Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів оптимального кодування та стиснення даних
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-2

Мичка С. О.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Голян Н. В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
студента Мички Святослава Олеговича
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження методів оптимального кодування і стиснення даних».

Затверджена наказом по університету від 29.03.2024р. № 250 Ст

2. . Термін подання студентом роботи до екзаменаційної комісії 20.06.2024

3. Вихідні дані до роботи електронні ресурси за обраною тематикою, стандарти реалізації алгоритмів, середовища розробки Visual Studio 2022 Community та Visual Studio Code, бібліотеки SharpZipLib, OpenXML, база даних MSSQL, технології .NET 6, C#, ASP.NET, React

4. Перелік питань, що потрібно опрацювати в роботі актуальність області дослідження, мета, постановка задачі, аналіз проблемної області, аналіз методів стиснення, планування експерименту

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	23.01 – 14.02.24	*виконано*
2	Аналіз та вибір алгоритмів для дослідження	15.02 – 20.02.24	*виконано*
3	Аналіз та моделювання предметної області	17.02 – 28.02.24	*виконано*
4	Планування експериментів	25.02 – 28.02.24	*виконано*
5	Створення програмної реалізації експериментів	28.02 – 01.04.24	*виконано*
6	Експериментальні дослідження	02.04 – 20.04.24	*виконано*
7	Аналіз результатів експериментальних досліджень	20.04 – 23.04.24	*виконано*
8	Розробка програмної реалізації демонстрації результатів експериментів	23.04 – 04.05.24	*виконано*
9	Підготовка пояснювальної записки	04.05 – 31.05.24	*виконано*
10	Підготовка презентації та доповіді	01.06 – 06.06.24	*виконано*
11	Нормоконтроль	07.06 – 11.06.24	*виконано*
12	Рецензування	11.06 – 16.06.24	*виконано*
13	Занесення диплома в електронний архів	18.06.2024	*виконано*
14	Попередній захист	18.06.2024	*виконано*
15	Допуск до захисту у зав. кафедри	19.06.2024	*виконано*

Дата видачі завдання 22 січня 2024р

Студент _____ Мичка С. О.
(підпис)

Керівник кваліфікаційної роботи _____ к.т.н., доц. каф. ПІ Голян Н. В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить: 88 с., 69 рис., 2 табл., 19 джерел.

СТИСНЕННЯ, ОПТИМАЛЬНЕ КОДУВАННЯ, АЛГОРИТМ, ТЕОРІЯ ІНФОРМАЦІЇ, ЕНТРОПІЯ, C#, ASP .NET

Об’єктом дослідження є методи та алгоритми стиснення даних різних типів.

Метою дослідження є визначити найбільш оптимальні алгоритми стиснення для певних даних. В ньому необхідно вирішити наступні завдання:

- розглянути методи оптимального кодування даних, що існують і використовуються для стиснення даних без втрат, зокрема текстових;
- порівняти методи стиснення даних, означити їхні сильні й слабкі сторони, порівняти їх із іншими методами, представленими для досліджень;
- розробити програмну систему для підбору методу стиснення для користувацьких даних, використовуючи дані, отримані з дослідження.

Предметом дослідження виступатимуть бібліотеки та інші програмні рішення для алгоритмів і методів стиснення даних, що розглядаються. Основою методології дослідження слугує теорія про стиснення даних шляхом зменшення надлишковості, яку вони представляють.

В дослідженні застосовуються дані про ентропію, надлишковість, основи зберігання даних, застосовуються методи теоретичного моделювання. В результаті виконання дослідження маємо отримати порівняльну характеристику розглянутих алгоритмів із даними про їхню роботу з даними певних видів.

Маємо отримати висновки про доцільність їхнього застосування з даними.

COMPRESSION, OPTIMAL ENCODING, ALGORITHM, INFORMATION THEORY, ENTROPY, C#, ASP .NET

The object of research is methods and algorithms of data compression of various types.

The research method is the most optimal embossing algorithms for certain data. In the research, the solutions for following problems are required:

- research methods of optimal data encoding, which are used and used for lossless data compression, in particular text data;
- compare data compression methods, identify their strengths and weaknesses, compare them with other methods presented for research;
- develop a software system for selecting a method of using data, using the data obtained from the research.

The subject of the study are the libraries and other software solutions for the algorithms and data compression methods under consideration.

The main research methodology is the theory of data compression by reducing the redundancy they represent. The study uses data on entropy, redundancy, the basics of data storage, and uses theoretical modeling methods.

As a result of the research, we should obtain a comparative characteristic of the presented algorithms with data on their work with data of certain types. We must obtain conclusions about the expediency of their application with the data.

Я, Мичка Святослав Олегович, студент групи ІПЗм-22-2, здобувач вищої освіти на другому (магістерському) рівні, кафедра «Програмна інженерія», заявляю: представлена робота на тему «Дослідження методів оптимального кодування та стиснення даних» виконана мною самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

У сучасному світі кількість нових даних невпинно зростає. За дослідженням Каліфорнійського університету в Берклі, щороку об’єм даних, що створюються людьми, зростає на 30%. Відповідно, затрати на зберігання й транспортування таких об’ємів даних теж зростають.

Актуальність розробки та поліпшення алгоритмів стиснення даних полягає, насамперед, у тому, що сильніше стиснення даних із можливістю точно (за використання алгоритмів стиснення без втрат) відновити початкову інформацію дозволяє зменшити витрати на їх зберігання, транспортування й обробку. Тому існують змагання, наприклад, Hutter Prize, призові фонди яких сягають кількох тисяч євро.

Сучасні алгоритми стиснення даних включають такі його види, як ентропійне (частотне), словникове, змішування контексту тощо. Часто алгоритми доповнюють один одного, дозволяючи значно покращити коефіцієнти стиснення. Одним із відомих таких прикладів є Deflate, що включає в себе два інших – LZ77 і оптимальний код Хафмана.

Метою роботи є дослідження алгоритмів і порівняння характеристик їхньої роботи, знаходження алгоритмів стиснення, що працюють краще на певних видах даних. Об’єктом дослідження в даній роботі виступають алгоритми стиснення без втрат, предмет дослідження – аналіз роботи алгоритмів із даними різного виду (текст, зображення, відео) та розміру.

Отримані результати дослідження будуть використані для визначення найефективнішого методу стиснення даних залежно від вхідних даних. Також будуть запропоновані рекомендації щодо підбору оптимальних методів та можливих напрямків подальших досліджень у цій галузі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Алгоритми стиснення даних

Алгоритм стиснення (компресії) даних – це таке кодування даних, яке забезпечує зменшення об’єму, що вони займають. Зазвичай такі перетворення є оборотними, тобто мають обернену дію, яку називають відновленням або декомпресією, яка полягає в частковому або повному відновленні вихідних даних. Стиснення даних базується на зменшенні їхньої надлишковості, так, ентропійне кодування кодує символи, що зустрічаються в даних частіше, більш короткими послідовностями, що зазвичай менші за 8 бітів.

Основною характеристикою алгоритмів стиснення даних є коефіцієнт стиснення – відношення розміру початкових даних до розміру даних після роботи алгоритму. Якщо алгоритм працює з коефіцієнтом 1, то він не виконує корисної роботи, оскільки об’єм даних не зменшується, тобто не виконується, власне, стиснення. Таку властивість мають такі алгоритми, як перетворення Барроуза-Уіллера, який не є алгоритмом стиснення, проте може використовуватись як проміжний етап підготовки даних для інших алгоритмів. Якщо алгоритм працює із коефіцієнтом, меншим за 1, то він виконує «шкідливу» роботу, тобто замість стиснення фактично збільшує об’єм вхідних даних.

Часом алгоритми характеризують і за іншими характеристиками: час кодування та декодування, максимум використання оперативної пам’яті тощо. Однак ці дані можуть відрізнятись і залежать від апаратної частини більше, ніж від реалізації алгоритму, тому вони не відіграють основної ролі в оцінці роботи алгоритмів стиснення в таких бенчмарках, як Large Text Compression Benchmark [1].

