаються криві виду $C^i = c^i_1 \cup c^i_2 \cup \ldots \cup c^i_{indexOfSegment}$, де $indexOfSegment$ – кількість сегментів, з якої складається неперервна крива, а також початкове значення параметра побудови $startStep$. Секретним ключом є таблиця паттернів та список фінальних параметрів вбудовування, які передаються надійним каналом зв'язку.

Вилучення виконується за останніми двома сегментами $c^i_{indexOfSegment-1} \cup c^i_{indexOfSegment}$, які на першому етапі об'єднуються при значенні параметра $t_i = finalParameter$, а $indexOfSegment$ зменшується на одиницю. Для сегментів $c^i_{indexOfSegment-1}$ та $c^i_{indexOfSegment}$ виконується:

- визначення наступного значення $t_i = t_{i-1} - \Delta t$ перебором усіх значень зміни параметра Δt у таблиці паттернів;

- згідно з формулами (2.14.1) обраховуються додаткові точки за координатами опорних точок $c^i_{indexOfSegment-1}, c^i_{indexOfSegment}$ та виконується перевірка умови (2.14.2) при поточному t_i;
- якщо умова (2.14.2) виконується, то здійснюється вилучення блоку даних m^i_j, що співвідноситься до кроку зміни параметра t_i, сегменти об'єднуються, а $indexOfSegment$ зменшується на одиницю;
- якщо умова (2.14.2) не виконується, то перебір продовжується, допоки не буде знайдене відповідне значення t_i.

Для вилучення останнього блоку m^i_j здійснюється перебір значень зміни параметра Δt у таблиці паттернів до тих пір, поки $t_{i-1} - \Delta t = startStep$. Дані, необхідні для вилучення методом паттернів відображені у табл. 3.5, а псевдокод операції виглядає як на рис. 3.18:

Таблиця 3.5 – Дані, з якими працює функція вилучення *decodePattern*

Вхідні дані:	- сукупність вилучених із зображення кривих *normalizedCurves*; - початкове значення параметра побудови *startStep*; - стеганоключ $patternSteganoKey = \{patternTable, finalParameters\}$; - похибка *accuracy*.
Вихідні дані:	вилучене двійкове повідомлення *decodedMessage*

```
1. normalizedCurves = selectCurves()
3. For(i = 0;i < normalizedCurves.length;i++)
3.1. For(j = 0; j < normalizedCurves[i].segments.length;i++)
3.2. decodePattern(maxBitQuantity,accuracy,normalizedCurves[i].segments[j],
     startStep, precision, finalParameters[i][j], patternTable)
3.2.1. decodedMessage = [], param = 0
3.2.2. segment = mergeSegments(normalizedCurves[i].segments[j−1],
normalizedCurves[i].segments[j])
3.2.2. For(k = normalizedCurves[i].segments[j].length − 1;t = steganoKey;k ≥ 0;k−−)
3.2.2.1. if (k = 0)  mergeSegments(t,normalizedCurves[i].segments[k−1],
         normalizedCurves[i].segments[k])
3.2.2.1.2.For(l = 0;l < patternTable.length;l++)
3.2.2.1.2.0. param = steganoKey − patternTable[l].step
3.2.2.1.2.1. segment1 = normalizedCurves[i].segments[k−1]
3.2.2.1.2.2. segment2 = normalizedCurves[i].segments[k]
3.2.2.1.2.3. adP1 = getAdditionalPoint( param,segment2[0],segment2[1])
3.2.2.1.2.4. adP2 = getAdditionalPoint( param,segment2[1],segment2[2])
3.2.2.1.2.5. adP3 = getCurvePoint( param,segment1[1],adP2)
3.2.2.1.2.5. if (isDifferenceSmaller (adP1,adP3,accuracy)||
             isDifferenceSmaller (segment2[2],adP1,accuracy)||
             isDifferenceSmaller (segment2[2],adP3,accuracy))
3.2.2.1.2.5.1. decodedMessage.push( patternTable.find ( param ).pattern)
3.2.2.1.2.5.2. mergeSegments(param,normalizedCurves[i].segments[k−1],
               normalizedCurves[i].segments[k])
3.2.2.1.2.5.3. k−−
3.2.2.2. else  For(l = 0;l < patternTable.length;l++)
3.2.2.2.1. param = steganoKey − patternTable[l].step
3.2.2.2.1. if ( param = startStep )
3.2.2.2.1.1.  decodedMessage.push( patternTable.find ( param ).pattern)
4. return  decodedMessage
```

Рисунок 3.18 – Псевдокод функції вилучення *decodePattern*

3.5 Висновки до третього розділу

Мова програмування JavaScript дозволяє легко взаємодіяти та маніпулювати об’єктами, що зберігаються у файлі SVG. Її широка інтеграція з різними браузерами робить створений на ній веб-застосунок крос-платформним та легко застосовуваним для різного рівню комплексних та професійних задач.

Перед початком процесу приховування повідомлення необхідно провести обробку контейнера з метою представлення різнорідних даних у зручному для виконання стеганографічних перетворень вигляді. В оброблений контейнер можливо вбудовувати дані за програмно розробленими алгоритмами розбиття кривих відповідно до поточного значення параметру побудови кривої.

Розроблені операції вилучення даних використовують ту ж саму параметричну залежність: оберненим шляхом об’єднання розбитої кривої при переданому надійним каналом зв’язку параметрі побудови кривої встановлюється, яке значення інформаційних біта або блока вилучати.

4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ РОЗРОБЛЕНИХ АЛГОРИТМІВ, ОБҐРУНТУВАННЯ ПРАКТИЧНОЇ РЕКОМЕНДАЦІЇ ЩОДО ЇХ ВПРОВАДЖЕННЯ

4.1 Дослідження швидкісних характеристик, коефіцієнтів спотворення та збільшення розміру контейнерів

За розробленими алгоритмами було проведено експериментальні дослідження для виявлення середніх значень швидкісних характеристик, оцінки коефіцієнтів спотворення стегоконтейнера даних та збільшення його розміру після вбудовування секретних. Дослідження проводилися при зміні таких параметрів як кількість бітів, що можуть бути приховані в одній кривій, $maxBitQuantity$, точність вбудовування $precision$ та похибка при відтворенні $accuracy$.

Для проведення дослідження було використано контейнери з середнім розміром 6 КБ. У Додатку А наведені результати експериментальних досліджень у табличному графічному виглядах. У табл. А.1-А.2 відображено коефіцієнти збільшення контейнерів при застосуванні обох методів. З таблиць А.1-А.2 видно, що точність вбудовування впливає на розмір контейнера, що міститиме секретні дані. Побітовий метод має вищі показники коефіцієнтів збільшення, адже в цьому випадку виконується більш частий поділ кривих і відповідно з’являється необхідність збереження більшої кількості точок.

На рис. 4.1 приведена узагальнююча діаграма, яка відображає залежність розміру стегоконтейнера від кількості біт, що вбудовуються в одну криву, та параметру точності координат, що розраховуються.

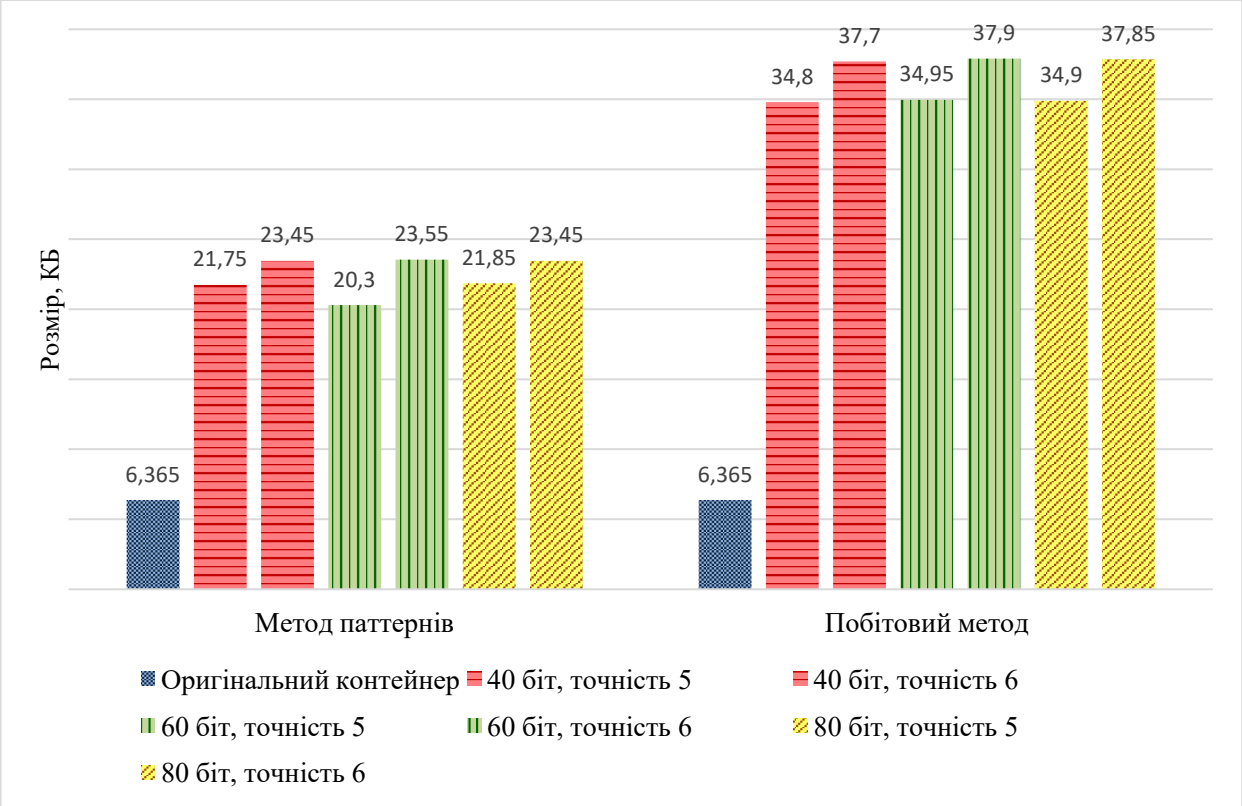

Рисунок 4.1 – Діаграма залежності розміру стегоконтейнера від кількості біт, що вбудовуються в одну криву, та точності розрахунку координат.

Швидкість вбудовування та вилучення двійкових даних визначалась програмно для кожного контейнера. У табл. А.3-А.5 відображено швидкісні характеристики виконання операцій приховання та відтворення даних методом паттернів для 40, 60 та 80 біт, які приходяться на одну криву.

З таблиць А.3-А.5 видно, що приблизний час вбудовування методом паттернів становить 0.35 секунд, а вилучення – 0.33 секунд. Такі досить високі швидкісні показники пояснюються невеликим розміром вхідних контейнерів, а звідси – меншою кількістю операцій щодо розрахунку нових кривих.

У табл. А.6-А.8 відображено швидкісні характеристики виконання операцій приховання та відтворення даних побітовим методом для 40, 60 та 80 біт, які приходяться на одну криву. З таблиць 4.6-4.8 видно, що середній час вбудовування побітовим методом становить 0.5 секунд, а вилучення – 0.1 секунди. Швидкість вбудовування нижча порівняно з методом паттернів через те, що у такому випадку поділ кривої відбувається стільки разів, скільки двійкових одиниць має вхідна

послідовність. У свою чергу, при застосуванні методу паттернів поділ кривої виконується сталою кількістю разів, яка визначається відношенням довжини послідовності, яка вбудовується в одну криву, до довжини блоку паттерну: $\frac{maxBitQuantity}{patternLength}$.

На рис. А.1-А.2 відображено графіки залежності швидкості вбудовування та вилучення даних від кількості біт, що приховуються в одній кривій. У загальному випадку зі збільшенням довжини послідовності, що вбудовується, зменшується час виконання операцій поділу. Швидкість вилучення методом паттернів збільшується зі значенням кількості біт, що приходяться на одну криву, а у випадку з побітовим методом – зменшується.

Необхідно зазначити, що приведені швидкісні характеристики визначають конкретно час операцій вбудовування та вилучення даних без урахування функцій попередньої обробки контейнера та приведення його до нормального виду після приховування інформації.

У табл. А.9 приведено середні коефіцієнти візуального спотворення контейнерів після вбудовування секретних даних. Показники були отримані за допомогою ПЗ AntiDubl.NET-2.3.9, попередньо виконавши конвертування стегоконтейнерів у зображення растрової графіки (формат PNG). Як видно з табл. А.9, зазначене програмне забезпечення не знайшло значних відмінностей стегоконтейнерів від оригінального зображення.

4.2 Дослідження стійкості до афінних перетворень

Дослідження стійкості до афінних перетворень виконувалось програмним чином, використовуючи формули (1.2)-(1.7). Спочатку реалізовується операція вбудовування інформації довжиною 2400 біт у контейнер. Потім над закодованим контейнером відбувається певне афінне перетворення і виконується спроба вилучити повідомлення. Для наступного перетворення використовуватиметься стегоконтейнер, отриманий на попередньому кроці. Таким чином забезпечується

багаторазове накладання операції зміни положення координат на одне й теж ж саме зображення.

Згідно з формулами (1.2)-(1.7) для перетворень використовуються такі коефіцієнти:

- перенесення – $(c, f) \in [-500, 500]$;
- поворот – $\alpha = 1$;
- зсув за віссю абцис – $b = 0,01$;
- зсув за віссю ординат – $d = 0,01$;
- масштабування для стиснення – $a = e = 0,99$;
- масштабування для розширення – $a = e = 1,01$;
- майже афінне перетворення повороту з додаванням шуму – $a = \cos(\alpha)$, $b = -\sin(\alpha)$, $d = \sin(\alpha)$, $e = -\cos(\alpha)$, $\alpha = 1$, $n_1 = n_4 = n_6 = -0,0001$, $n_2 = n_3 = n_5 = 0,0001$.

Накладання перетворень виконувались 10 раз на кожен попередній контейнер. Зміна кроку для побітового методу дорівнює 0.0005, для побітового методу початковий крок дорівнює 0.0005, а довжина паттерну – 4 (таблиця паттернів відображена на рис. 3.15). Результати досліджень у табличному вигляді представлені у Додатку Б, а графіки залежностей – у Додатку В.

Операції приховування та вилучення виконуються з різними наборами значень кількості біт, що вбудовуються в одну криву, точності розрахунку координат та похибки відтворення, які відображені у табл. Б.1.

На рис. В.1 – В.2 відображені графіки залежності відсотка втрачених біт від експерименту при проведенні перетворення перенесення. Кожний контейнер набував нового вигляду шляхом послідовного додавання до координат значень від -500 до 500 з кроком 100. Побітовий метод продемонстрував високі показники стійкості (рис. В.1) – з 12 експериментів лише при першому та дев’ятому експериментах було втрачено невеликий відсоток інформаційних бітів. Метод

паттернів у більшості випадків також стійкий до перенесення, при цьому при п’ятому, дев’ятому та десятому експериментах показники втрат досягають 30-40%.