Алгоритми стиснення даних поділяють на дві підгрупи: з утратами та без втрат. Алгоритми з втратами зазвичай дають більші коефіцієнти стиснення, але при цьому початкові дані після роботи алгоритмів відновити неможливо. Зазвичай алгоритми стиснення з втратами використовуються для стискання специфічних даних, зокрема звукових, відео або зображень, причому алгоритми, пристосовані для архівації, наприклад, звукових даних, зазвичай не працюють із іншими,

наприклад, із зображеннями. Такі алгоритми зменшують надлишковість із такими втратами, які або непомітні, або малопомітні для людських органів чуття. Так, людина, що прослуховує музичний трек на непрофесійних пристроях, наприклад, у навушниках або за допомогою динаміка телефону, не відчує різниці між однією й тою ж композицією в форматі MP3, що допускає втрати, та FLAC, що стискає аудіофайли без втрат [2].

Алгоритми стиснення без втрат працюють так, що вихідні дані можна відновити після кодування в повному обсязі. Зазвичай вони зменшують надлишковість даних за допомогою кодування окремих символів або їхніх послідовностей, тому можуть застосовуватись для кодування різних форматів даних, особливо таких, для яких важливо точно зберегти оригінальну послідовність байтів. Такими даними можуть бути текстові або виконувані файли, оскільки якщо такі втрати матимуть втрати, то коректне їх сприйняття читачем або машиною гарантувати буде неможливо. Головним недоліком алгоритмів стиснення без втрат є те, що вони часто дають менші коефіцієнти стиснення, ніж алгоритми з втратами.

Для будь-яких алгоритмів однією з основних вимог є швидкість стиснення та якнайменше використання оперативної пам’яті, оскільки часто дані необхідно стискати паралельно з іншою логікою. Прикладом може бути стиснення даних, що використовуються для навчання моделей пошуку об’єктів на зображеннях [3].

1.2 Постановка задачі

Виходячи з аналізу предметної області, можна поставити такі задачі для вирішення в ході проведення магістерського дослідження:

- дослідити алгоритми стиснення без втрат даних різних видів та , зокрема текстових, відео, статичних зображень тощо, та їхні поєднання;
- визначити критерії порівняння алгоритмів, що досліджуються, та використати їх для порівняння ефективності застосування методів стиснення до певних видів файлів;

– використовуючи отримані дані, розробити додаток, що підбиратиме необхідний алгоритм стиснення для даних, наданих користувачем;
– провести ряд експериментів для оцінки ефективності роботи розробленого додатку.

Цей комплекс поставлених задач має на меті покращити розуміння методів стиснення даних. Під час їх вирішення та проведення додаткових експериментів будуть порівняні різні алгоритми та їх комбінації, що дасть змогу оцінити корисність як більш нових, так і старіших методів стиснення даних.

1.3 Огляд алгоритмів для дослідження

В рамках дослідження планується розглянути алгоритми стиснення без втрат та порівняти їхню роботу з різними форматами даних, враховуючи обмеження алгоритмів. Розглянемо наступні алгоритми:

– LZ77 – один із основних словникових алгоритмів стиснення без втрат, що лежить в основі інших алгоритмів, зокрема Deflate [4];
– оптимальний код Хафмана – алгоритм ентропійного кодування, що заснований на кодування символів, що зустрічаються в повідомленні частіше, меншою кількістю бітів. Він, як і LZ77, лежить в основі алгоритму Deflate, а також інших алгоритмів;
– Deflate – алгоритм, що застосовується в таких форматах, як ZIP, PNG та GIF. Складається з двох вищезгаданих алгоритмів – LZ77 та оптимального кодування Хафмана;
– Brotli – алгоритм стиснення, розроблений Google, який використовується для стискання веб-ресурсів;
– Run-Length Encoding (RLE) – простий алгоритм стиснення без втрат, який замінює повторювані послідовності символів одним символом і числом повторень;
– Prediction by Partial Matching (PPM) – статистичний алгоритм, заснований на моделюванні й передбаченні контексту [5].

Дані алгоритми є основними з тих, що використовуються для стиснення даних у наш час. Вони можуть застосовуватись для різних цілей, так, RLE добре підходить для звукових даних та відео, попередньо оброблених за допомогою дельта-кодування, тобто представлення кожного наступного кадру як суму змін попереднього. PPM дає кращі результати при застосуванні до текстових даних природними мовами тощо.

1.3.1 LZ77

LZ77 – це алгоритм стиснення без втрат, який відноситься до сімейства LZ (від Лемпель-Зів, прізвищ розробників), що також включає інші алгоритми, зокрема LZ78, LZW та LZMA. Він відноситься до словникових методів стиснення, використовуючи «ковзне вікно» – буфер, що зберігає інформацію про послідовності символів, що вже зустрічались у повідомленні.

1.3.2. LZW

Алгоритм LZW – ще один алгоритм, що відноситься до словникових методів сімейства LZ. Замість «ковзного вікна» він використовує словник, що містить послідовності символів, що раніше зустрічались у тексті.

Процес кодування в алгоритмі LZW починається з початкового словника, що містить всі можливі одиночні символи вхідного алфавіту. Під час обробки вхідних даних алгоритм поступово додає нові комбінації символів до словника, коли виявляє нові повторювані послідовності. Кожна така послідовність представлена унікальним кодом, який замінює її в стиснених даних.

1.3.3 Алгоритм RLE

Run-length encoding (кодування довжин серій), або RLE – це алгоритм стиснення даних, суть якого полягає в кодуванні символів, що повторюються, за допомогою одного символа й кількості його повторів.

Властивістю кодувати повторювані символи він схожий на LZ77, розглянутий вище, але останній кодує не лише повтори символів, а й окремих їхніх комбінацій.

1.3.4 BWT

Перетворення Барроуза-Вілера (BWT) не є алгоритмом стиснення, проте воно дозволяє видозмінити вхідний потік так, щоб він містив символи, що повторюються, у великих кількостях. Це дозволяє зробити такі алгоритми, як RLE, що розглядається зараз, більш ефективними.

1.3.5 MTF

Алгоритм Move to front (MTF), як і BWT, не є алгоритмом стиснення, натомість він перетворює деякий набір символів у послідовність чисел. Він не використовує матриці ротацій, на відміну від BWT, що робить його швидшим, проте його ефективність часто нижча за перетворення Барроуза-Вілера.

1.3.6 Оптимальне кодування Хафмана

Оптимальне кодування Хафмана – це ентропійний алгоритм стиснення даних, що був розроблений Девідом Хафманом як продовження старшого алгоритму Шеннона-Фано. Він заснований на використанні префіксного дерева кодів символів, тобто жоден із кодів не зустрічається в жодному іншому коді, що дозволяє однозначно визначати, який саме символ було закодовано в дереві.

Таким чином надлишковість зменшується за рахунок зменшення кількості бітів, що необхідні для кодування одного символа (фактично – байту).

Алгоритм влаштовано таким чином, що символи, що зустрічаються частіше, мають коротші префіксні коди, і навпаки – символи з меншою частотою появи в тексті кодуються більш довгими послідовностями.

Існує два варіанти алгоритму: адаптивний і неадаптивний. Адаптивний алгоритм не потребує передачі в повідомленні даних про структуру префіксного

дерева, проте не гарантує найбільшого можливого стиснення, на відміну від неадаптивного.

1.3.7 Алгоритм Deflate

Алгоритм стиснення Deflate – це поєднання двох алгоритмів, описаних вище: LZ77 та оптимального кодування Хафмана. Хоча алгоритм Deflate використовується в багатьох форматах, для стиснення природного тексту часом підходять більш сучасні алгоритми, оскільки ні кодування Хафмана, ні LZ77 не враховують контекст і не виконують моделювання саме природніх мов.

1.3.8 BZip2

Алгоритм стиснення BZip2 використовує послідовність алгоритмів BWT, MTF та кодування Хафмана. Під час кодування дані розбиваються на блоки фіксованої довжини, від 100 до 900 Кб.

1.3.9 Алгоритм Brotli

Brotli – це новий алгоритм, розроблений у 2013 році Google як заміна загальноприйнятого Gzip для стиснення веб-ресурсів. Незважаючи на те, що Brotli, як і Gzip, використовує LZ77 і код Хафмана, його основною перевагою є наявність словника: Brotli містить 120-кілобайтний словник і динамічний модуль для пришвидшення передавання даних [6].

1.3.10 Алгоритм PPM

Алгоритм Prediction by Partial Matching (PPM) – це алгоритм, суть якого зводиться до передбачення певного символу виходячи з *n* попередніх. Інша назва цього алгоритму – передбачення за моделлю Маркова.

Алгоритм PPM працює шляхом аналізу контексту – послідовностей символів, що передують поточному символу, і побудови ймовірнісної моделі на основі цих послідовностей. Коли алгоритм стикається з новим символом, він використовує збережені контексти для передбачення ймовірностей появи цього

символу. Якщо контекстів недостатньо для точного передбачення, алгоритм поступово зменшує довжину контексту, доки не знайде відповідний контекст у своїй моделі.

Реалізації алгоритму PPM, зокрема PPMd, PPMb тощо, використовуються в таких архіваторах, як 7-Zip і WinZip, причому сама модель PPM не виконує безпосереднього стиснення, а лише передбачає значення символа залежно від попередніх.

2 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

2.1 Опис алгоритмів стиснення даних

До алгоритмів стиснення без втрат, на відміну від тих, що допускають втрати, основною вимогою є можливість повністю відновити дані, що були стиснуті. Відповідно, кожен алгоритм має власний механізм забезпечення повної зворотності дій зі стиснення.

2.1.1 LZ77

Стандартний алгоритм LZ77 кодує повторювані послідовності в словнику за допомогою двох чисел – відступу (offset) і довжини збігу (length of match). Таким чином, під час проходу повідомленням, для кожної послідовності символів $x[i]..x[j]$, де i, j – індекси символів повідомлення, починаючи від 0, перевіряється наявність цих послідовностей у буфері.

При знаходженні збігу, перевіряється наявність послідовності $x[i]..x[j+1]$, і так до того моменту, коли для чергового j для $x[i]..x[j+1]$ не буде знайдено збігу.

Коли збіг не знайдено, знаходиться відступ c від кінцевого індексу ковзного вікна-буфера до початку найближчої послідовності, що збігається з $x[i]..x[j]$, та довжина збігу l. Після цього в кінець буферу записується послідовність $x[i]..x[j+1]$, а до списку збігів записуємо три значення: c, l і окремий символ $x[j]$.

Якщо збігу не знайдено, значення c, l вважаємо 0, і, оскільки така ситуація можлива тільки якщо послідовність $x[i]..x[j]$ складається з одного символа, тобто $i = j$, записуємо цей символ в кінець буфера.

Алгоритм повторюється циклічно, доки не скінчаться елементи повідомлення. Варто зауважити, що якщо частина послідовності збігається з послідовністю в буфері, а інша частина присутня на початку повідомлення, що розглядається, її дозволяється не розривати, оскільки при декодуванні розірвана послідовність не відрізнятиметься від нерозривної.

Декодування (відновлення) даних, стиснених алгоритмом LZ77, відбувається в зворотному порядку. Для кожного значення (c, l, $x[j]$) записуємо в буфер послідовність, що починається на c-му елементі буфера з кінця й продовжується l символів, після чого дописуємо в кінець буфера символ $x[j]$. Якщо $l > c$, то, за визначенням, послідовність продовжується циклічно вже з урахуванням щойно доданих символів до буфера.

Нехай маємо рядок символів “abcabdabdabf”. Візьмемо довжину буфера ковзного вікна 5 і стиснемо повідомлення за допомогою алгоритма LZ77.

Для першого символа, оскільки буфер не має жодного значення, збігів немає, отже, запишемо (0, 0, “a”) та “a” до буфера. Аналогічно повторюємо для другого та третього символів і отримуємо в відповіді (0, 0, “a”), (0, 0, “b”), (0, 0, “c”), а в буфері – (“a”, “b”, “c”).

Для четвертого й п’ятого символів у буфері маємо співпадіння “ab”, отже, запишемо до відповіді (3, 2, “d”), оскільки збіг починається на третій позиції з кінця ковзного вікна й продовжується два символи. Додавши символ “d” отримуємо відсутність збігів, отже, він має йти після “ab”. Після цих дій буфер має вигляд (“a”, “b”, “c”, “a”, “b”, “d”).

Для сьомого символу маємо збіг (3, 3, “a”), але він може бути продовжений циклічно до (3, 5, “f”), тобто “a” і “b” можна взяти циклічно з буфера так, що послідовність збігу набуває виду “abdab”. Цей збіг є останнім для повідомлення, оскільки “f” – останній символ у ньому.

Таким чином, маємо відповідь – {(0, 0, “a”), (0, 0, “b”), (0, 0, “c”), (3, 2, “d”), (3, 5, “f”)}. До результату вона записується за допомогою послідовностей байтів. Так, позиція в буфері зазвичай має тип *ushort*, тобто беззнаковий формат розміром 2 байти. Після цього записують ще два байти для довжини збігу в буфері та, власне, символу. Таким чином кожен запис може бути представлений за допомогою чотирьох байтів, які фактично кодують число типу *int*.

На рисунку 2.1 зображено процес кодування поетапно.

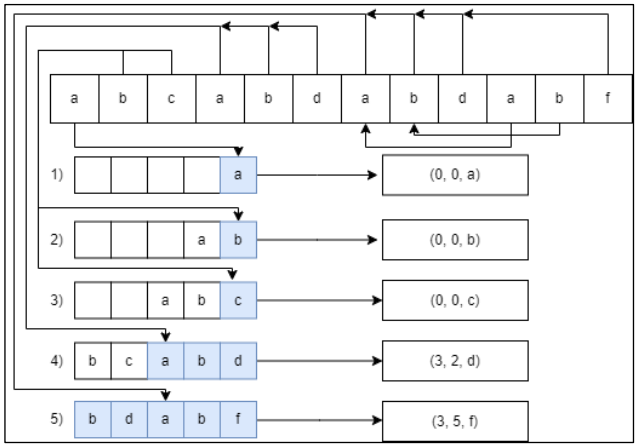


Рисунок 2.1 – Процес кодування LZ77 (рисунок виконано самостійно)

Декодування повідомлення відбувається у зворотному порядку, але з використанням одного спільного буфера. Три перших символи записуємо без змін, оскільки вони мають c і l рівні 0. Для четвертого збігу необхідно взяти третій з кінця буфера символ і другий після нього, додавши “d” в кінець. Таким чином отримуємо “abcabd”. Для п’ятого збігу необхідно з третього з кінця символа взяти три символи, а четвертий і п’ятий взяти циклічно звідти ж, після чого дописати “f”. Таким чином отримуємо “abcabdabdabf”, що й було вхідним повідомленням. На рисунку 2.2 зображено процес декодування поетапно.

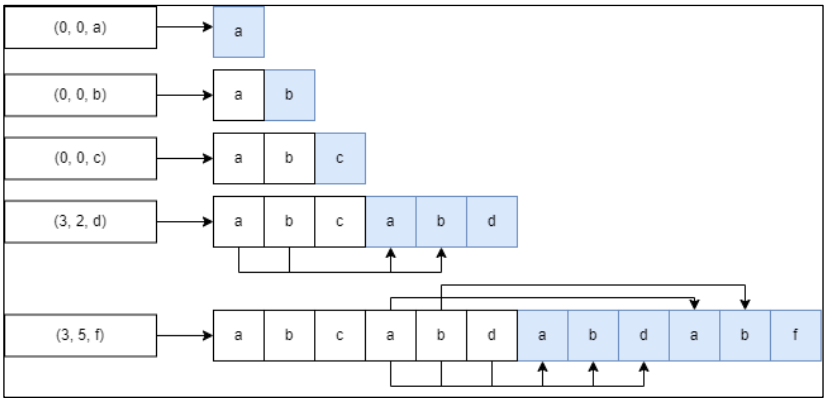


Рисунок 2.2 – Процес декодування LZ77 (рисунок виконано самостійно)

Очевидно, алгоритм LZ77 має недоліки. З прикладу видно, що, хоча початкове повідомлення було закодоване 12-ма байтами, «стиснені» дані займають вже 15 байтів. Це відбувається, зокрема, через те, що більше половини збігів закодовані одиничними символами, на які витрачається втричі більше пам’яті

через необхідність зберігати значення c і l, a, отже, вони займають доволі велику частину стисненого файлу. Відповідно, для зменшення такого ефекту необхідне повідомлення достатньо великого розміру.