На рис. В.3 – В.4 відображені графіки залежності відсотка втрачених біт від експерименту при проведенні перетворення повороту. У цьому випадку контейнер поступово повертався на один градус десять разів. Для побітового методу при найнесприятливіших параметрах досягається втрата бітів у приблизно 45%, а для методу паттернів – 100%.

На рис.В.5 – В.6 відображені графіки залежності відсотка втрачених біт від експерименту при проведенні перетворення зсуву за віссю абцис. При цьому x-координата змінюється у 0,01 раз, забезпечуючи поступовий зсув контейнера. Для побітового методу при найнесприятливіших параметрах досягається втрата бітів у приблизно 45%, а для методу паттернів – 100%.

На рис. В.7 – В.8 відображені графіки залежності відсотка втрачених біт від експерименту при проведенні перетворення зсуву за віссю ординат. При цьому y-координата змінюється у 0,01 раз, забезпечуючи поступовий зсув контейнера. Для побітового методу при найнесприятливіших параметрах досягається втрата бітів у приблизно 46%, а для методу паттернів – 100%.

На рис. В.9 – В.10 відображені графіки залежності відсотка втрачених біт від експерименту при проведенні перетворення масштабування для стиснення. При цьому кожна координата змінюється у 0,99 раз, забезпечуючи поступове стиснення контейнера. Для побітового методу при найнесприятливіших параметрах досягається втрата бітів у приблизно 44%, а для методу паттернів – 100%.

На рис. В.11 – В.12 відображені графіки залежності відсотка втрачених біт від експерименту при проведенні перетворення масштабування для розширення. При цьому кожна координата змінюється у 1,01 раз, забезпечуючи поступове розширенн контейнера. Для побітового методу при найнесприятливіших параметрах досягається втрата бітів у приблизно 45%, а для методу паттернів – 100%.

На рис. В.13 – В.14 відображені графіки залежності відсотка втрачених біт від експерименту при проведенні майже афінного перетворення повороту. У цьому випадку контейнер поступово повертався на один градус десять разів, і до кожної координати додавалось певне значення, яке відповідає за шум. Для побітового методу при найнесприятливіших параметрах досягається втрата бітів у приблизно 33%, а для методу паттернів – 100%.

Отримані результати відрізняються від вперше представлених у роботах [23-24] О. Кінзерявого більш високими показниками втрачених біт навіть при меншій кількості разів проведення перетворень над контейнером. Це можна пояснити особливостями програмної реалізації, використовуваними контейнерами, більшим об'ємом відстежуваних винятків при вбудовуванні/вилученні повідомлення тощо.

У загальному випадку отримані результати за допомогою програмно розроблених у даній роботі методів співпадають із представленими результатами О. Кінзерявого. Виключенням є сто відсоткова втрата блоків при застосуванні методу паттернів при таких перетворення як поворот, зсув за осями абцис та ординат, масштабування для стиснення та розширення, а також при деяких параметрах майже афінного перетворення повороту. В обох реалізація найбільш сприятливим є параметр точності у розмірі 6, при якому не залежачи від кількості біт, що кодуються в криву, та похибки відтворення, досягається нульовий (або майже нульовий) відсоток втрачених або неправильно вилучених біт.

За допомогою програмно розроблених методів можлива їх реалізація у різноманітних веб-застосунках для виконання факту перевірки ключових даних. Такими даними можуть бути мітки часу або певна величина, яка формується шляхом відстежування усіх елементів на сторінці і може бути використана для запобігання зловмисного впровадження надбудованих конструкцій, що можуть бути застосовані для перехоплення персональних даних користувачів (така технологія обману носить назву клікджекінгу). Мітки часу можуть бути використані в якості підтвердження існування певних даних у деякий момент.

Окрім цього існує практика створення компонентів DOM-структури із SVG зображень і цілком можливою є ситуація, коли вся веб-сторінка або її складова частина складається із елементів векторної графіки. Цей факт може бути використаний не тільки для збереження даних, а й у якості підтвердження прав авторства, володіння тощо.

4.3 Висновки до четвертого розділу

Приведені результати експериментальні досліджень швидкісних характеристик, коефіцієнтів спотворення та збільшення розміру контейнерів демонструють їх залежність від таких параметрів як кількість біт, що вбудовуються в одну криву, та значення точності координат точок, що розраховуються при поділі. Так, методи паттернів та побітового приховування дозволяють приховувати однакову кількість секретної інформації, однак операція вбудовування блоками забезпечує меншу сегментацію кривої, а звідси – нижче навантаження опису кривої точками. Збільшення розміру приховуваного повідомлення можливе у випадку застосування методу паттернів за рахунок вбудовування більшої кількості блоків даних, проте це закономірно приведе до підвищення розміру файлу зображення.

Результати досліджень показників стійкості до афінних перетворень показали, що метод паттернів програє побітовому. Це пов’язано у першу чергу із алгоритмом вилучення інформації методу паттернів. Так, можлива ситуація втрати цілих блоків інформаційних бітів, коли не знаходиться перша частина послідовності (перший паттерн), а звідси – неможливим стає знаходження наступного коефіцієнту побудови кривої, при якому відбулося б об’єднання сегментів. Така залежність присутня в обох методах, однак побітовий у випадку, коли не виконується умова відтворення, вилучають двійковий «0» (тобто це може бути навіть неправильно вилучений біт), і процес декодування продовжується.

Дослідження стійкості до афінних перетворень також показали, що найбільш сприятливими для приховання та вилучення є випадки, в яких застосовувалась точність 6 для розрахунку координат, при цьому похибка та кількість біт, що приходяться на одну криву, – неважливі.

ВИСНОВКИ

У роботі було проведено дослідження моделей та методів стеганографічних перетворень векторних зображень, на основі точково-заданих кривих Без'є третього ступеню, а також виконана їх програмна реалізація на мові програмування JavaScript.

Найбільш вдалим для проведення стеганографічних перетворень є формат векторної графіки SVG, який завдяки своїй структурі дозволяє легко маніпулювати об'єктами, з яких складається. Його широка підтримка різними платформами також дозволяє підвищити рівень скритності при проведенні передачі секретних даних шляхом передачі звичайних на перший погляд файлів медіа.

Мова програмування JavaScript дозволяє легко взаємодіяти та маніпулювати об'єктами, що зберігаються у файлі SVG. Її широка інтеграція з різними браузерами робить створений на ній веб-застосунок крос-платформним та легко застосовуваним для різного рівню комплексних та професійних задач.

У векторній графіці застосовуються такі математичні об'єкти: точка, пряма лінія, криві першого та другого порядку, криві Без'є, а також вузли, дотичні лінії та керуючі точки. Усі основні векторні елементи будуються за сукупністю точок згідно з відповідною аналітичною формулою із належними параметрами. Для того, аби створювати плавні криві, найчастіше використовується алгоритм інтерполяції Без'є.

Завдяки алгоритму де Кастельжо побудови кривих Без'є можливе проведення стеганографічних перетворень шляхом розбиття кривих на сегменти у відповідних точках, які залежать від параметру побудови. Прийняття правил щодо точок розбиття дозволяє вбудовувати інформаційні двійкові послідовності.

Розроблена програмна реалізація дозволяє виконувати приховування повідомлень у зображення формату SVG двома методами – побітовим та за допомогою паттернів. Для реалізації процесу вбудовування виконується попередня

обробка контейнера з метою приведення строкового запису кривих у вигляд об’єктів, над якими зручно виконувати операції. Крім того, під час такого парсингу вмісту зображення відбувається визначення придатності кривої для вбудовування інформації на основі розрахунку відстані між контрольними точками та аналізу її складових (якщо такі присутні) на належність до інших геометричних об’єктів.

Приведені результати експериментальні дослідження демонструють швидкісні характеристики розроблених методів та залежності розмірів стегоконтейнерів від таких параметрів як кількість біт, що вбудовуються в одну криву, та значення точності координат точок, що розраховуються при поділі, а також показники стійкості до афінних перетворень.

Так, методи паттернів та побітового приховування дозволяють приховувати однакову кількість секретної інформації, однак операція вбудовування блоками забезпечує меншу сегментацію кривої, а звідси – нижче навантаження опису кривої точками. Швидкість вбудовування залежить від кількості кривих у зображенні та від кількості біт, що вбудовується в одну криву, адже вона визначає кількість операцій поділу (розрахунку нових точок). Метод паттернів показав нижчі показники часу при вбудовуванні та нижчі – при вилученні. Це пов’язано з тим, що при вилученні відбувається багаторазова перевірка умови відтворення для кожного кроку з таблиці паттернів.

При дослідженні стійкості до афінних перетворень побітовий метод продемонстрував нижчі показники відсотку втрачених біт. Такі результати пов’язані із залежністю кроку параметра побудови кривої від його попереднього значення. Так, при побітовому вилученні навіть у випадку невиконання умови відтворення відбувається вилучення двійкового «0» і перехід до наступного значення кроку побудови. Відтворення методом паттернів може привести до ситуації втрати блоку повідомлення у розмірі, який відповідає кількості біт, що вбудовуються в одну криву, адже не знайшовши перший паттерн, стає неможливим перехід до наступного елемента без відповідного виявленого кроку зміни параметра побудови із таблиці паттернів.

При порівнянні результатів досліджень з представленою роботою О. Кінзерявого було встановлено, що у загальному випадку при застосуванні розробленої програмної реалізації досягаються схожі показники стійкості до афінних перетворень. Винятком є метод паттернів, який при даній імплементації страждає таким недоліком як втрата цілих блоків даних через неможливість знаходження правильного значення з таблиці паттернів на першому кроці вилучення. Проблема може полягати у точності проміжних розрахунків або у необхідності обробки даного випадку окремим кроком алгоритму.

Практичне застосування методів приховування інформації у векторній графіці полягає у можливості збереження таких ключових даних як мітки часу, права авторства або володіння, значень контрольної суми компонентів DOM-структури для аналізу стану веб-застосунку у випадках застосування анти-клікджекінгових технологій тощо.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Abdulgader Almutairi, “A Comparative Study on Steganography Digital Images: A Case Study of Scalable Vector Graphics (SVG) and Portable Network Graphics (PNG) Images Formats” International Journal of Advanced Computer Science and Applications (IJACSA), 9(1), 2018. URL: http://dx.doi.org/10.14569/IJACSA.2018.090123 (дата звернення: 24.04.2021)
2. What is a vector file? MODassic : веб-сайт. URL: https://modassicmarketing.com/understanding-image-file-types (дата звернення: 10.03.2021).
3. Vector Graphic Definition. TechTerms The Computer Dictionary : веб-сайт. URL: https://techterms.com/definition/vectorgraphic (дата звернення: 24.04.2021).
4. Офіційний сайт Scalable Vector Graphics (SVG). URL: https://www.w3.org/Graphics/SVG/ (дата звернення: 12.05.2021).
5. Can I use... Support tables for HTML5, CSS3 : веб-сайт. URL: https://caniuse.com/?search=svg (дата звернення: 12.05.2021)
6. О. О. Кузнецов, С. П. Євсеєв, О. Г. Король. Стеганографія : навч. посіб. : Вид. ХНЕУ, 2011. – 232 с.
7. Qin J, Luo Y, Xiang X, Tan Y, Huang H. Coverless Image Steganography: A Survey. IEEE Access 2019;7:171372–94. URL: https://doi.org/10.1109/ACCESS.2019.2955452 (дата звернення: 26.05.2021).
8. Cox I, Miller M, Bloom J, Fridrich J, Kalker T. Digital Watermarking and Steganography, 2nd Ed. 2nd Edition. Amsterdam ; Boston: Morgan Kaufmann; 2007.
9. Paraskevov H, Stefanov A. Approach to Find Optimal Graphic Container Using Segmental Steganographic Algorithm. Mathematical and Software Engineering 2018;4:7–11.
10. Schöttle P, Böhme R. Game Theory and Adaptive Steganography. IEEE Transactions on Information Forensics and Security 2016;11:760–73. URL:

https://link.springer.com/chapter/10.1007/978-3-642-36373-3_9 (дата звернення: 26.05.2021).

11. Doncel VR, Nikolaidis N, Pitas I. An optimal detector structure for the fourier descriptors domain watermarking of 2D vector graphics. IEEE Transactions on Visualization and Computer Graphics. 2007 Sep-Oct;13(5):851-863. DOI: 10.1109/tvcg.2007.1050.
12. Wang, X. T., Chengyong Shao, X. Xu and X. Niu. Reversible Data-Hiding Scheme for 2-D Vector Maps Based on Difference Expansion. IEEE Transactions on Information Forensics and Security 2, 2007, P. 311-320.
13. Wu, Dan & Wang, Guozhao & Gao, Xingliang. Reversible Watermarking of SVG Graphics. Communications and Mobile Computing, 2009. CMC '09. WRI International Conference on Volume: 3. P. 385 - 390. DOI: 10.1109/CMC.2009.86.
14. Peng, Fei, Zi-Xing Lin, X. Zhang and Min Long. Reversible Data Hiding in Encrypted 2D Vector Graphics Based on Reversible Mapping Model for Real Numbers. IEEE Transactions on Information Forensics and Security 14, 2019. 2400-2411.
15. Kinzeryavyy O, Kinzeriava I, Olenyuk A, Sulkowsky K. Steganographic Method of Bitwise Information Hiding in Point-Defined Curves of Vector Images. In: Hu Z, Petoukhov S, Dychka I, He M, editors. Advances in Computer Science for Engineering and Education, Cham: Springer International Publishing; 2019, p. 478–86. URL: https://doi.org/10.1007/978-3-319-91008-6_48 (дата звернення: 26.05.2021).
16. K. Park, K. Kim, H. Kang, S. Han. Digital geographical map watermarking using polyline interpolation. IEEE Pacific Rim Conference on Multimedia, 2002. P. 58-65.
17. H. Sonnet, T. Isenberg, J. Dittmann, T. Strothotte. Illustration watermarks for vector graphics. 11th Pacific Conference on Computer Graphics and Applications, 2003. P. 8-10.