Необхідно також зауважити, що кодуватися можуть лише такі послідовності, що не перевищують за довжиною розмір ковзного вікна. В реальних архіваторах розмір ковзного вікна може сягати кількох мегабайтів, що збільшує використання оперативної пам'яті, проте дає змогу закодувати більші об'єми даних. Схожа проблема, пов'язана з використанням ковзного вікна – це неможливість кодування підрядків, що розташовані один від одного на відстані, більшій за довжину буфера.

2.1.2 LZW

Алгоритм LZW використовує один словник як для стиснення, так і для відновлення, тому він не додається до закодованих даних. Спочатку до нього записуються всі можливі символи по одному. Після цього зчитується перший символ повідомлення та ініціалізується як початкова фраза W. Також циклічно зчитується наступний символ K після W, і, якщо фраза WK присутня у словнику, фраза W прирівнюється до WK (тобто конкатенації рядка із попередньої фрази W та символу K). Цикл повторюється доти, доки не буде знайдено таких WK, що не міститься у словнику, при цьому W прирівнюється до K, до словника додається фраза WK, а до результату записується код для фрази W.

Нехай маємо деякий рядок «ANANASBANANA». Для його кодування використаємо словник із 26 літер латиниці та символу кінця рядка, який позначимо «#». Для їх стиснення необхідно проініціалізувати словник усіма однолітерними фразами, пронумерувавши їх від 0 (символ кінця рядка) до 26 (літера Z). Зчитуючи символи один за одним, додаємо до словника рядки AN і NA, оскільки вони відсутні в ньому, і нумеруємо їх відповідно 27 і 28. Зчитуючи ці фрази далі в тексті, кодуємо її згідно з їхніми кодами у словнику. На рисунку 2.3 зображено діаграму із описом процесу стиснення рядка.

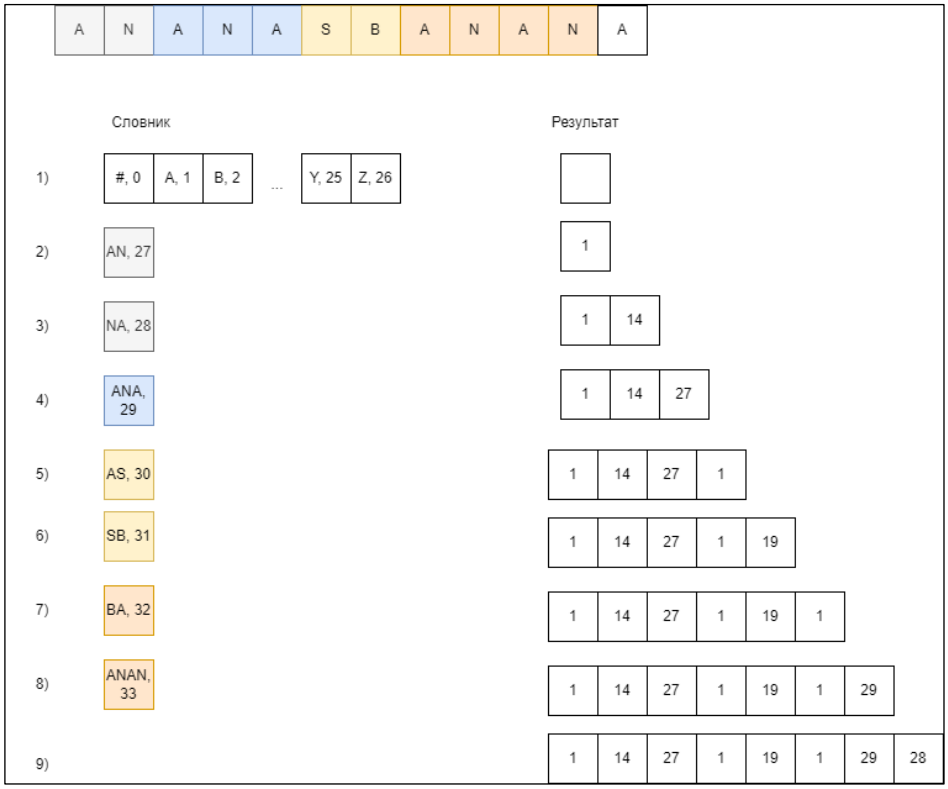


Рисунок 2.3 – Стиснення LZW (рисунок виконано самостійно)

Можна побачити, що стиснення покращується зі збільшенням довжини рядка, оскільки в словнику з’являються нові, довші послідовності. Так, якщо вхідний рядок був представлений 12 символами, то на виході алгоритму маємо лише 8, отже, коефіцієнт стиснення дорівнює $\frac{12}{8} = 1{,}5$. Можна помітити, що зі збільшенням довжини вхідного рядку збільшується й об’єм словника, отже, на певному етапі кількість бітів на кодування фрази теж збільшиться. Деякі реалізації включають також очищення фраз, які мало використовуються, звільняючи коди для нових фраз.

Для відновлення даних також необхідно спочатку ініціалізувати словник та послідовно йти рядком стиснених даних, також заносячи нові послідовності до словника так, як і під час стиснення.

На рисунку 2.4 зображено процес відновлення раніше закодованої фрази.

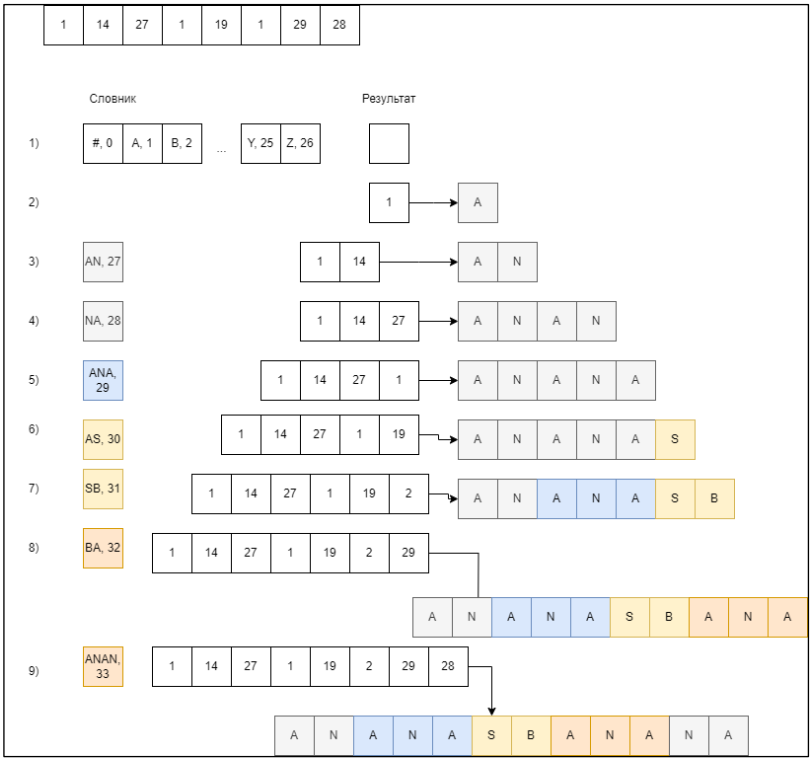


Рисунок 2.4 – Відновлення даних алгоритмом LZW (рисунок виконано самостійно)

Стиснення LZW застосовується для стиснення зображень, включаючи анімовані (GIF, TIFF). Оскільки LZW зчитує символи один за одним, жадібно записуючи до словника, він іноді є швидшим за LZ77, проте може давати гірші коефіцієнти стиснення.

2.1.3 RLE

Алгоритм кодування довжин серій (Run-length encoding, RLE) кодує повторювані символи за допомогою додатних чисел, а символи, що не повторюються – від'ємних. Закодований рядок виглядатиме наступним чином: "-1B2N-1.2A-2&A", де числа показують, скільки символів займає послідовність. Можна побачити, що таке кодування часто призводить лише до зростання обсягу файлу, так, вихідний файл мав 8 байтів інформації, а оброблений стисненням RLE – 11, за умови використання знакових байтів. Крім того, виходить, що за використання типу *sbyte* максимальна довжина послідовності, яку можна закодувати – 128 повторюваних або унікальних символів підряд. Саме тому в

чистому вигляді RLE майже не використовується в сучасних архіваторах, розробники яких надають перевагу LZ-алгоритмам, два з яких були описані вище [8]. Однак його можна використовувати як додатковий засіб для отримання більших коефіцієнтів стиснення.

2.1.4 BWT

Для того, аби перетворити повідомлення за допомогою BWT (перетворення Барроуза-Вілера), необхідно отримати список усіх циклічних перестановок рядка й відсортувати стовпці отриманої матриці, взявши останній [9]. Так, нехай маємо слово «.BANANA&», де "&" - EOL-символ, яке необхідно перетворити за алгоритмом Барроуза-Вілера. Тоді отримуємо таку матрицю циклічних перестановок:

$$
\begin{array}{cccccccc}
. & B & A & N & A & N & A & \& \\
\& & . & B & A & N & A & N & A \\
A & \& & . & B & A & N & A & N \\
N & A & \& & . & B & A & N & A \\
A & N & A & \& & . & B & A & N \\
N & A & N & A & \& & . & B & A \\
A & N & A & N & A & \& & . & B \\
B & A & N & A & N & A & \& & .
\end{array}
$$

Відсортувавши її, отримаємо

$$
\begin{array}{cccccccc}
A & N & A & N & A & \& & . & B \\
A & N & A & \& & . & B & A & N \\
A & \& & . & B & A & N & A & N \\
B & A & N & A & N & A & \& & . \\
N & A & N & A & \& & . & B & A \\
N & A & \& & . & B & A & N & A \\
. & B & A & N & A & N & A & \& \\
\& & . & B & A & N & A & N & A
\end{array}
$$

Узявши останній її стовпець, отримаємо "BNN.AA&A", що й є шуканим рядком. Для того, аби з цього рядку отримати вихідний, необхідно додавати до матриці по одному стовпцю, сортуючи її рядки. Коли матриця буде повністю заповнена, необхідно знайти рядок з EOL-символом наприкінці, що й буде шуканим рядком. Однак таке правило діятиме лише тоді, коли ми маємо унікальний символ кінця рядка, що не повторюється більше ніде в повідомленні.

2.1.5 MTF

Алгоритм Move-to-front, або MTF, кодує вхідний рядок, перетворюючи його на послідовність чисел. На початку створюється словник із усіх можливих числових значень символів, який оновлюється після кодування кожного символу так, що отриманий символ стає першим у ньому, посуваючи інші на одну позицію [10].

Нехай маємо деякий рядок «ABACBAC». Використовуючи англійський алфавіт у якості словника, можна перетворити даний рядок у послідовність символів {0, 1, 1, 2, 2, 2, 2}. Бачимо, що, хоча повтори символів у даному рядку були відсутні, вони присутні в результаті перетворення. Таким чином, цей алгоритм можна використовувати як попередню обробку вхідних даних перед застосуванням словникових алгоритмів.

Відновлення даних відбувається в тій же послідовності: зчитуючи «0» записуємо перший символ у словнику «A», після чого зчитуємо «1» і записуємо «B», зсуваючи його наперед тощо. Таким чином отримуємо рядок до перетворення.

Алгоритм MTF є доволі швидким порівняно з BWT, оскільки не виконує сортування рядків у матриці зсувів. Проте він менш гарантовано дає повторювані послідовності, оскільки бере до уваги лише поточний символ і стан словника.

2.1.6 Оптимальне кодування Хафмана

Алгоритм Хафмана можна умовно поділити на дві частини: побудова оптимального кодового дерева й власне генерація закодованої послідовності. В неадаптивному алгоритмі для побудови оптимального кодового дерева необхідно підрахувати частоту кожного символа і, використовуючи отриману таблицю (словник) частот й узявши символи за листки дерева, застосувати наступний алгоритм:

– обрати два вузли з найменшими частотами, що приймаються за їхню «вагу»;

– від них створити батьківський вузол, вага якого буде сумою ваг обраних дочірніх елементів;
– для лівої та правої дуги від батьківського елемента до дочірніх ставиться в відповідність 0 та 1;
– кроки 1-3 повторюються, причому тепер у списку вільних для вибору вузлів буде новостворений вузол, а його дочірні елементи стають недоступними для вибору;
– коли в дереві залишається лише один вільний вузол, дерево вважається завершеним, і цей вузол є його коренем, що зберігає в собі суму ваг всіх початкових символів.

Таким чином, дерево будується від листків до кореня. На рисунку 2.5 показано приклад побудови дерева для певної таблиці частот символів.

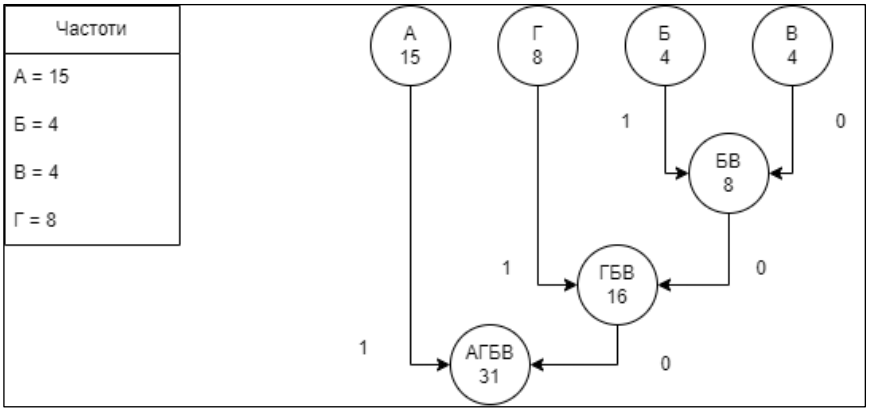


Рисунок 2.5 – Приклад побудови дерева Хафмана. Вузол «АГББВ» (31) – корінь дерева (рисунок виконано самостійно)

Після побудови дерева фактично отримуємо схему, за якою кодується будь-який символ зі словника. Для кодування повідомлення (послідовності символів) для кожного з них необхідно від відповідного листка пройти шлях до кореня, і при кожному переході до коду додавати відповідно 1 або 0. Досягнувши кореня, послідовність необхідно обернути, отримавши шуканий код. Так, у прикладі на рис. 3.3 символ «А» кодуватиметься одним бітом «1», а «Б» - трьома («001»). Відповідно, для цього повідомлення, що початково складалось із $31 * 8 = 248$ байтів (за умови використання одного байту на символ), стиснута версія матиме

розмір $15 * 1 + 8 * 2 + (4 * 3) * 2 = 55$ бітів, або 6,88 байтів. Звісно, що на сучасних операційних системах і пристроях мінімальною одиницею зчитування й запису інформації є байт, тому насправді до стиснутого файлу буде записано 56 бітів, або 7 байтів. Це можна виконати багатьма способами, найпростіший – додати нульові біти до останнього «неповного» байту, вказавши при цьому в окремому точно визначеному байті файлу їхню кількість.

Для декодування за допомогою неадаптивного алгоритму необхідно мати дані про частоти символів, щоб побудувати дерево, аналогічне тому, що було представлене на рис. 2.5. Тоді, маючи стиснуті дані й беручи до уваги, що всі коди є префіксними, можемо застосувати такий алгоритм:

– узяти наступний біт із послідовності й додати його до буферу;
– якщо значення в буфері збігається з кодом символа – записати цей символ;
– повторювати, доки не буде досягнуто кінця закодованого повідомлення.

Нехай маємо послідовність бітів «001101011000». Використовуючи дерево з попереднього прикладу, розкодуємо послідовність. Перша послідовність бітів, що відповідає листку – «001», що в побудованому дереві відповідає символу «Б». Повторивши із наступними кодами – «1», «01», «01», «1» і «000» – отримаємо повідомлення «БАГГАВ».