18. X. Niu, C. Shao, X. Wang. A survey of digital vector map watermarking. International Journal of Innovative Computing, Information and Control ICIC International. — 2006. — vol.2, №6. — pp. 1301-1316.
19. Карпінець В.В., Яремчук Ю.Є. Аналіз стійкості до зловмисних атак методу вбудовування цифрових водяних знаків у векторні зображення. Вісн. Вінниц. політехн. Інституту № 4, 2011. С. 154-159.
20. Карпінець В.В., Яремчук Ю.Є., Іванішина Д.О. Вирішення проблеми захисту авторських прав векторних зображень в інформаційно-комунікаційних системах. Науково-технічний журнал «Захист інформації» №4 (53), 2011. С. 21–28.
21. Карпінець В.В., Яремчук Ю.Є. Зменшення відхилень координат точок внаслідок вбудовування цифрових водяних знаків у векторні зображення. Прав., нормат. та метрол. забезп. системи захисту інформації в Україні. Вип. 2, 2010. С. 69-78.
22. Карпінець В.В., Яремчук Ю.Є. Методи захисту векторних зображень цифровими водяними знаками : [монографія], 2013. 156 с.
23. Kinzeryavyy O, Kinzeriava I, Olenyuk A, Sulkowsky K. Steganographic Method of Bitwise Information Hiding in Point-Defined Curves of Vector Images. In: Hu Z, Petoukhov S, Dychka I, He M, editors. Advances in Computer Science for Engineering and Education, Cham: Springer International Publishing; 2019, P. 478–86.
24. Kinzeryavyy O. Steganographic methods for hiding data into vector images that are resistant to active attacks based on affine transformations. Thesis. 2015.
25. Foundations of Physically Based Modeling and Animation. Affine Transformations. URL: https://people.cs.clemson.edu/~dhouse/courses/401/notes/affines-matrices.pdf (дата звернення: 12.05.2021).
26. Coste, Arthur. Image Processing : Affine Transformation, Landmarks registration, Non linear Warping, 2012. 10.13140/RG.2.2.13653.27360
27. Маценко В.Г. Комп’ютерна графіка : навч. посіб. Ч. : Рута, 2009. 343 с.

28. Яглом И.М. Идеи и методы аффинной и проективной геометрии : [часть I] М. : Учпедгиз, 1962. 248 с.
29. Scalable Vector Graphics (SVG) 1.1 (Second Edition). W3C : веб-сайт. URL: https://www.w3.org/TR/2011/REC-SVG11-20110816 (дата звернення: 12.05.2021).
30. Scalable Vector Graphics (SVG) 2 n.d. W3C : веб-сайт. URL: https://www.w3.org/TR/SVG2/ (дата звернення: 12.05.2021).
31. Basic Shapes – SVG 1.1 (Second Edition). W3C : веб-сайт. URL: https://www.w3.org/TR/SVG11/shapes.html (дата звернення: 12.05.2021).
32. Чириков С. В. Алгоритмы компьютерной графики (методы растрирования кривых). Учебное пособие – СПб: СПб ГИТМО(ТУ), 2001. – 120 с.
33. Malin Christersson. De Casteljau's Algorithm and Bézier Curves. malinc.se : веб-сайт. URL: http://www.malinc.se/m/DeCasteljauAndBezier.php (дата звернення: 20.03.2021).
34. Кінзерявий О.М. Стеганографічний метод приховування даних у векторних зображеннях. Інфокомунікації - сучасність та майбутнє : третя міжнародна науково-практична конференція молодих вчених — О. : ОНАЗ, 2013. — Ч.3. — С. 160-162.
35. Кінзерявий О.М., Дорошенко Б.С. Побітовий метод приховування у векторні зображення стійкий до афінних перетворень. Механізми управління безпекою підприємств в сучасних умовах господарювання. — К. : Видавництво Європейського університету, 2013. — С. 91-94.
36. Кінзерявий О.М. Шаблонний метод приховування даних у векторні зображення на основі розбиття кривих Без'є. Захист інформації і безпека інформаційних систем: третя міжнародна науково-технічна конференція. — Л. : Українська академія друк., 2014. — С. 86-87.
37. Кінзерявий О.М., Кінзерявий В.М. Метод шаблонного приховування даних у векторні зображення. Інфокомунікації – сучасність та майбутнє: четверта

міжнародна науково-практична конференція. — О. : ОНАЗ, 2014. — Ч.4. — С. 48-52.

38. Рейтинг мов програмування 2021. dou.ua : веб-сайт. URL: https://dou.ua/lenta/articles/language-rating-jan-2021 (дата звернення: 12.05.2021).
39. Top 5 reasons to choose JavaScript over Java for web development. WSA : веб-сайт. URL: https://www.webstackacademy.com/top-5-reasons-choose-javascript-java-web-development (дата звернення: 16.05.2021).
40. Can I use... Support tables for HTML5, CSS3 : веб-сайт. URL: https://caniuse.com/?search=javaScript (дата звернення: 12.05.2021).
41. Top 15 JavaScript IDEs and JS Editors. Jelvix: веб-сайт. URL: https://jelvix.com/blog/best-javascript-ides (дата звернення: 09.05.2021).

ДОДАТОК А

Результати експериментальних досліджень щодо швидкісних характеристик, коефіцієнтів спотворення та збільшення розміру контейнерів

Таблиця А.1 – Середні значення коефіцієнтів збільшення розміру контейнерів при вбудовуванні різної кількості біт в одну криву повідомлення довжиною у 1024 біт методом паттернів

	40 біт на одну криву		60 біт на одну криву		80 біт на одну криву	
Точність координат	5	6	5	6	5	6
Розмір оригінального контейнера, КБ	6.365					
Розмір стегоконтейнера, КБ	21.75	23.45	20.3	23.55	21.85	23.45
Коефіцієнт збільшення розміру, раз	3.4	3.6	3.1	3.6	3.4	3.6

Таблиця А.2 – Середні значення коефіцієнтів збільшення розміру контейнерів при вбудовуванні різної кількості біт в одну криву повідомлення довжиною у 1024 біт побітовим методом

	40 біт на одну криву		60 біт на одну криву		80 біт на одну криву	
Точність координат	5	6	5	6	5	6
Розмір оригінального контейнера, КБ	6.365					
Розмір стегоконтейнера, КБ	34,8	37,7	34,95	37,9	34,95	37,85
Коефіцієнт збільшення розміру, раз	5.4	5.9	5.4	5.9	5.4	5.9

Таблиця А.3 – Середні швидкісні характеристики методу паттернів *encodePattern* при вбудовуванні 40 біт в одну криву повідомлення довжиною 1024 біт

Точність	Похибка	Час вбудовування, с	Швидкість вбудовування, біт/с	Час вилучення, с	Швидкість вилучення, біт/с
5	0.00002	0.398	2572.9	0.307	3335.5
5	0.00004	0.373	2745.3	0.235	4357.4
6	0.00002	0.345	2968.1	0.278	3683.4
6	0.00004	0.38	2694.7	0.346	2959.5

Таблиця А.4 – Середні швидкісні характеристики методу паттернів *encodePattern* при вбудовуванні 60 біт в одну криву повідомлення довжиною 1024 біт

Точність	Похибка	Час вбудовування, с	Швидкість вбудовування, біт/с	Час вилучення, с	Швидкість вилучення, біт/с
5	0.00002	0.363	2820.9	0.317	3230.3
5	0.00004	0.361	2836.6	0.284	1799.6
6	0.00002	0.341	3002.9	0.326	3141.1
6	0.00004	0.362	2828.7	0.349	2934.1

Таблиця А.5 – Середні швидкісні характеристики методу паттернів *encodePattern* при вбудовуванні 80 біт в одну криву повідомлення довжиною 1024 біт

Точність	Похибка	Час вбудовування, с	Швидкість вбудовування, біт/с	Час вилучення, с	Швидкість вилучення, біт/с
5	0.00002	0.332	3084.3	0.248	4129.03
5	0.00004	0.351	2917,3	0.343	2985.4
6	0.00002	0.337	3038,6	0.361	2836.
6	0.00004	0.355	2884,5	0.367	2790.2

Таблиця А.6 – Середні швидкісні характеристики побітового методу *encodeBit* при вбудовуванні 40 біт в одну криву повідомлення довжиною 1024 біт

Точність	Похибка	Час вбудовування, с	Швидкість вбудовування, біт/с	Час вилучення, с	Швидкість вилучення, біт/с
5	0.0002	0.516	1984.5	0.096	10666.6
5	0.0004	0.578	1771.6	0.111	9225.2
6	0.0002	0.498	2056.2	0.138	7420.3
6	0.0004	0.502	2039.8	0.108	9481.5

Таблиця А.7 – Середні швидкісні характеристики побітового методу *encodeBit* при вбудовуванні 60 біт в одну криву повідомлення довжиною 1024 біт

Точність	Похибка	Час вбудовування, с	Швидкість вбудовування, біт/с	Час вилучення, с	Швидкість вилучення, біт/с
5	0.0002	0.498	2056.2	0.108	9481.5
5	0.0004	0.5	2048.0	0.093	11010.7
6	0.0002	0.505	2027.7	0.124	8258.06
6	0.0004	0.495	2068.6	0.125	8192.0

Таблиця А.8 – Середні швидкісні характеристики побітового методу *encodeBit* при вбудовуванні 80 біт в одну криву повідомлення довжиною 1024 біт

Точність	Похибка	Час вбудовування, с	Швидкість вбудовування, біт/с	Час вилучення, с	Швидкість вилучення, біт/с
5	0.0002	0.49	2089.7	0.118	8677.9
5	0.0004	0.464	2206.9	0.108	9481.5
6	0.0002	0.496	2064.5	0.107	9570.09
6	0.0004	0.525	1950.5	0.07	14628.6

Таблиця А.9 – Середні значення коефіцієнту спотворення контейнерів при вбудовуванні різної кількості біт в одну криву повідомлення довжиною у 1024 біт

Метод	40 біт на одну криву	60 біт на одну криву	80 біт на одну криву
Побітовий	0.07	0.05	0.04
Паттернів	0.07	0,06	0,06

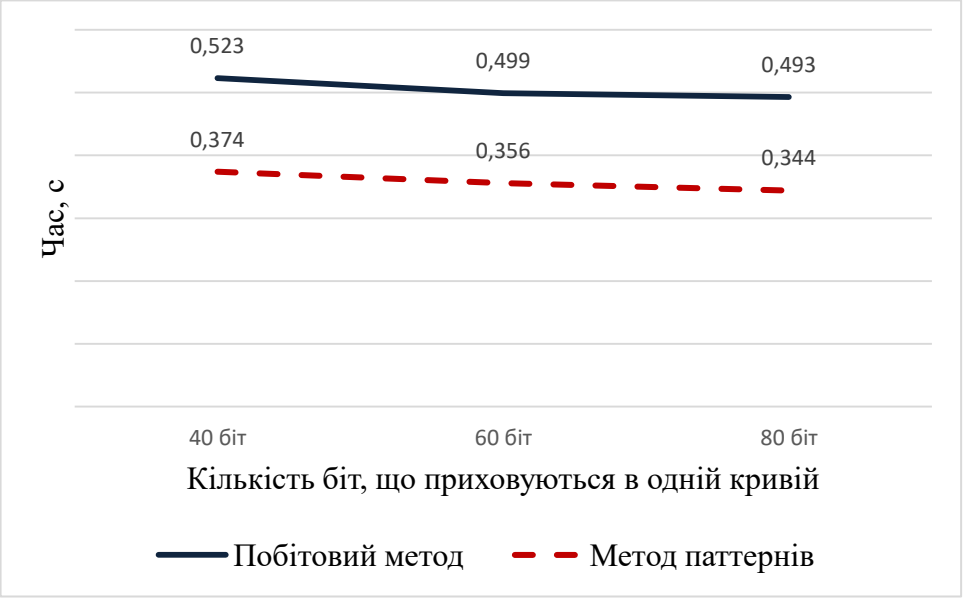

Рисунок А.1 – Графік залежності швидкості вбудовування від кількості біт, що приховуються в одній кривій

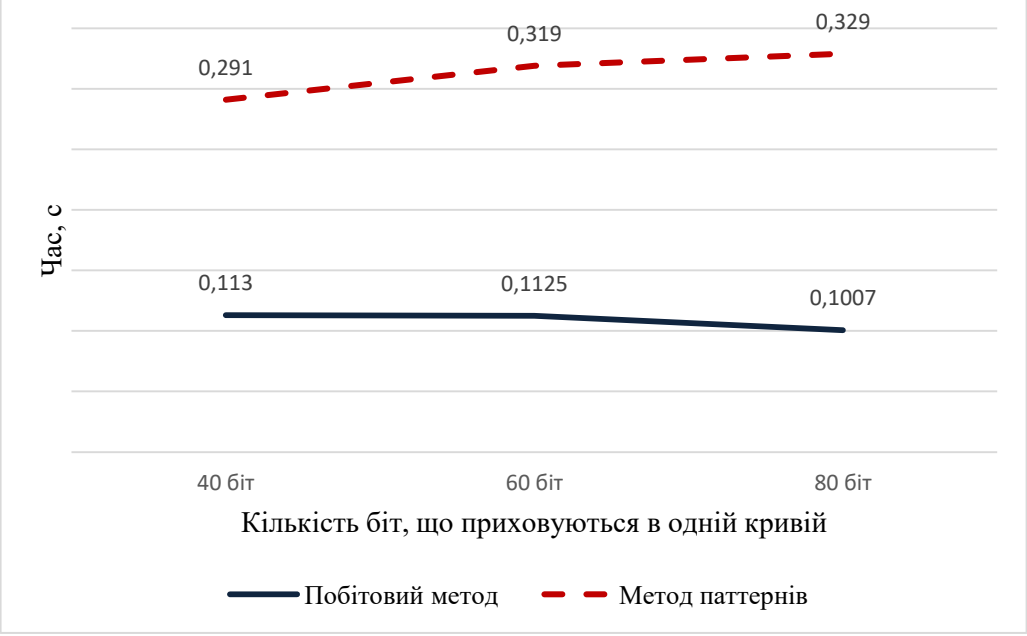

Рисунок А.2 – Графік залежності швидкості вилучення від кількості біт, що приховуються в одній кривій

ДОДАТОК Б

Результати експериментальних досліджень щодо стійкості методів до афінних перетворень

Таблиця Б.1 – Вхідні дані для експериментальних досліджень

Номер експерименту	Біт/крива	Точність	Похибка
e_1	40	5	0,0002
e_2	40	5	0,0004
e_3	40	6	0,0002
e_4	40	6	0,0004
e_5	60	5	0,0002
e_6	60	5	0,0004
e_7	60	6	0,0002
e_8	60	6	0,0004
e_9	80	5	0,0002
e_{10}	80	5	0,0004
e_{11}	80	6	0,0002
e_{12}	80	6	0,0004

Таблиця Б.2 – Результати декодування побітовим методом після перетворення перенесення

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
t_1	0,75	0	0	0	0	0	0	0	2,125	0	0	0
t_2	0,75	0	0	0	0	0	0	0	0	0	0	0
t_3	0	0	0	0	0	0	0	0	0	0	0	0
t_4	0	0	0	0	0	0	0	0	0	0	0	0
t_5	0	0	0	0	0	0	0	0	0	0	0	0
t_6	0	0	0	0	0	0	0	0	0	0	0	0
t_7	0	0	0	0	0	0	0	0	0	0	0	0