Алгоритм Хафмана є одним із основних алгоритмів кодування текстової інформації, що застосовується в різних форматах стиснутих даних, зокрема ZIP і PNG. Основним його недоліком є слабка ефективність при роботі з даними з високою ентропією, тобто такими, де частоти всіх символів відрізняються не набагато. Тому алгоритм Хафмана частіше використовується як інструмент додаткового стиснення даних, що були приведені до певного структурованого виду, наприклад, описаними вище LZ77 або LZW.

2.1.7 Алгоритм Deflate

Типовий потік Deflate складається із блоків, перед кожним із яких іде заголовок із трьох бітів, що визначають, чи блок є останнім, а також тип

кодування. Так, перший біт, виставлений в нуль, означає, що блок не є останнім, і навпаки, одиниця означає, що блок, який розглядається, останній у потоці. Два наступних біти блоку визначають тип кодування Хафмана, що застосовувалось до даних у ньому. Їхні можливі значення та їх тлумачення наведено в таблиці 2.1. В першому стовпці наведено значення для першого біту, а в першому рядку – для другого.

Таблиця 2.1 – Тип кодування Хафмана залежно від значень бітів (зі специфікації Deflate [7])

Перший біт / Останній біт	0	1
0	Незакодовані дані	Статичне кодування
1	Динамічне кодування	Помилка

Стиснення даних в алгоритмі Deflate відбувається спочатку за допомогою алгоритму LZ77, що замінює повторювані рядки вказівниками, приводячи повідомлення до певного структурованого вигляду, що дозволяє зменшити словник для алгоритму Хафмана, що застосовується на другому етапі.

Зазвичай блоки кодуються за допомогою динамічного (адаптивного) кодування Хафмана, що представляє для кожного окремого блоку оптимізоване дерево кодів. При кодуванні статичним (неадаптивним) методом, дані для побудови дерева представляються одразу після заголовку.

2.1.8 BZip2

BZip2 стискає вхідні дані, попередньо розбиті на блоки фіксованої довжини, за допомогою послідовності BWT – MTF – кодування Хафмана. Формат також включає метадані, зокрема число π у форматі BCD для позначення початку нестиснених даних, і $\sqrt{\pi}$ у тому ж форматі для позначення кінця. Розмір блока та вказівники на метод стиснення вказуються на початку файлу [11].

2.1.9 Алгоритм Brotli

Brotli постачається разом зі статичним словником, що містить більш ніж 10000 слів і термінів із англійської та деяких інших мов та їхніх перетворень, що визначаються за префіксами й суфіксами. Алгоритм використовує їх, щоб розпізнавати точки в коді й не читати код повністю, натомість він знаходить певні «підказки», визначає за ними певний термін і рухається далі.

З використанням словника Brotli значно покращив стиснення Gzip, яке його не використовує. Робота Brotli теж заснована на використанні LZ77 та оптимального кодування Хафмана, але також Brotli застосовує алгоритм моделювання контексту. Всього алгоритму доступні чотири режими визначення контексту, що визначають його ідентифікатор. Вони базуються на двох останніх байтах потоку p_1 і p_2. На початку потоку ці два байти ініціалізуються як нульові. Контекст визначається за допомогою таких способів (режимів):

- LSB6, де контекст визначається шістьома найменш значущими бітами p_1;
- MSB6, де контекст визначається шістьома найбільш значущими бітами p_1;
- UTF8, де контекст визначається складеною функцією від p_1 і p_2, оптимізованою для стиснення тексту;
- Знакова (Signed), де контекст визначається складеною функцією від p_1 і p_2, оптимізованою для стиснення цілих чисел.

Brotli визначає контекст за допомогою таблиць, що постачаються разом із ним. Так, існує три таблиці: LUT0, LUT1 і LUT2, від LookUp Table [12]. Відповідно, ID контексту знаходиться для кожного зі способів по-різному:

- для LSB6: CID = p1 & 0x3f;
- для MSB6: CID = p1 >> 2, де “>>” - побітовий зсув праворуч;
- для UTF8: CID = LUT0[p1] | LUT1[p2] , де “|” – побітовий оператор OR;
- для Signed: CID = (Lut2[p1] << 3) | Lut2[p2].

2.1.10 Алгоритм PPM

Основна ідея алгоритму передбачення за частковим співпадінням (Prediction by partial matching, PPM) – знаходження контексту поточного символа з деякого числа попередніх. Сама модель не виконує стиснення, а лише передбачає значення символу, що стискається безпосередньо методами ентропійного стиснення, такими як оптимальне кодування Хафмана або арифметичне кодування.

Зазвичай довжина контексту, що використовується, сильно обмежується. Вона позначається як *n,* а модель порядку *n* позначається як PPM(n). Якщо передбачення контексту з n символів не може бути виконане, то застосовується контекст із n-1 символами. Переходи відбуваються рекурсивно, допоки контекст не буде знайдено, або доки *n* не стане 0. Модель нульового порядку еквівалентна випадку, коли ймовірність кожного символу пропорційна частоті його появи в повідомленні, що подібне до оптимального кодування Хафмана, з яким дана модель і застосовується. PPM представляє варіант стратегії перемішування, коли оцінки ймовірностей, зроблені за допомогою контекстів різної довжини, об’єднуються в загальну ймовірність, і потім ця оцінка кодується певним ентропійним архіватором, зазвичай, оптимальним кодуванням Хафмана або арифметичним. Власне, стиснення відбувається саме на етапі використання ентропійного кодування[13].

Наприклад, маючи фразу “ANANAS-AND-BANAN” і символ “A”, ми можемо визначити контекст за допомогою алгоритму PPM другого порядку. Фактично, це означає, що ми повинні взяти два останніх байти й знайти, які символи йдуть після них. Якщо буде знайдено символ, який ми кодуємо, то процес кодування завершується. В таблиці 2.2 показано знайдені символи після “AN” у послідовності.

Таблиця 2.2 – Символи для контексту “AN” (виконана самостійно)

A	D	EOL
3	1	1

Оскільки було знайдено символ “A”, можемо закодувати його за допомогою частоти $\frac{3}{5}$. Його можна виконати, наприклад, за допомогою оптимального кодування Хафмана, прийнявши як частоту появи символа “A” $\frac{3}{5}$. Таким чином, частота появи символа залежить від певного «контексту» з кількох інших символів за ним [14].

2.2 Критерії порівняння алгоритмів

Задача багатокритеріального вибору полягає в виборі алгоритму стиснення, що стискає файли із мінімальним відношенням розміру стисненого файлу до початкового, працює якнайшвидше і використовує найменше оперативної пам’яті. Отже, маємо наступні критерії:

– коефіцієнт стиснення;
– час кодування;
– час декодування;
– кількість використаної оперативної пам’яті під час кодування;
– кількість використаної оперативної пам’яті під час декодування.

За цими критеріями маємо такі шкали:

– коефіцієнт стиснення: кількісна шкала відношень. Приймає будь-які раціональні значення, більші за 0, в дослідженні округлюється до 2 знаків після коми;
– час кодування: кількісна шкала різниць. Може приймати будь-які цілі значення більші за нуль. Вимірюється в мілісекундах;
– час декодування: кількісна шкала різниць. Може приймати будь-які цілі значення більші за нуль. Вимірюється в мілісекундах;
– кількість використаної оперативної пам’яті під час кодування: абсолютна кількісна шкала. Може приймати цілі значення, більші за 0. Вимірюється в байтах;

– кількість використаної оперативної пам’яті під час декодування: абсолютна кількісна шкала. Може приймати цілі значення, більші за 0. Вимірюється в байтах.

За цими критеріями необхідно провести порівняння роботи алгоритмів, а також їхніх поєднань, із різними типами даних. Отримані результати повиннні обґрунтовувати вибір системи, що розробляється, стосовно даних того чи іншого типу.

2.3 Критерії порівняння тестових даних

В сучасних обчислювальних пристроях і носіях інформації кожен файл представлений в вигляді певної послідовності байтів, що містять числа від 0 до 255. Кожен із цих байтів можна співставити з відповідним символом у таблиці ANSI [15]. Відповідно, існує можливість створити словник, у якому кожен байт означає конкретний символ, причому неважливо, чи є символ друкованим, чи керівним. Тоді будь-який файл можна представити як скінченну послідовність символів із такого словника, отже, за його допомогою дані можуть бути стиснуті за допомогою ентропійного кодування.