Продовження таблиці Б.2 – Результати декодування побітовим методом після перетворення перенесення

t_8	0	0	0	0	0	0	0	0	0	0	0	0
t_9	0,75	0	0	0	0	0	0	0	1,083	0	0	0
t_{10}	0	0	0	0	0	0	0	0	1,042	0	0	0

Таблиця Б.3 – Результати декодування побітовим методом після перетворення повороту

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
t_1	9	0	0	0	14,25	0	0	0	15,67	0	0	0
t_2	24,21	0	0	0	29,54	0	0	0	33,67	0	0	0
t_3	29,92	0	0	0	29,71	0	0	0	32,75	0	0	0
t_4	33,12	0	0	0	36,46	0,375	0	0	37,79	0	0	0
t_5	39,12	0,458	0	0	41,62	2,208	0	0	42,87	0	0	0
t_6	36,17	0,708	0	0	41,04	0	0	0	40,5	0,917	0	0
t_7	40,46	3,5	0	0	43,92	2,792	0	0	42,92	2,042	0	0
t_8	41,46	3,958	0	0	42,87	3,083	0	0	43,37	6	0	0
t_9	40,12	2,875	0	0	43,58	5,333	0	0	42,46	5,67	0	0
t_{10}	42,29	4,875	0	0	44,42	11,67	0	0	44,71	15,46	0	0

Таблиця Б.4 – Результати декодування побітовим методом після перетворення зсуву за віссю абцис

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
t_1	6,583	0	0	0	7,21	0	0	0	15,12	0	0	0
t_2	25,37	0	0	0	26,87	0	0	0	35,21	0	0	0
t_3	37	0,333	0	0	40,54	0,208	0	0	42,08	4,58	0	0
t_4	40,33	11,92	0	0	42,75	9,667	0	0	44	18,12	0	0
t_5	43,37	26,42	0	0	43,71	29,04	0	0	44,87	34,33	0	0
t_6	44,04	34,04	0	0	44,62	37,21	0	0	44,87	41,29	0	0
t_7	44,75	36,75	0	0	44,62	40,08	0	0	44,96	43,33	0	0
t_8	45,08	40,21	0	0	45,46	41,54	0	0	45,29	44	0	0
t_9	45	42,08	0	0	45,17	42,67	0	0	45,5	44,42	0	0
t_{10}	45,37	41,96	0	0	45,62	42,92	0	0	45,46	44,42	0	0

Таблиця Б.5 – Результати декодування побітовим методом після перетворення зсуву за віссю ординат

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
t_1	6,625	0	0	0	6,042	0	0	0	14,62	0	0	0
t_2	24,33	0	0	0	28,79	0	0	0	34,46	0	0	0
t_3	36,83	0	0	0	40,71	1	0	0	40,42	0	0	0
t_4	41,29	7,375	0	0	44,12	8,167	0	0	43,42	11	0	0
t_5	42,79	23,46	0	0	45,08	23,42	0	0	44,25	26,79	0	0
t_6	44	29,96	0	0	45,58	34,96	0	0	45,29	36,62	0	0
t_7	43,62	37,33	0	0	45,83	39,54	0	0	45,46	40,21	0	0
t_8	44,29	40,12	0	0	45,87	42,33	0	0	45,5	43,17	0	0
t_9	45,12	41,37	0	0	45,96	43,21	0	0	45,83	43,08	0	0
t_{10}	45,12	41,54	0	0	46,04	44,08	0	0	45,83	43,58	0	0

Таблиця Б.6 – Результати декодування побітовим методом після перетворення пропорційного масштабування для стиснення

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
t_1	11,54	0	0	0	13,37	0	0	0	18,96	0	0	0
t_2	23,04	0	0	0	27,37	0	0	0	26,71	0	0	0
t_3	29,25	0	0	0	33,83	0	0	0	37,42	0	0	0
t_4	33,08	0,083	0	0	37,17	0,708	0	0	38,83	0	0	0
t_5	37,33	0	0	0	36,17	0	0	0	41,67	0,542	0	0
t_6	40,04	1,042	0	0	39,17	0	0	0	42,17	0	0	0
t_7	39,96	1,917	0	0	41,92	2,167	0	0	43,58	1,583	0	0
t_8	41,33	4,33	0	0	40,25	1,542	0	0	42,96	6,167	0	0
t_9	43,25	3,33	0	0	41,12	4,208	0	0	44,21	6,625	0	0
t_{10}	43,04	3,83	0	0	43,92	5,375	0	0	44,29	5,625	0	0

Таблиця Б.7 – Результати декодування побітовим методом після перетворення пропорційного масштабування для розширення

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
t_1	3,417	0	0	0	16,42	0	0	0	19,04	0	0	0
t_2	29,42	0	0	0	31,12	0	0	0	39,5	0	0	0

Продовження таблиці Б.7 – Результати декодування побітовим методом після перетворення пропорційного масштабування для розширення

t_3	36,04	0,042	0	0	38,08	0,042	0	0	41,29	0,042	0	0
t_4	39,46	0	0	0	42,87	0,208	0	0	44,5	5,167	0	0
t_5	41,79	3,667	0	0	44,37	5	0	0	44,79	9,917	0	0
t_6	43	11,87	0	0	44,46	15,5	0	0	45,21	23,37	0	0
t_7	43,75	19,87	0	0	45,04	23,25	0	0	45,79	30,29	0	0
t_8	44,46	27,87	0	0	45,42	30,87	0	0	45,62	38,62	0	0
t_9	44,42	35,33	0	0	45,46	37,96	0	0	46	40,42	0	0
t_{10}	44,96	38,96	0	0	45,62	39,92	0	0	45,83	43,5	0	0

Таблиця Б.8 – Результати декодування побітовим методом після майже афінного перетворення повороту

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
t_1	10,12	0	0	0	14,46	0	0	0	21,08	0	0	0
t_2	8,542	0	0	0	11,54	0	0	0	16,29	0	0	0
t_3	7,583	0	0	0	7,958	0	0	0	18,08	0	0	0
t_4	6,125	0	0	0	8,458	0	0	0	14,67	0	0	0
t_5	3,833	0	0	0	10,21	0	0	0	9,667	0	0	0
t_6	4,417	0	0	0	6,25	0	0	0	8,708	1,208	0	0,167
t_7	13,46	0,083	0	0,333	17,04	1,333	0	0,75	19,67	1,667	0,083	1,167
t_8	7,667	0,833	0	1,583	19,46	3,333	0	4,417	20,79	9,792	0,333	9,542
t_9	12	2,417	0,33	2,333	20,67	7,083	1,583	6,625	22,79	11,96	2,75	12,12
t_{10}	14,5	9,917	2,5	9,833	19,62	14,25	7,833	15,58	33,67	20,71	10,46	20,42

Таблиця Б.9 – Результати декодування методом паттернів після перетворення перенесення

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
t_1	0	0	0	0	24,63	0	0	0	36,87	33,33	0	0
t_2	0	0	0	0	25,57	0	0	0	26,25	33,33	0	0
t_3	0	0	0	0	26,67	0	0	0	27,71	25,96	0	0
t_4	0	0	0	0	28,33	0	0	0	31,25	25,96	0	0
t_5	0	0	0	0	25,78	0	0	0	31,25	25,96	0	0

Продовження таблиці Б.9 – Результати декодування методом паттернів після перетворення перенесення

t_6	0	0	0	0	28,33	0	0	0	31,25	25,96	0	0
t_7	0	0	0	0	26,67	0	0	0	27,71	25,96	0	0
t_8	0	0	0	0	25,57	0	0	0	26,25	33,33	0	0
t_9	0	0	0	0	24,63	0	0	0	36,87	33,33	0	0
t_{10}	0	0	0	0	24,33	0	0	0	38,75	26,81	0	0

Таблиця Б.10 – Результати декодування методом паттернів після перетворення повороту

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
t_1	43,83	0	0	0	71,37	0	0	0	93,25	36,71	0	0
t_2	67,21	0	0	0	86,21	0	0	0	100	41,42	0	0
t_3	73,75	0	0	0	90,37	0	0	0	93,33	46,33	0	0
t_4	84,75	0	0	0	96,71	0	0	0	100	71,83	0	0
t_5	94,29	0	0	0	100	13,37	0	0	100	70,83	0	0
t_6	92,54	26,46	0	0	100	36,67	0	0	100	58,42	0	0
t_7	95,12	0	0	0	100	26,62	0	0	100	77,29	0	0
t_8	100	30,96	0	0	100	53,87	0	0	100	75,04	0	0
t_9	100	37,96	0	0	100	53,53	0	0	100	74,75	0	0
t_{10}	98,71	38,83	0	0	100	54,79	0	0	100	88,54	0	0

Таблиця Б.11 – Результати декодування методом паттернів після перетворення зсуву за віссю абцис

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
t_1	31,5	0	0	0	68	0	0	0	95,54	37,54	0	0
t_2	73,37	0	0	0	88,12	0	0	0	100	38,96	0	0
t_3	93,37	0	0	0	100	22,46	0	0	100	68,71	0	0
t_4	97,75	38,25	0	0	100	62,04	0	0	100	95,33	0	0
t_5	98,87	70,08	0	0	100	82,42	0	0	100	92,92	0	0
t_6	100	89,21	0	0	100	92,71	0	0	100	100	0	0
t_7	100	95,42	0	0	100	100	0	0	100	100	0	0
t_8	100	97,83	0	0	100	100	0	0	100	100	0	0

Продовження таблиці Б.11 – Результати декодування методом паттернів після перетворення зсуву за віссю абцис

t_9	100	98,87	0	0	100	100	0	0	100	100	0	0
t_{10}	100	98,87	0	0	100	100	0	0	100	100	25,87	0

Таблиця Б.12 – Результати декодування методом паттернів після перетворення зсуву за віссю ординат

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
t_1	38,04	0	0	0	57,92	0	0	0	93	44,25	0	0
t_2	72,16	0	0	0	92,37	0	0	0	100	53,67	0	0
t_3	86,62	0	0	0	96,33	27,37	0	0	100	77	0	0
t_4	97,54	32,62	0	0	100	46,33	0	0	100	86,58	0	0
t_5	100	60,12	0	0	100	65,46	0	0	100	95,12	0	0
t_6	100	79,04	0	0	100	92,79	0	0	100	97,62	0	0
t_7	100	88,08	0	0	100	97,21	0	0	100	100	0	0
t_8	100	91,08	0	0	100	100	0	0	100	100	0	0
t_9	100	95,54	0	0	100	100	0	0	100	100	22,83	0
t_{10}	100	96,62	0	0	100	100	0	0	100	100	25,62	0

Таблиця Б.13 – Результати декодування методом паттернів після перетворення пропорційного масштабування для стиснення

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
t_1	46	0	0	0	69	0	0	0	95,29	44,58	0	0
t_2	67,37	0	0	0	83	0	0	0	100	41,54	0	0
t_3	78,37	0	0	0	92,21	0	0	0	100	47,83	0	0
t_4	87,92	0	0	0	95,46	33,04	0	0	100	41,17	0	0
t_5	94,12	13,04	0	0	98,62	34,29	0	0	100	57,79	0	0
t_6	95,5	9,5	0	0	100	4,87	0	0	100	66,12	0	0
t_7	98,96	12,62	0	0	100	48,5	0	0	100	72,33	0	0
t_8	99	37,21	0	0	100	43,62	0	0	100	76,62	0	0
t_9	100	38,62	0	0	100	53,46	0	0	100	86,54	0	0
t_{10}	100	28,83	0	0	100	52,5	0	0	100	88,71	0	0

Таблиця Б.14 – Результати декодування методом паттернів після перетворення пропорційного масштабування для розширення

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
t_1	47,21	0	0	0	69,25	0	0	0	97,54	33,13	0	0
t_2	72,92	0	0	0	90,54	0	0	0	100	47,29	0	0
t_3	89,42	0	0	0	96,79	0	0	0	100	66,92	0	0
t_4	94,62	36,37	0	0	100	19,58	0	0	100	74,71	0	0
t_5	97,54	37,79	0	0	100	42,54	0	0	100	86,38	0	0
t_6	100	43,71	0	0	100	56,21	0	0	100	95,17	0	0
t_7	100	60,12	0	0	100	73,54	0	0	100	97,54	0	0
t_8	98,83	71	0	0	100	97,13	0	0	100	100	0	0
t_9	100	86,96	0	0	100	100	0	0	100	100	0	0
t_{10}	100	90,58	0	0	100	100	0	0	100	100	32,25	0

Таблиця Б.15 – Результати декодування методом паттернів після майже афінного перетворення повороту

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
t_1	39,83	0	0	0	73,71	0	0	0	91,08	36,08	0	0
t_2	40,79	0	0	0	60,25	0	0	0	89,71	35,92	0	0
t_3	38,58	0	0	0	63,46	0	0	0	90,5	32,33	0	0
t_4	38,42	0	0	0	40,88	0	0	0	83,04	37,54	0	0
t_5	50,25	0	0	0	65,5	0	0	0	92,33	41,96	13,79	13,79
t_6	34,08	0	0	0	60,42	0	0	0,08	86	45,42	13,79	37,04
t_7	49,25	35,88	0	14,25	58,71	46,63	0	0,58	97,63	48,29	13,79	29,71
t_8	46,88	0,96	0	0,67	64,67	43,5	0,21	38,75	100	57,38	42,46	59,92
t_9	54,79	33,96	14,21	1,58	80,96	47,88	28,88	47,79	91,21	68,5	46,71	62,88
t_{10}	49,88	6,58	0,75	4,88	80,63	55,58	49,5	49,13	100	83,08	59,79	77,46

ДОДАТОК В

Результати експериментальних досліджень щодо стійкості методів до афінних перетворень у графічному вигляді

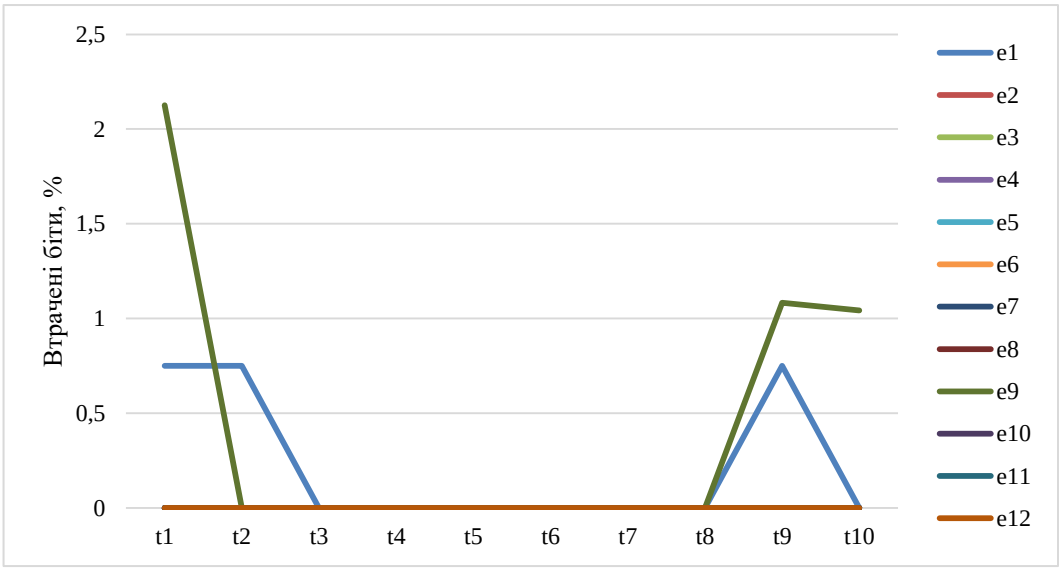

Рисунок В.1 – Результати вилучення інформації з контейнерів після перетворення перенесення побітовим методом

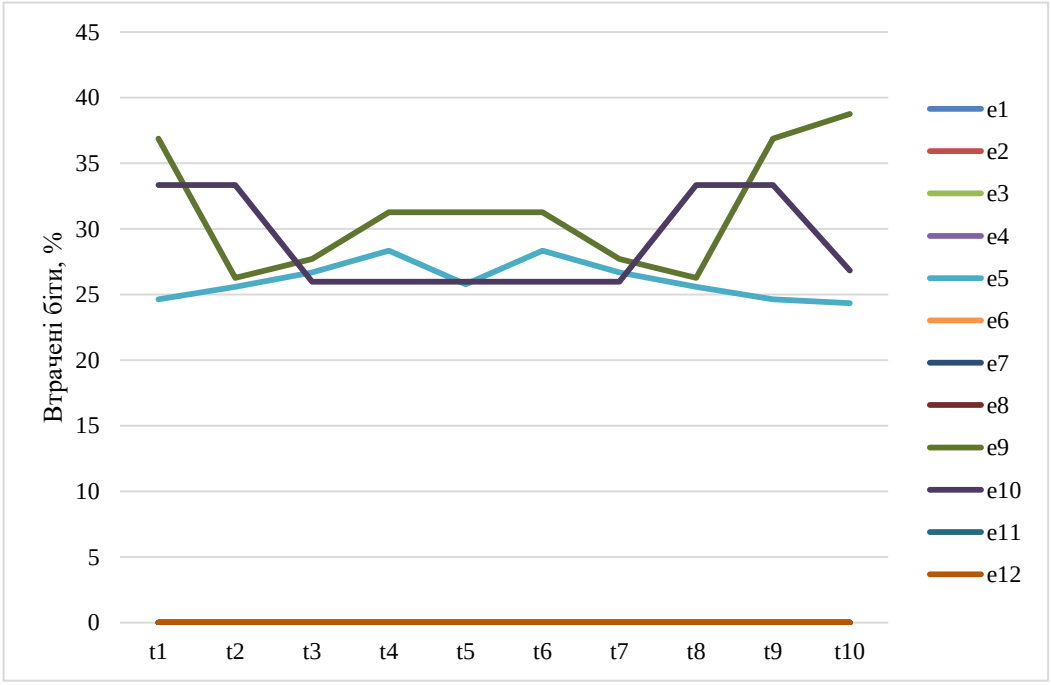

Рисунок В.2 – Результати вилучення інформації з контейнерів після перетворення перенесення методом паттернів

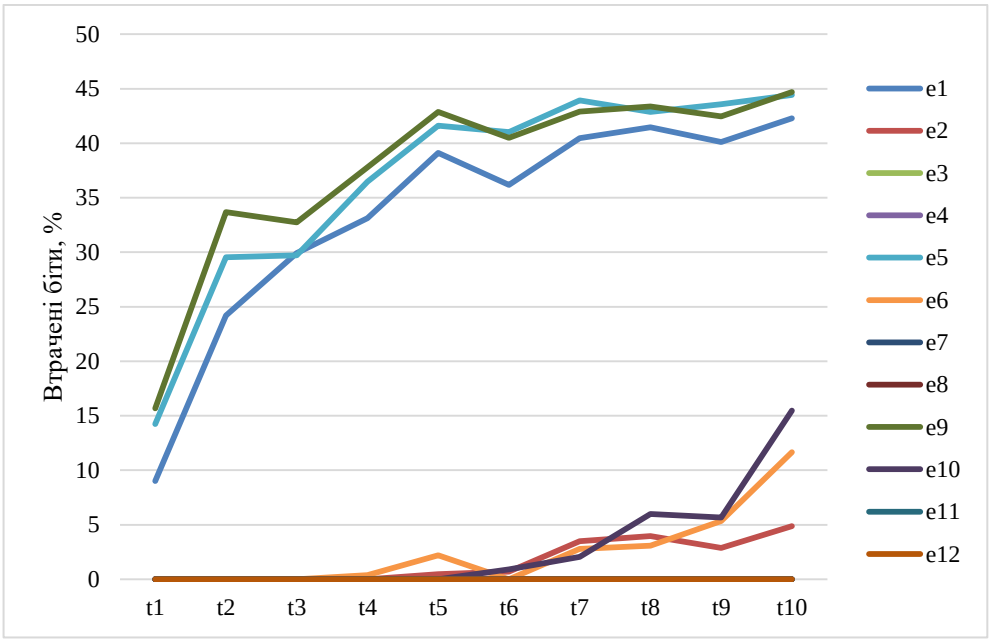

Рисунок В.3 – Результати вилучення інформації з контейнерів після перетворення повороту побітовим методом

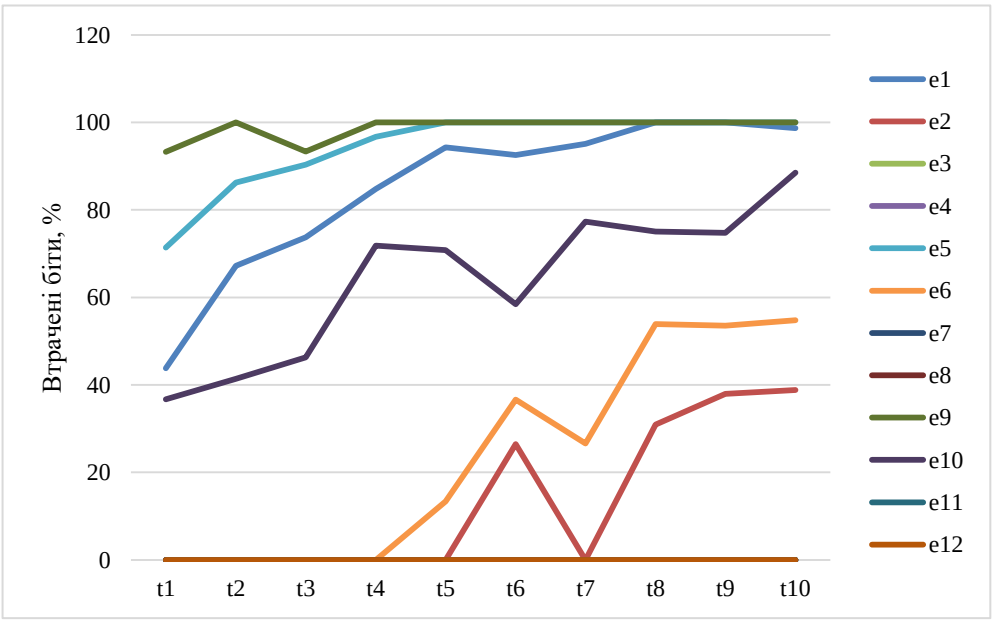

Рисунок В.4 – Результати вилучення інформації з контейнерів після перетворення повороту методом паттернів

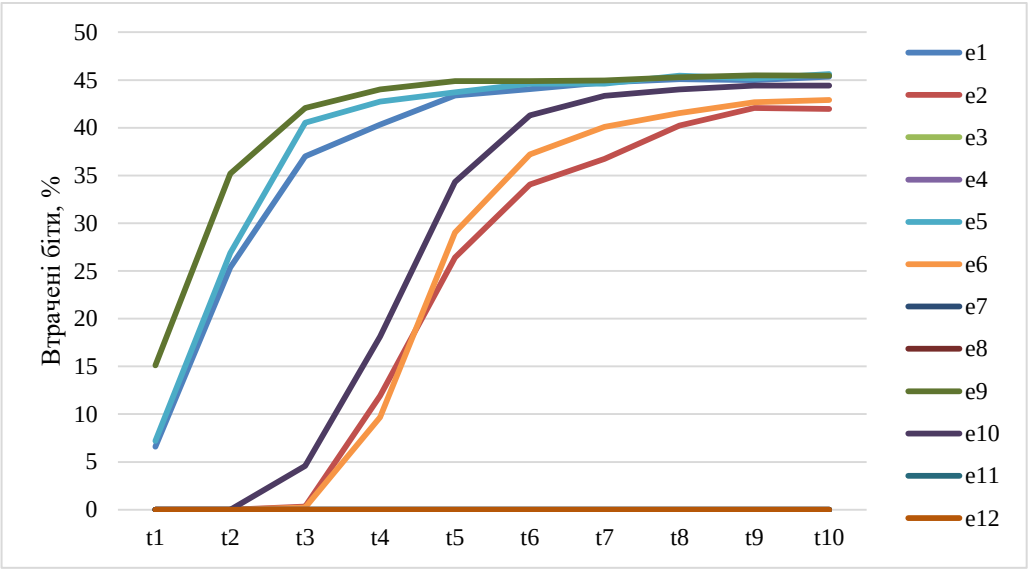

Рисунок В.5 – Результати вилучення інформації з контейнерів після перетворення зсуву за віссю абцис побітовим методом

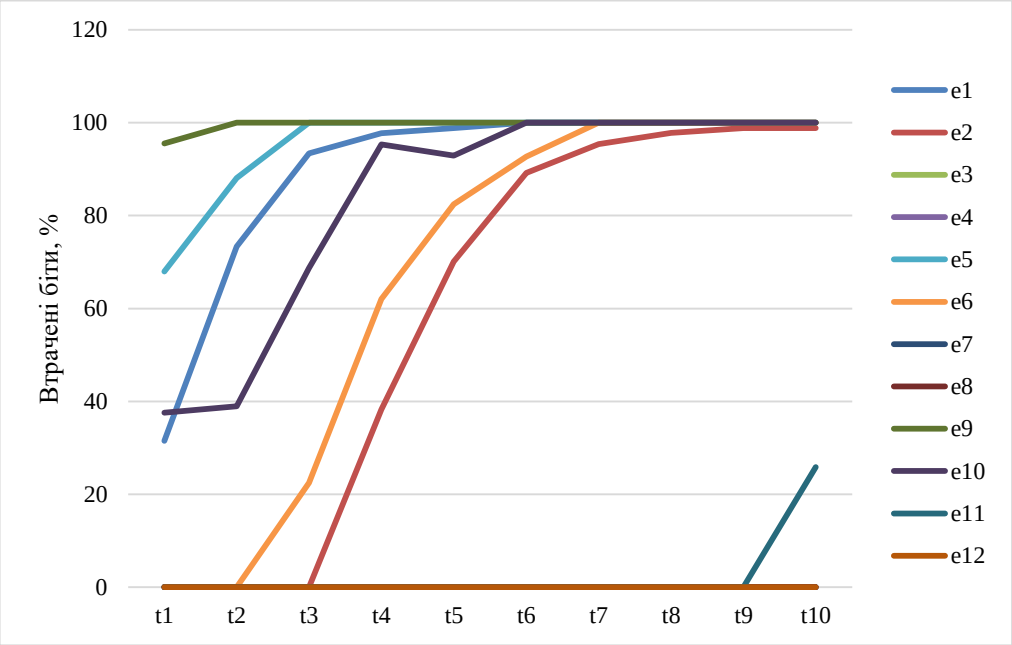

Рисунок В.6 – Результати вилучення інформації з контейнерів після перетворення зсуву за віссю абцис методом паттернів

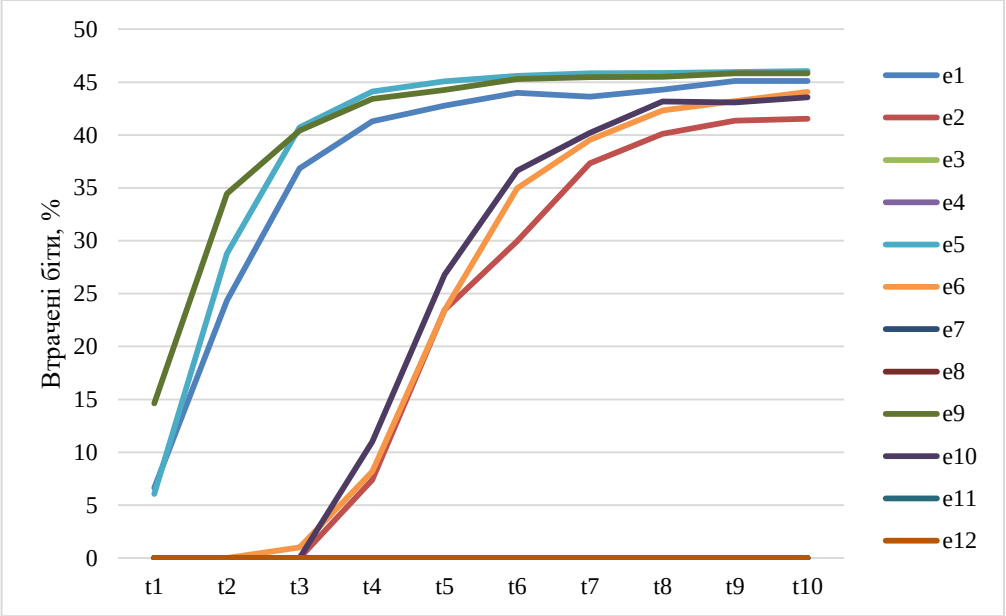

Рисунок В.7 – Результати вилучення інформації з контейнерів після перетворення зсуву за віссю ординат побітовим методом