Згідно з теоремою Шеннона про джерело кодування, деяка кількість N випадкових змінних з ентропією H(X) можуть бути стиснуті без значного ризику втрати інформації в не менш ніж NH(X) бітів інформації [16]. Відповідно, існує певна межа стиснення інформації, що залежить від її ентропії: краще стиснути можна дані, що є більш передбачуваними. Тоді вхідні дані можна розрізняти за значенням середньої ентропії.

Розмір даних також може впливати на їхнє стиснення. Так, дані, менші за певні порогові значення, більш імовірно матимуть коефіцієнт стиснення, менший за 1. Наприклад, для кодування даних за допомогою алгоритму Хафмана необхідно записати до файлу дані про префіксне дерево, що можуть займати більше ніж нестиснені дані. Тому об’єм вхідних даних може зіграти суттєву роль для алгоритмів, які потребують зберігання більшої кількості даних для декодування.

Дані для експериментів і обробки в цілому подаються як файли фіксованого розміру. Отже, їх можна порівнювати за такими характеристиками:

– середня ентропія повідомлення;
– розмір.

За цими критеріями маємо метрики:

– середня ентропія повідомлення – кількісна шкала відношень. Може приймати значення в діапазоні [0; 8]. Вимірюється в бітах;
– розмір – абсолютна кількісна шкала. Може приймати цілі значення, більші за 0. Вимірюється в байтах.

За допомогою цих метрик можна визначити групи даних, що використовуватимуться для дослідження, і, відповідно, застосовувати дані, згруповані за цими метриками в програмній реалізації.

2.4 Обґрунтування методів дослідження

Об’єктом дослідження є методи стиснення та оптимального кодування даних. Тоді предмет дослідження – характеристики цих методів, найбільш ефективні алгоритми для вирішення конкретних задач стиснення даних певного виду, формату тощо.

Для визначення ефективності алгоритмів стиснення даних без втрат визначається коефіцієнт стиснення – відношення початкового розміру нестиснених даних до розміру даних на виході роботи алгоритму (формула 2.1).

$$k = \frac{V_0}{V_{\text{стисн}}} \tag{2.1}$$

Вхідні дані необхідно порівнювати за середньою ентропією повідомлення, яку можна визначити за формулою Шеннона (формула 2.2):

$$H(X) = -K \sum_{i=1}^{n} p_i * \log_2 p_i \tag{2.2}$$

де H(X) – середня ентропія повідомлення;

K – сталий коефіцієнт, що визначає одиниці виміру ентропії, а його зміна рівносильна зміні основи логарифму;

p_i – частота появи і-го символу в повідомленні;

n – потужність словника, фактично дорівнює кількості символів у ньому [17].

Таким чином, значення середньої ентропії є максимальним для повідомлень, у яких частоти появи всіх символів зі словника є рівними. За формулою 2.3

$$max(H) = -n * \frac{1}{n} * \log_2 \frac{1}{n} = -\log_2 \frac{1}{n} \quad (2.3)$$

де max(H) – максимальна середня ентропія повідомлення для словника з n символів, частота кожного з яких – $\frac{1}{n}$.

Мінімальне значення середньої ентропії для повідомлення, розмір якого необмежений, наближається до 0, проте саме нульового значення вона може набути лише за використання словника, що складається з 1 символа, оскільки

$$-1 * \log_2 1 = 0$$

Для повідомлень, що використовують більше одного символа, мінімальна середня ентропія може бути набутою тоді, коли різниця між частотами символів є найбільшою. За умови, що кожен символ зі словника зустрічається в повідомленні хоча б один раз і повідомлення має сталу довжину, маємо найвищу можливу частоту одного символу (формула 2.4):

$$\max(p) = 1 - \frac{d-1}{n} = \frac{n-d+1}{n}, n \geq d \quad (2.4)$$

де max(p) – максимальна можлива частота символу,

d – потужність словника,

n – довжина повідомлення.

Середня ентропія повідомлення визначає те, скільки в середньому інформації несе один символ у повідомленні. Через це вона не може набувати значень більших за ту кількість бітів, що надається для кожного символу, тому, оскільки всі методи, що розглядаються, оперують байтами, значення ентропії не можуть бути більшими за 8. При цьому за формулою (2.3) максимальне значення досягається лише за умови коли застосовуються всі 256 значень байтів, і їхня частота є рівною $\frac{1}{256}$.

Вищезазначені формули можуть бути використані для генерації тестових даних, оскільки за їх допомогою можна знайти необхідну кількість частот символів, а також визначити мінімальний розмір словника для їх заповнення. Отже, планується використовувати емпіричні та математичні методи дослідження. За необхідності потрібно створити реалізації алгоритмів, або використати бібліотеки, в яких ці алгоритми було реалізовано.

2.5 Проектування бази даних

Для зберігання результатів експериментів необхідно мати базу даних, що дозволятиме зберігати такі класи об’єктів:

– алгоритми стиснення даних;
– характеристики тестових даних;
– дані з дослідження роботи алгоритмів із тестовими даними.

На рис. 2.4 зображено ER-діаграму, на якій зазначено сутності для зберігання даних, отриманих в результаті проведення експериментів, а також їхні відносини.

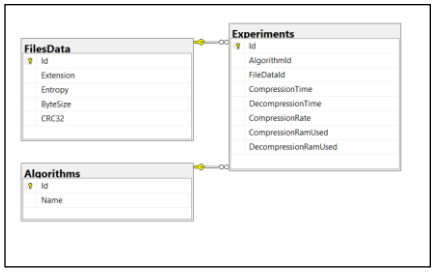


Рисунок 2.4 – ER-діаграма сутностей для проведення експериментів (рисунок виконано самостійно)

База даних має такі сутності:

– Algorithms – дані про алгоритми. Має поле Name типу varchar, що зберігає назву алгоритму;
– Files – дані про файли, що представляють тестові дані. Має поле Name (varchar) для зберігання назви файлу, а також Entropy (double) та ByteSize (int) для зберігання метрик. Для визначення однакових файлів під час тестів застосовується CRC32, що також зберігається у відповідному полі в БД;
– CompressionData – дані про результати експериментів із певними алгоритмами, визначеними за ідентифікатором AlgorithmId, та даними, визначеними за FileId. Має власні поля CompressionRate (double), EncodingTime (double), DecodingTime (double), CompressionRamUsed (int), DecompressionRamUsed (int) для зберігання метрик.

2.6 Планування експериментальних досліджень

Для проведення експериментальних досліджень необхідно визначити, які алгоритми та їх поєднання будуть застосовуватись до тестових даних. Так, маємо наступні алгоритми:

– LZ77;
– LZW;
– RLE;
– кодування Хафмана;
– Brotli;
– PPM;
– MTF;
– BWT.

Вже було розглянуто такі поєднання вищезазначених алгоритмів, як LZ77 – код Хафмана (Deflate), BWT – RLE та MTF – RLE. Слід також зауважити, що алгоритм Brotli теж є поєднанням алгоритмів LZ77, оптимального кодування

Хафмана та модифікованого контекстного моделювання. Для стиснення за допомогою алгоритму PPM застосовується арифметичне кодування.

Додатково можна застосовувати послідовно такі алгоритми, як BWT – MTF – кодування Хафмана – такий метод стиснення реалізовано, зокрема, в програмі bzip2. В цій послідовності можна замінити алгоритм MTF на RLE, або додати його перед кодуванням Хафмана, отримавши нові послідовності. Проаналізувавши їхню роботу з тестовими даними, можна зрозуміти, як на роботу методу в цілому впливає той чи інший алгоритм. Таким чином, отримаємо 14 алгоритмів та їх поєднань, що виконують безпосередньо стиснення для тестування:

– LZ77;
– LZW;
– RLE;
– кодування Хафмана;
– Brotli (3 режими роботи);
– PPM;
– Deflate;
– Deflate із використанням LZW замість LZ77;
– MTF – LZ77;
– MTF – LZW;
– MTF – RLE;
– BWT – RLE;
– BWT – MTF – кодування Хафмана;
– bZip2.