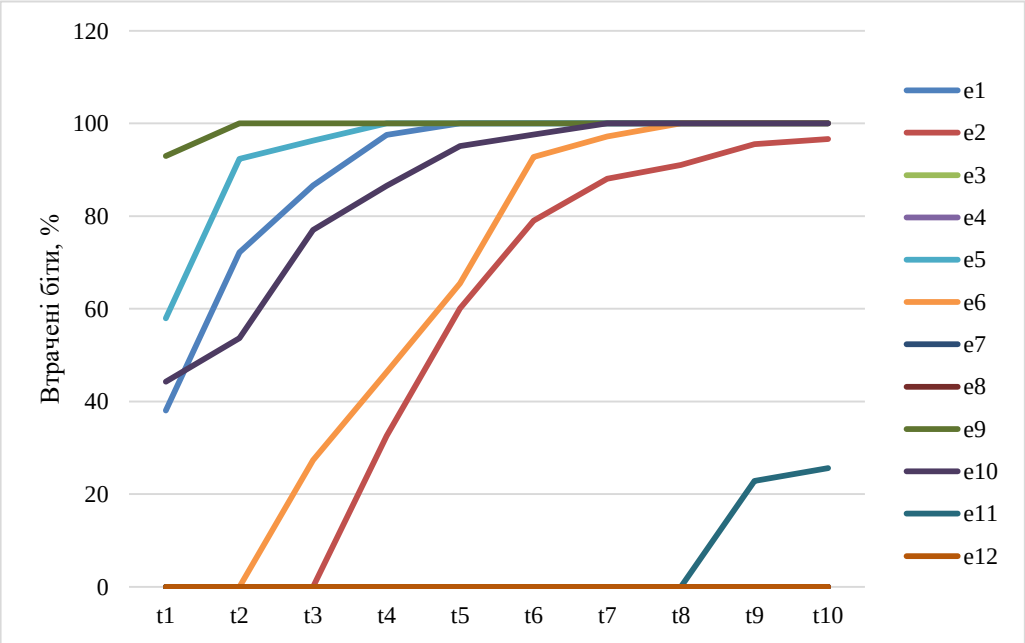

Рисунок В.8 – Результати вилучення інформації з контейнерів після перетворення зсуву за віссю ординат методом паттернів

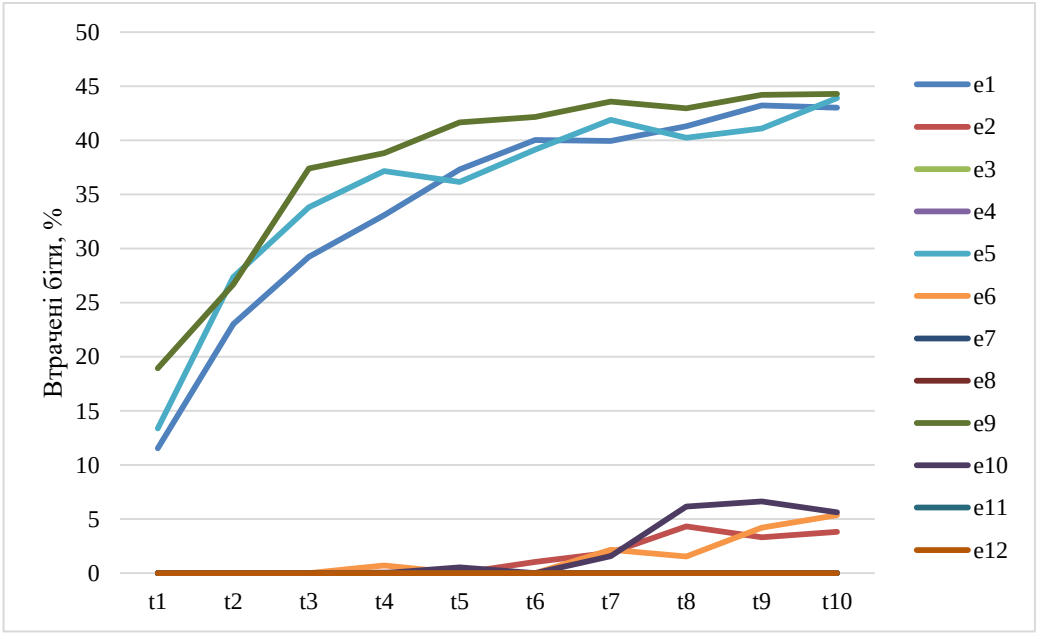

Рисунок В.9 – Результати вилучення інформації з контейнерів після перетворення масштабування для стиснення побітовим методом

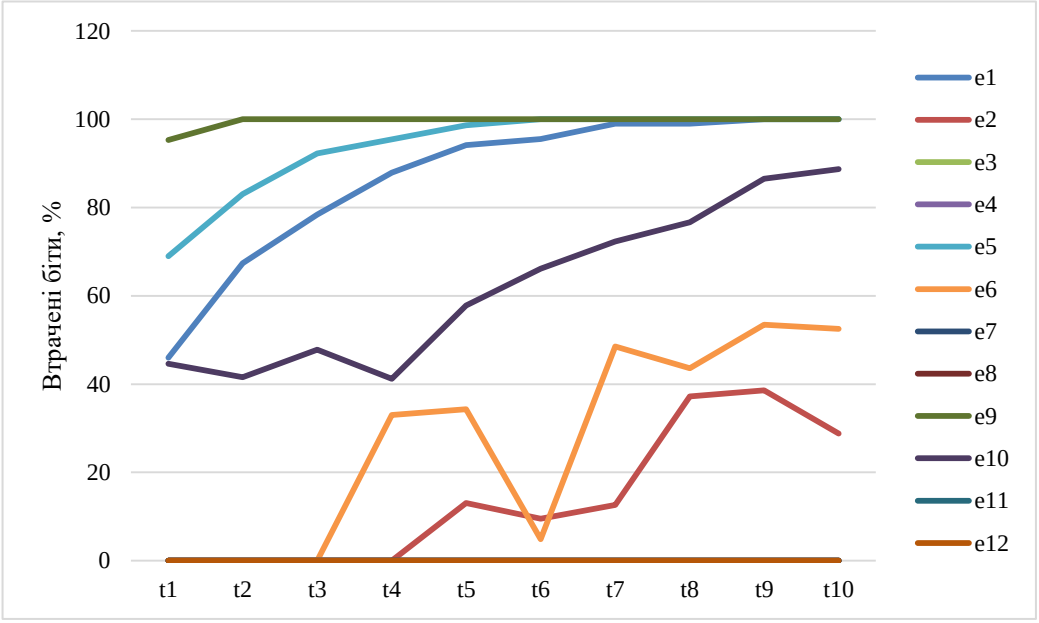

Рисунок В.10 – Результати вилучення інформації з контейнерів після перетворення масштабування для стиснення методом паттернів

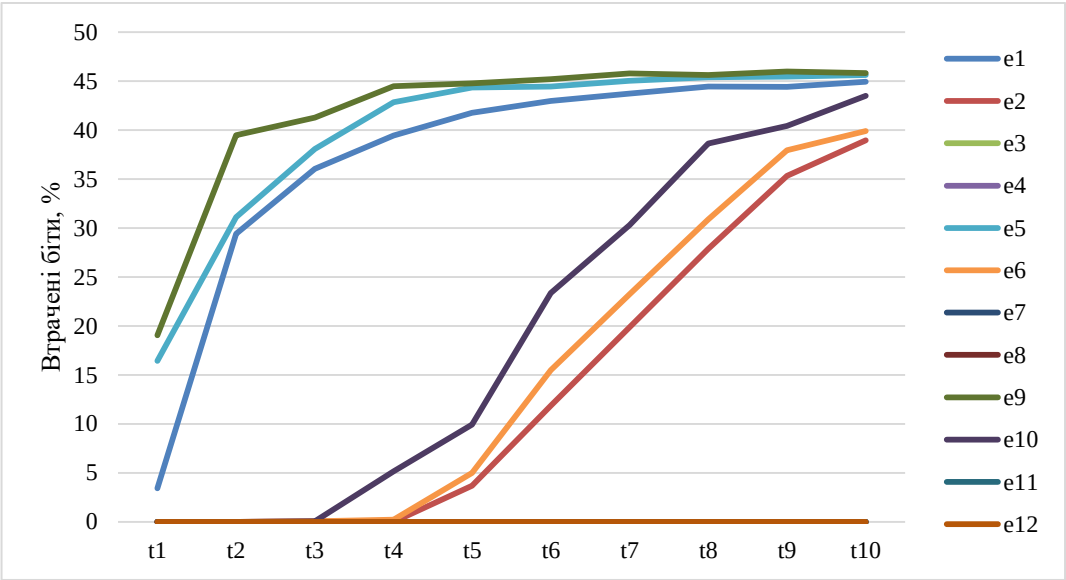

Рисунок В.11 – Результати вилучення інформації з контейнерів після перетворення масштабування для розширення побітовим методом

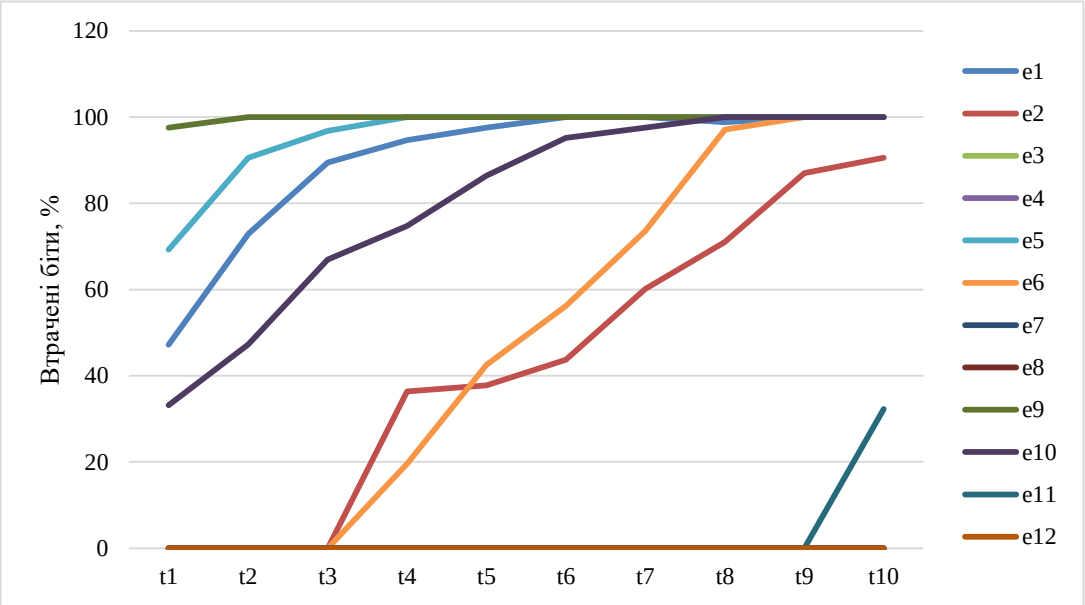

Рисунок В.12 – Результати вилучення інформації з контейнерів після перетворення масштабування для розширення методом паттернів

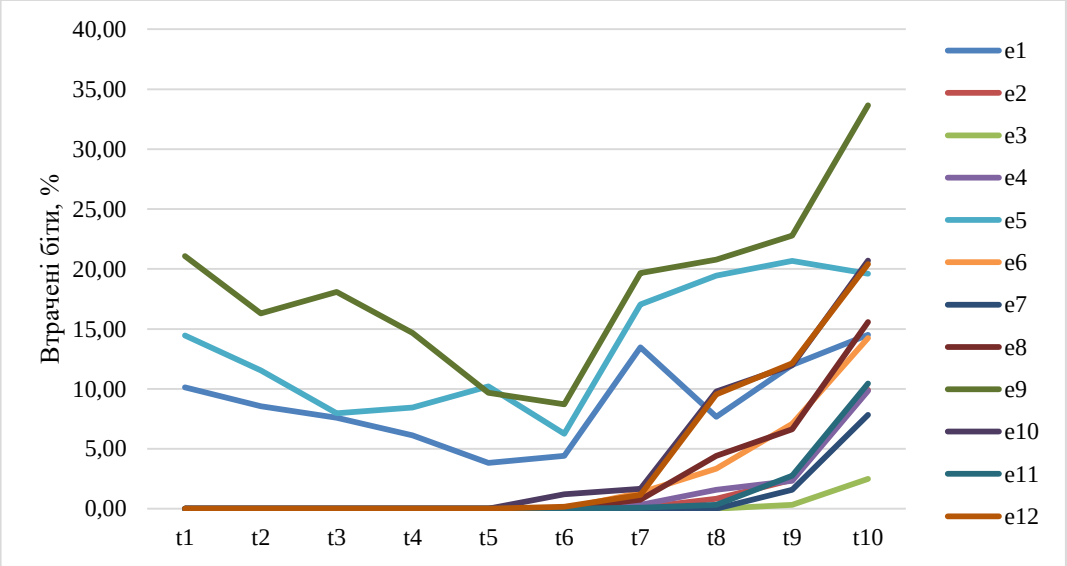

Рисунок В.13 – Результати вилучення інформації з контейнерів після майже афінного перетворення повороту побітовим методом

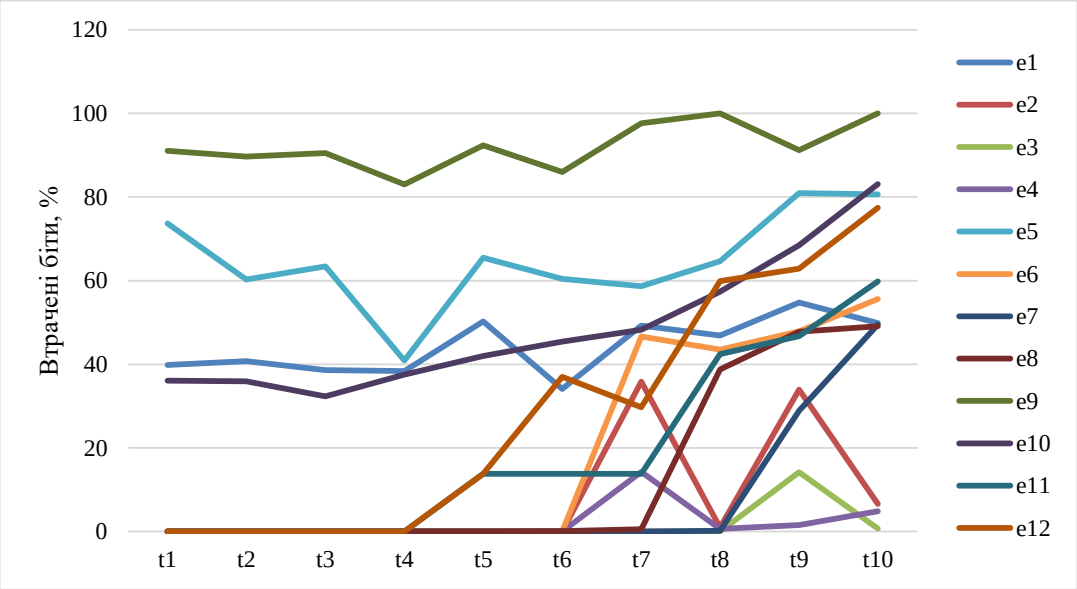

Рисунок В.14 – Результати вилучення інформації з контейнерів після майже афінного перетворення повороту методом паттернів

ДОДАТОК Г

Лістинги програмних засобів

1) Вихідний код функцій обробки контейнера

```
function selectCurves(stringPath) {
    let res = [];
    let splittedPath = stringPath.split(' ').filter(elem => elem !== '');
    let curveLiterals = getAllIndexes(splittedPath, /[Mm]/)
    let segmentPoints = [], segments = [];
    for (let i = 0; i < curveLiterals.length; i++) {
        if (i === curveLiterals.length - 1) {
            segmentPoints.push(splittedPath.slice(curveLiterals[i],
splittedPath.length))
        } else {
            segmentPoints.push(splittedPath.slice(curveLiterals[i],
curveLiterals[i + 1]))
        }
        let segmentsLiterals = getAllIndexes(segmentPoints[i], /[CcLlHhVvZz]/);
        let splittedPoints = [];
        for (let index = 0; index < segmentsLiterals.length; index++) {
            let points = [];
            if (index === segmentsLiterals.length - 1) {
                points = segmentPoints[i].slice(segmentsLiterals[index],
splittedPath.length)
            } else {
                points = segmentPoints[i].slice(segmentsLiterals[index],
segmentsLiterals[index + 1])
            }
            splittedPoints.push(new segment('none', points[0], points.slice(1,
points.length)))
        }
        res.push(new Curve(segmentPoints[i][0], segmentPoints[i][1],
splittedPoints))
    }
    return res;
}

function extractCurves(curveObj, requiredOrder) {
    let res = [];
    for (let i = 0; i < curveObj.length; i++) {
        let chunkedSegments = [];
        for (let j = 0; j < curveObj[i].segments.length; j++) {
            if (curveObj[i].segments[j].points.length > requiredOrder) {
                chunkedSegments.push(chunkByFixedLength(curveObj[i].segments[j],
3));
            } else if (curveObj[i].segments[j].literal.match(/[Ll]/)) {
                chunkedSegments.push(curveObj[i].segments[j]);
            } else {
                chunkedSegments.push(new segment('none',
curveObj[i].segments[j].literal, curveObj[i].segments[j].points));
            }
        }
        if (chunkedSegments.length > 0) {
            res.push(new Curve(curveObj[i].literal, curveObj[i].startPoint,
chunkedSegments.flat()))
```

```
        } else {
            res.push(curveObj[i])
        }
    }
return res;
}

function getCanonicalForm(curves) {
    for (let i = 0; i < curves.length; i++) {
        for (let j = 0; j < curves[i].segments.length; j++) {
            if (!curves[i].segments[j].literal.match(/[LlVvHhZz]/)) {
                if (j === 0) {
                    curves[i].segments[j].points.unshift(curves[i].startPoint)
                } else {
                    curves[i].segments[j].points.unshift(curves[i].segments[j -
1].points[curves[i].segments[j - 1].points.length - 1])
                }
            }
        }
    }
    return curves;
}

function checkIfCurveIsPolybezierAndTransform(curves) {
    let register = "C";
    for (let i = 0; i < curves.length; i++) {
        if (curves[i].segments.length >= 1) {
            for (let j = 0; j < curves[i].segments.length; j++) {
                if (curves[i].segments[j].literal !== register) {
                    if (curves[i].segments[j].literal.match(/[Zz]/)) {
                        curves[i].segments[j].literal =
curves[i].segments[j].literal.toUpperCase();
                        curves[i].segments[j].points.shift();
                        break;
                    }
                    let startPoint = curves[i].segments[j].points[0];
                    curves[i].segments[j].points =
curves[i].segments[j].points.map(el => addPoints(el, startPoint));
                    curves[i].segments[j].points[0] =
subtractPoints(curves[i].segments[j].points[0], startPoint);
                    curves[i].segments[j].literal =
curves[i].segments[j].literal.toUpperCase();
                    curves[i].segments = refreshSegments(curves[i].segments);
                }
            }
        }
    }
    return curves;
}

function isValid(points, minDistance) {
    let validness = 'yes'
    if (points.length < curveOrder){
        return validness = 'no'
    }
    for (let i = 1; i < points.length; i++) {
        if (getDistanceBetween2points(points[i-1], points[i]) <= minDistance) {
            validness = 'no';
            break;
        }
    }
    return validness;
```

```
}
```

2) Вихідний код *encodeBit*

```
function encodeBit(segments, inputValue, parameter, precision) {
    if (segments.valid === 'yes' && typeof inputValue !== 'undefined') {
        let encodedSegment = [], splittedSegment = [];
        let indexOfSegment = 0, finalParameter = 0;
        for (let t = parameter, index = 0; index < inputValue.length; index++) {
            t = parseFloat(t.toFixed(precision))
            if (inputValue[index] === '1') {
                if (indexOfSegment === 0) {
                    splittedSegment.push(splitCurve(t, segments.points,
precision))
                    splittedSegment[indexOfSegment][0].shift();
                    encodedSegment.push(splittedSegment[indexOfSegment][0])
                } else {
                    splittedSegment.push(splitCurve(t,
splittedSegment[indexOfSegment - 1][1], Number(precision)))
                    splittedSegment[indexOfSegment][0].shift();
                    encodedSegment.push(splittedSegment[indexOfSegment][0])
                }
                indexOfSegment++;
            }
            t += parameter;
            finalParameter = parseFloat(t.toFixed(precision));
        }
        splittedSegment[indexOfSegment - 1][1].shift();
        encodedSegment.push(splittedSegment[indexOfSegment - 1][1])
        return {
            encoded: encodedSegment.map(elem => new segment(segments.valid,
segments.literal, elem)),
            finalParameter: finalParameter
        };
    } else {
        segments.points.shift();
        return {
            encoded: [segments],
            finalParameter: null
        }
    }
}
```

3) Вихідний код *encodePattern*

```
function encodePattern(segments, inputValue, precision, startStep, patternLength,
maxBitQuantity, patternTable) {
    if (segments.valid === 'yes' && typeof inputValue !== 'undefined') {
        let encodedSegment = [], splittedSegment = [];
        inputValue = chunkString(inputValue, patternLength)
        let indexOfSegment = 0;
        let finalParameter = 0;
        for (let t = startStep, index = 0; index < inputValue.length; index++) {
            if (patternTable.find(elem => elem.pattern === inputValue[index])) {
                let currentPosition = patternTable.find(elem => elem.pattern ===
inputValue[index]);
                if (indexOfSegment === 0) {
                    splittedSegment.push(splitCurve(parseFloat((t +
currentPosition.step).toFixed(precision)), segments.points, precision))
                    splittedSegment[indexOfSegment][0].shift();
                    encodedSegment.push(splittedSegment[indexOfSegment][0])
                } else {
```

```
                    splittedSegment.push(splitCurve(parseFloat((t +
currentPosition.step).toFixed(precision)), splittedSegment[indexOfSegment - 1][1],
Number(precision)))
                    splittedSegment[indexOfSegment][0].shift();
                    encodedSegment.push(splittedSegment[indexOfSegment][0])
                }
                indexOfSegment++;
                t += currentPosition.step;
                t = parseFloat(t.toFixed(precision))
            }
            finalParameter = parseFloat(t.toFixed(4));
        }
        splittedSegment[indexOfSegment - 1][1].shift();
        encodedSegment.push(splittedSegment[indexOfSegment - 1][1])
        return {
            encoded: encodedSegment.map(elem => new segment(segments.valid,
segments.literal, elem)),
            finalParameter: finalParameter,
        };
    } else {
        segments.points.shift()
        return {
            encoded: [segments],
            finalParameter: null,
        };
    }
}
```

4) Вихідний код *decodeBit*

```
function decodeBit(maxBitQuantity, accuracy, segments, parameterStep, precision,
steganoKey) {
    if (segments.length > 1) {
            let output = [];
            if (typeof steganoKey === 'undefined') {
                steganoKey = (maxBitQuantity + 1) * parameterStep;
            }
            let merged = segments.slice();
            for (let t = steganoKey - parameterStep; t >= 0; t -= parameterStep) {
                let i = merged.length - 1;
                if (!(merged[i].literal.match(/[ZzLlHhVv]/))) {
                    t = parseFloat(t.toFixed(precision))
                    let tmp = [];
                    if (i === segments.length - 1) {
                        tmp = [segments[i - 1], segments[i]];
                    } else {
                        if (merged.length !== 0) {
                            tmp = [merged[merged.length - 2], merged[merged.length
- 1]];
                        }
                    }
                    if (i === 0 || t === 0) {
                        if (output.length !== maxBitQuantity) {
                            merged.splice(1, 1);
                            output.push('0');
                        }
                        break;
                    } else {
                        let p02_2 = getAdditionalPoint(t, tmp[1].points[0],
tmp[1].points[1]);
                        let p11 = getAdditionalPoint(t, tmp[1].points[1],
tmp[1].points[2]);
                        let p02_3 = getCurvePoint(t, tmp[0].points[1], p11);
```

```
                        if (isDifferenceSmaller(p02_2, p02_3, accuracy) === true
|| isDifferenceSmaller(tmp[1].points[2], p02_2, accuracy) === true ||
isDifferenceSmaller(tmp[1].points[2], p02_3, accuracy) === true) {
                            output.push('1');
                            merged.splice(i - 1, 2, mergeSegments(t, tmp[0],
tmp[1]));
                            i--;
                        } else {
                            output.push('0');
                        }
                    }
                } else {
                    merged.pop();
                }
            }
            if (output.length < maxBitQuantity) {
                output = makeNbitAtEnd(output, maxBitQuantity)
            }
            return output.reverse().join('');
    }
}
```

5) Вихідний код *decodePattern*

```
function decodePattern(maxBitQuantity, accuracy, segments, startStep, precision,
steganoKey, patternTable) {
    if (!segments[0].literal.match(/[ZzLlHhVv]/) && segments.length >
maxBitQuantity / patternLength) {
        if (steganoKey !== null) {
            let output = [], parameter = 0;
            let merged = segments.slice();
            let t = steganoKey;
            let flag = true;
            do {
                let i = merged.length - 1;
                t = parseFloat(t.toFixed(4))
                let tmp = [];
                if (i === segments.length - 1) {
                    merged.splice(merged.length - 2, 2, mergeSegments(t,
merged[merged.length - 2], merged[merged.length - 1]));
                    i--;
                    tmp = [merged[merged.length - 2], merged[merged.length - 1]];
                    for (let j = 0; j < patternTable.length; j++) {
                        parameter = parseFloat((steganoKey -
patternTable[j].step).toFixed(4));
                        let p02_2 = getAdditionalPoint(parameter,
tmp[1].points[0], tmp[1].points[1]);
                        let p11 = getAdditionalPoint(parameter, tmp[1].points[1],
tmp[1].points[2]);
                        let p02_3 = getCurvePoint(parameter, tmp[0].points[1],
p11);

                        if (isDifferenceSmaller(p02_2, p02_3, accuracy) === true
|| isDifferenceSmaller(tmp[1].points[2], p02_2, accuracy === true ||
isDifferenceSmaller(tmp[1].points[2], p02_3, accuracy) === true)) {

                            output.push(patternTable.find(elem => elem.step ===
patternTable[j].step).pattern);
                            tmp = mergeSegments(parameter, merged[merged.length -
2], merged[merged.length - 1]);
                            merged.splice(merged.length - 2, 2, tmp);
                            t = parameter;
                            break;
```

```
                        }
                    }
                } else if (merged.length === 1) {
                    for (let j = 0; j < patternTable.length; j++) {
                        let nextParameter = parseFloat((parameter -
patternTable[j].step).toFixed(4));
                        if (nextParameter === startStep) {
                            output.push(patternTable.find(elem => elem.step ===
patternTable[j].step).pattern);
                            flag = false;
                            break;
                        }
                    }
                } else {
                    let tmp = [merged[merged.length - 2], merged[merged.length -
1]];
                    for (let j = 0; j < patternTable.length; j++) {
                        parameter = parseFloat((t -
patternTable[j].step).toFixed(precision));
                        let p02_2 = getAdditionalPoint(parameter,
tmp[1].points[0], tmp[1].points[1]);
                        let p11 = getAdditionalPoint(parameter, tmp[1].points[1],
tmp[1].points[2]);
                        let p02_3 = getCurvePoint(parameter, tmp[0].points[1],
p11);

                        if (isDifferenceSmaller(p02_2, p02_3, accuracy) === true
|| isDifferenceSmaller(tmp[1].points[2], p02_2, accuracy === true ||
isDifferenceSmaller(tmp[1].points[2], p02_3, accuracy) === true)) {
                            output.push(patternTable.find(elem => elem.step ===
patternTable[j].step).pattern);
                            tmp = mergeSegments(parameter, merged[merged.length -
2], merged[merged.length - 1]);

                            merged.splice(merged.length - 2, 2, tmp);
                            t = parameter
                            break;
                        }
                        if (j === patternTable.length - 1) {
                            console.error('Error of decoding! Could not find
parameter..')
                            return;
                        }
                    }
                }
            }
            while (flag !== false);
            return output.reverse().join('');
        }
    }
}
```

6) Вихідний код реалізації об’єкту кривої

```
class Curve {
    literal;
    startPoint;
    segments;

    constructor(literal, startPoint, segments) {
        this.literal = literal;
        this.startPoint = startPoint;
        this.segments = segments;
```

```
    }
}


class segment{
    literal;
    points;
    valid;
    constructor(validness, literal, points) {
        this.valid = validness;
        this.literal = literal;
        this.points = points;
    }

    toString(){
        let points = '';
        for (let i = 0; i < this.points.length; i++) {
            points += this.points[i].toString();
        }
        return String(this.literal) + ' ' + points;
    }
}

class Point {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }

    multiplyPointOnNumber(number) {
        return new Point(this.x * number, this.y * number);
    }

    toString() {
        return (JSON.stringify(this.x) + ',' + JSON.stringify(this.y) + ' ');
    };
}
```

7) Вихідний код реалізації допоміжних функцій

```
function dividePointOnNumber(point, number){
    return new Point(round((point.x / number), precision), round((point.y /
number), precision));
}

function addPoints(point1, point2) {
    return new Point(round((point1.x + point2.x), precision), round((point1.y +
point2.y), precision));
}

function subtractPoints(point1, point2){
    return new Point(point1.x - point2.x, point1.y - point2.y);

}

function getCurvePoint(parameter, point1, point2) {//get point as expression (1-
t)*P1+t*P2 (de Casteljau), i.e. to get point of curve
    return addPoints(point1.multiplyPointOnNumber(1 - parameter),
        point2.multiplyPointOnNumber(parameter));
}

function getAdditionalPoint(parameter, point1, point2){//get point as expression
(point1 - t*point2)/(1 - t)
```

```
    return
dividePointOnNumber(subtractPoints(point1,(point2.multiplyPointOnNumber(parameter)
)), (1 - parameter))
}

function getInterimPointOnAdditionalLines(parameter, point1, point2, precision){
     let res = addPoints(point1, subtractPoints(point2,
point1).multiplyPointOnNumber(parameter))
    return new Point(round(res.x, Number(precision)), round(res.y,
Number(precision)));
}

function getDifferenceBetweenPoints(point1, point2){
    return [Math.abs(point1.x - point2.x), Math.abs(point1.y - point2.y)];
}

function isDifferenceSmaller(point1, point2, number){
    let values = getDifferenceBetweenPoints(point1, point2);
    return values[0] <= number && values[1] <= number;
}

function getDistanceBetween2points(point1, point2) {
    return Math.sqrt(
        ((Math.pow(point1.x - point2.x, 2)) + (Math.pow(point1.y - point2.y, 2)))
    );
}
function isPointsTheSame(point1, point2){
    if (point1.x === point2.x && point1.y === point2.y) return true;
}

function splitCurve(parameter, points, precision) {
    let segment1 = [], segment2 = [], matrixDeCasteljau =
initMatrix(points.length);
    for (let i = 0; i < matrixDeCasteljau.length; i++) {
        for (let j = 0; j < matrixDeCasteljau[i].length; j++) {
            if (i === 0) {
                matrixDeCasteljau[i][j] = points[j]
            } else {
                matrixDeCasteljau[i][j] =
getInterimPointOnAdditionalLines(parameter, matrixDeCasteljau[i - 1][j],
matrixDeCasteljau[i - 1][j + 1], precision);
            }
        }
    }
    for (let i = 0; i < matrixDeCasteljau.length; i++) {
        segment1.push(matrixDeCasteljau[i][0]);
        segment2.push(matrixDeCasteljau[i][matrixDeCasteljau[i].length - 1])
    }
    return [segment1, segment2.reverse()];
}

function mergeSegments(parameter, segment1, segment2){
    let p2 = getAdditionalPoint(parameter, segment2.points[2],
segment2.points[3]);
    let p11 = getAdditionalPoint(parameter, segment2.points[1],
segment2.points[2]);
    let p1 = getAdditionalPoint(parameter, p11, p2);
    return new segment('none', segment1.literal, [segment1.points[0], p1, p2,
segment2.points[3]]);
}
```

8) Вихідний код реалізації запуску процесу обробки контейнера

```
document.getElementById('loadSVGBtn').addEventListener('click', function (e) {
    normalizedCurves = [];
    let selectFile = document.getElementById('fileSelect');
    selectFile.click();
    let path = '';
    let container = document.getElementById('objectContainer'),
        fileName = document.getElementById('file-name'),
        fileCurves = document.getElementById('file-curves'),
        bezierCurves = document.getElementById('file-bezier-curves'),
        availableCurves = document.getElementById('file-available-curves'),
        extractedCurves = document.getElementById('file-extracted-curves'),
        recInputSize = document.getElementById('file-inputSize');
    let attributeCurves = [];
    selectFile.addEventListener('change', function () {
        path = selectFile.value;
        let index = path.indexOf('fakepath') + 9;
        container.data = '../images/'.concat(path.slice(index, path.length));
        container.onload = function () {
            fileName.innerHTML = path.slice(index, path.length);
            let svgPath = container.contentDocument.getElementsByTagName('path');
            let bezierCount = 0;
            for (let i = 0; i < svgPath.length; i++) {
                attributeCurves.push(svgPath[i].getAttribute('d'));
                if (isAvailable(attributeCurves[i])) bezierCount++;
            }
            bezierCurves.innerHTML = String(bezierCount);
            copyOfOriginalSVG = container.contentDocument.cloneNode(true);
            if (attributeCurves.length !== 0) {
                let selectedCurves = []; )
                for (let i = 0; i < attributeCurves.length; i++) {
                    selectedCurves.push(selectCurves(attributeCurves[i]));
                }
                let count = 0;
                for (let i = 0; i < selectedCurves.length; i++) {
                    count += selectedCurves[i].length
                }
                fileCurves.innerHTML = String(count);
                for (let i = 0; i < selectedCurves.length; i++) {
                    for (let j = 0; j < selectedCurves[i].length; j++) {
                        for (let k = 0; k < selectedCurves[i][j].segments.length;
k++) {
                            for (let l = 0; l <
selectedCurves[i][j].segments[k].points.length; l++) {
                                selectedCurves[i][j].segments[k].points[l] =
stringPointToPoint(selectedCurves[i][j].segments[k].points[l])
                            }
                        }
                        selectedCurves[i][j].startPoint =
stringPointToPoint(selectedCurves[i][j].startPoint);
                    }
                }
                for (let i = 0; i < selectedCurves.length; i++) {
                    normalizedCurves.push(extractCurves(selectedCurves[i],
curveOrder));
                }
                for (let i = 0; i < normalizedCurves.length; i++) {
                    getCanonicalForm(normalizedCurves[i]);
                }
                for (let i = 0; i < normalizedCurves.length; i++) {
                    checkIfCurveIsPolybezierAndTransform(normalizedCurves[i]);
```

```
                }
                let availableCount = 0;
                for (let i = 0; i < normalizedCurves.length; i++) {
                    for (let j = 0; j < normalizedCurves[i].length; j++) {
                        for (let k = 0; k <
normalizedCurves[i][j].segments.length; k++) {
                             for (let l = 1; l <
normalizedCurves[i][j].segments[k].points.length; l++) {
                            normalizedCurves[i][j].segments[k].valid =
isValid(normalizedCurves[i][j].segments[k].points, minDistance)
                            if (normalizedCurves[i][j].segments[k].valid ===
'yes') availableCount++
                        }
                    }
                }
                availableCurves.innerHTML = String(availableCount)
                extractedCurves.innerHTML =
String(getSegmentsAmount(normalizedCurves));
                recInputSize.innerHTML = String(availableCount * maxBitQuantity)
                return normalizedCurves;
            } else {
                alert('There are no available curves to be used for
encoding/decoding!');
            }
        }
    });
});
```

9) Вихідний код реалізації запуску процесу приховування даних

```
document.getElementById('encodeBtn').addEventListener('click', function () {
    if (document.getElementById('objectContainer').data.match(/images/)) {
        if (normalizedCurves.length !== 0) {
            let progress = document.getElementById('progress');
            progress.max = parseInt(document.getElementById('file-extracted-
curves').innerHTML);
            let percent = 0;
            let input =
convertTextToBinary(document.getElementById('message').innerHTML);
            if (input) {
                if
(document.getElementById('bitMethod').classList.contains('selected')) {
                    input = chunkString(input, maxBitQuantity)
                } else if
(document.getElementById('patternMethod').classList.contains('selected')) {
                    input = chunkString(input, maxBitQuantity)
                }
                console.time('encode')
                let finalParameters = [];
                let encodedCurves = [];
                let patternTable = initTableOfPatterns(patternLength,
maxBitQuantity)
                for (let i = 0, index = 0; i < normalizedCurves.length; i++) {
                    let newSegments = [];
                    for (let j = 0; j < normalizedCurves[i].length; j++) {
                        let tmpSegments = [];
                        for (let k = 0; k <
normalizedCurves[i][j].segments.length; k++) {
                            if
(document.getElementById('bitMethod').classList.contains('selected')) {

tmpSegments.push(encodeBit(normalizedCurves[i][j].segments[k], input[index],
deltaStep, precision));
```

```
                            } else {
                tmpSegments.push(encodePattern(normalizedCurves[i][j].segments[k],
input[index], precision, startStep, patternLength, maxBitQuantity, patternTable));
                            }
                            if (tmpSegments[k].finalParameter !== null) {
                                index++;
                            }
                            updateProgress(++percent, progress)
                        }
                        newSegments.push(tmpSegments)
                    }
                    encodedCurves.push(newSegments)
                }
                console.timeEnd('encode')
                let encodedCurvesExtractedData = [];
                for (let i = 0; i < encodedCurves.length; i++) {
                    let curve = [];
                    for (let j = 0; j < encodedCurves[i].length; j++) {
                        let segments = [], param = [];
                        for (let k = 0; k < encodedCurves[i][j].length; k++) {
                            segments.push(encodedCurves[i][j][k].encoded);
                            param.push(encodedCurves[i][j][k].finalParameter)
                        }
                        curve.push(segments);
                        finalParameters.push(param);
                    }
                    encodedCurvesExtractedData.push(curve);
                }
                patternSteganoKeys = {
                    patternTable: patternTable,
                    finalParameters: finalParameters
                };
                let encodedCurvesToString = [];
                for (let i = 0; i < encodedCurvesExtractedData.length; i++) {
                    let tmpSegments = [];
                    for (let j = 0; j < encodedCurvesExtractedData[i].length; j++)
{
                        if (j === 0) {
                            tmpSegments.push(new
Curve(normalizedCurves[i][j].literal, normalizedCurves[i][j].startPoint,
encodedCurvesExtractedData[i][j]));
                        } else {
                            tmpSegments.push(new Curve('m', new Point(0, 0),
encodedCurvesExtractedData[i][j]));
                        }
                    }
                    encodedCurvesToString.push(tmpSegments)
                }
                let path = []
                for (let i = 0; i < encodedCurvesToString.length; i++) {
                    path.push(arrayOfCurvesToString(encodedCurvesToString[i]))
                }
                let copyPath = copyOfOriginalSVG.getElementsByTagName('path');
                for (let i = 0; i < copyPath.length; i++) {
                    copyPath[i].setAttribute('d', path[i]);
                }
                showEncryptionEnd();
                if
(document.getElementById('patternMethod').classList.contains('selected')) {
                    let tempDOMObject = document.createElement('p');
                    tempDOMObject.classList.add('noDisplay');
                    tempDOMObject.innerHTML = JSON.stringify(patternSteganoKeys);
```

```
                    let svgName = document.getElementById('file-name').innerHTML;
                    let name = svgName.slice(0, svgName.length -
4).concat('.txt');
                    saveSteganoKeys(tempDOMObject,
                        ('patternKeysFor_'.concat(name)));
                } else if
(document.getElementById('bitMethod').classList.contains('selected')) {
                    let tempDOMObject = document.createElement('p');
                    tempDOMObject.classList.add('noDisplay');
                    tempDOMObject.innerHTML = JSON.stringify(patternSteganoKeys);
                    let svgName = document.getElementById('file-name').innerHTML;
                    let name = svgName.slice(0, svgName.length -
4).concat('.txt');
                    saveSteganoKeys(tempDOMObject,
                        ('bitKeysFor_'.concat(name)));
                }
            } else {
                alert('A message is not entered!')
            }

        } else {
            alert('There is no curves for encoding')
        }
    } else {
        alert('Load SVG image first!')
    }
});
```

10) Вихідний код реалізації запуску процесу вилучення даних

```
document.getElementById('decodeBtn').addEventListener('click', function () {
    if (normalizedCurves.length !== 0) {
        let decodedMessage = [];
        let progress = document.getElementById('progress');
        progress.max = parseInt(document.getElementById('file-extracted-
curves').innerHTML);
        let percent = 0;
        if
(document.getElementById('patternMethod').classList.contains('selected')) {
            let patternSteganoKeyForDecoding = '';
            let selectedFile = document.getElementById('fileSelectKeys');
            selectedFile.click();
            let path = '';
            selectedFile.addEventListener('change', function () {
                path = selectedFile.value;
                let index = path.indexOf('fakepath') + 9;
                let request = new XMLHttpRequest();
                request.open('GET', '../steganoKeys/'.concat(path.slice(index,
path.length)), true);
                request.onreadystatechange = function () {
                    if (request.readyState === 4 && request.status === 200) {
                        let keysHolder = document.createElement('p');
                        keysHolder.classList.add('noDisplay');
                        keysHolder.innerHTML = request.responseText;
                     Array.from(keysHolder.querySelectorAll("p")).forEach(function
(p) {
                            patternSteganoKeyForDecoding = JSON.parse(p.innerHTML)
                        });
                        console.time('decodePattern')
                        for (let i = 0; i < normalizedCurves.length; i++) {
                            for (let j = 0; j < normalizedCurves[i].length; j++) {
                                if
(!(normalizedCurves[i][j].segments[0].literal.match(/[Ll]/))) {
```

```
                                    decodedMessage.push(decodePattern(maxBitQuantity,
patternAccuracy, normalizedCurves[i][j].segments, startStep, precision,
patternSteganoKeyForDecoding.finalParameters[i][j],
patternSteganoKeyForDecoding.patternTable));
                                }
                                updateProgress(++percent, progress)
                            }
                        }
                        console.timeEnd('decodePattern')
                        document.getElementById('decodedMessage').innerHTML =
convertBinaryToText(decodedMessage.join(''));
                    }
                }
                request.send();
            });
        } else {
            if
(document.getElementById('bitMethod').classList.contains('selected')) {
                let bitSteganoKeyForDecoding = '';
                let selectedFile = document.getElementById('fileSelectKeys');
                selectedFile.click();
                let path = '';
                selectedFile.addEventListener('change', function () {
                    path = selectedFile.value;
                    let index = path.indexOf('fakepath') + 9;
                    let request = new XMLHttpRequest();
                    request.open('GET', '../steganoKeys/'.concat(path.slice(index,
path.length)), true);
                    request.onreadystatechange = function () {
                        if (request.readyState === 4 && request.status === 200) {
                            let keysHolder = document.createElement('p');
                            keysHolder.classList.add('noDisplay');
                            keysHolder.innerHTML = request.responseText;
                         Array.from(keysHolder.querySelectorAll("p")).forEach(function
(p) {
                                bitSteganoKeyForDecoding = JSON.parse(p.innerHTML)
                            });
                            console.time('decodeBit')
                            for (let i = 0; i < normalizedCurves.length; i++) {
                                for (let j = 0; j < normalizedCurves[i].length;
j++) {
                                    if
(!(normalizedCurves[i][j].segments[0].literal.match(/[Ll]/))) {
                                   decodedMessage.push(decodeBit(maxBitQuantity,
accuracy, normalizedCurves[i][j].segments, deltaStep, precision,
bitSteganoKeyForDecoding.finalParameters[i][j]));
                                    }
                                    updateProgress(++percent, progress)
                                }
                            }
                            console.timeEnd('decodeBit')

                        document.getElementById('decodedMessage').innerHTML =
convertBinaryToText(decodedMessage.join(''));
                        }
                    }
                    request.send();
                });
            }
        }
    }
});
```