Тестові дані для проведення експериментів необхідно згенерувати таким чином, щоб кожне значення ентропії було представлене однаковою кількістю файлів. Це забезпечить збалансованість вибірки і дозволить отримати більш точні та достовірні результати під час аналізу. Для досягнення цього можна використовувати різні методи генерації даних, які забезпечать необхідний розподіл значень ентропії.

Також необхідно провести тестування реальних даних. Наприклад, можна використовувати текстові дані природними мовами, HTML-сторінки та програмний код, нестиснені статичні та відеозображення (BMP, AVI) тощо. Таким чином можна досягти найбільш різноманітних за метриками тестових даних.

Алгоритми за можливістю необхідно тестувати з використанням багатопотоковості. Це дозволяє значно підвищити продуктивність і скоротити час виконання, особливо для великих обсягів даних. Наприклад, оптимізація алгоритму перетворення Барроуза-Віллера (BWT) може бути досягнута шляхом розділення створення матриці ротацій на декілька потоків. Також можна оптимізувати й інші алгоритми, розбивши повідомлення на блоки, що кодуються паралельно як окремі повідомлення, незалежно один від одного.

Під час тестування необхідно виміряти час кодування та декодування, використання оперативної пам’яті для цих процесів, а також ступінь стиснення (за формулою 2.1).

Дані до бази даних необхідно заносити після кожного виконання алгоритмом або їх поєднанням роботи. Для практичного застосування дані про файл, наданий користувачем, необхідно порівняти з тими, що використовувались у дослідженні, та обрати найближчий за значеннями метрик.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У даному розділі буде розглянуто створення та налаштування проектів для тестування алгоритмів і застосування результатів експериментів. Для роботи обидвох використовується .NET Core 8 та БД MS SQL. Для формування запитів використовується Entity Framework 8 [18].

3.1 Програмна реалізація експериментів

Для проведення досліджень було створено окремий застосунок, що включає в себе методи для керування даними, реалізації алгоритмів стиснення та бібліотеки. Зокрема, для таких алгоритмів як Deflate, bzip2 та Brotli було використано бібліотеки System.IO.Compression, SharpZipLib та SharpCompress, а для інших – окремі реалізації з відкритим кодом (PPM, LZW, LZ77 тощо). Такі алгоритми, як MTF та оптимальне кодування Хафмана було реалізовано самостійно.

За попереднім проектуванням було також розроблено класи моделей БД, необхідних для зберігання даних про алгоритми, файли та результати експериментів. SQL-запити для генерації моделей БД наведено нижче.

```
CREATE TABLE [dbo].[Algorithms](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [Name] [nvarchar](max) NOT NULL,
 CONSTRAINT [PK_Algorithms] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)
CREATE TABLE [dbo].[FilesData](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [Extension] [nvarchar](max) NOT NULL,
      [Entropy] [float] NOT NULL,
      [ByteSize] [int] NOT NULL,
      [CRC32] [bigint] NOT NULL,
 CONSTRAINT [PK_FilesData] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)
CREATE TABLE [dbo].[Experiments](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [AlgorithmId] [int] NOT NULL,
      [FileDataId] [int] NOT NULL,
      [CompressionTime] [bigint] NOT NULL,
```

```
    [DecompressionTime] [bigint] NOT NULL,
    [CompressionRate] [float] NOT NULL,
    [CompressionRamUsed] [bigint] NOT NULL,
    [DecompressionRamUsed] [bigint] NOT NULL,
 CONSTRAINT [PK_Experiments] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)
```

Для кожного зі значень ентропії від 0 до 9 було згенеровано дані різного об’єму, що визначається такими категоріями:

– 100 байтів;
– 1024 байти (1 кілобайт);
– 10240 байтів (10 кілобайтів);
– 102400 байтів (100 кілобайтів);
– 204800 байтів (200 кілобайтів);
– 512000 байтів (500 кілобайтів);
– 1048576 байтів (1 мегабайт).

Розмір файлу може відрізнятись від розміру, заявленого в категорії, не більш ніж на 6% для даних розміром 100 байтів і не більш ніж на 2% для інших, окрім винятків, описаних нижче.

За формулою (2.3) максимальною середньою ентропією повідомлення, якої можна досягти використовуючи 100 байтів є $-\log_2 \frac{1}{100} \approx 6{,}64$ біта на символ. Тому для досягнення ентропії, близької до 7 і 8 відповідно використовується 110 і 250 байтів відповідно, що дає середню ентропію повідомлення $-\log_2 \frac{1}{110} \approx 6.78$ і $-\log_2 \frac{1}{250} \approx 7{,}96$ бітів на символ відповідно. Файли інших розмірів генеруються без змін із використанням тих же частот символів.

Під час тестування кожен алгоритм працює з кожним файлом тричі. Проводиться вимірювання витрат часу та оперативної пам’яті на його стиснення та відновлення, після проходження циклу з них обираються найменші для зменшення впливу сторонніх чинників [19]. За допомогою формули (2.1) знаходиться коефіцієнт стиснення. Після виконання циклу підсумкові результати експерименту заносяться до бази даних.

Приклад коду для визначення об'єму оперативної пам'яті, використаної алгоритмом:

```
public static (long mem, T result) GetMemoryUsed<T, I>(I input, Func<I, T>
funcToMeasure)
{
    var initialMemory = GC.GetTotalMemory(forceFullCollection: true);
    var result = funcToMeasure(input);
    var finalMemory = GC.GetTotalMemory(forceFullCollection: false);
    var consumption = Math.Abs(finalMemory - initialMemory);
    return (consumption, result);
}
```

3.2 Програмна реалізація демонстраційного додатку

Для демонстрації результатів, отриманих у результаті експериментів, було розроблено додаток, що дозволяє аналізувати файли, що надаються користувачем, і визначати алгоритми, застосування яких дасть найбільші коефіцієнти стиснення. На стороні клієнта знаходиться середня ентропія наданих ним даних, розмір файлу та його розширення, які відправляються на сервер. Сервер, використовуючи раніше проведені експерименти, знаходить експеримент, що виконувався над файлом із найбільш схожими значеннями, і виділяє 5 алгоритмів, що мали найбільші коефіцієнти стиснення такого файлу, після чого клієнтська частина пропонує користувачеві стиснути наданий файл одним із цих алгоритмів. Стиснутий файл також можна відновити за допомогою іншого інтерфейсу.

Для реалізації серверної частини було використано ASP .NET, мову програмування C#. Дані отримуються з раніше створеної БД MS SQL, комунікація з якою відбувається за допомогою Entity Framework. Клієнтську частину розроблено за допомогою React із використанням бібліотеки Axios для обробки GET та POST запитів.

Клієнтський додаток може використовуватись у двох режимах: стиснення та відновлення даних. Під час стиснення користувач обирає файл, який аналізується на стороні клієнта і знаходить необхідні для подальшої обробки значення: середню ентропію, розмір та назву. Код для визначення даних файлу наведено нижче:

Після цього дані обробляються на сервері, де відбувається підбір п’яти рекомендованих алгоритмів. Код серверної частини підбору наведено нижче:

```
public async Task<IEnumerable<FileAnalysisDTO>> GetFileAnalysis([FromBody]
FileDataDTO givenFileData)
{
    var extension = givenFileData.Name.Split('.').Last();
    var possibleAlgExperiments = _context.Experiments
        .OrderBy(e => Math.Abs(Math.Round(e.FileData.Entropy -
givenFileData.Entropy)))
        .ThenBy(e => Math.Abs(e.FileData.ByteSize -
givenFileData.ByteSize))
        .ThenByDescending(e => e.FileData.Extension == extension)
        .ThenByDescending(e => e.CompressionRate)
        .Where(e => e.CompressionRate > 1)
        .Take(5);
    return await possibleAlgExperiments.Select(e => new FileAnalysisDTO()
    {
        DecompressionTime = e.DecompressionTime,
        AlgorithmId = e.AlgorithmId,
        CompressionRate = e.CompressionRate,
        CompressionTime = e.CompressionTime
    })
        .ToListAsync();
}
```

Серед рекомендацій користувач обирає необхідний йому компресор (рис. 3.1).

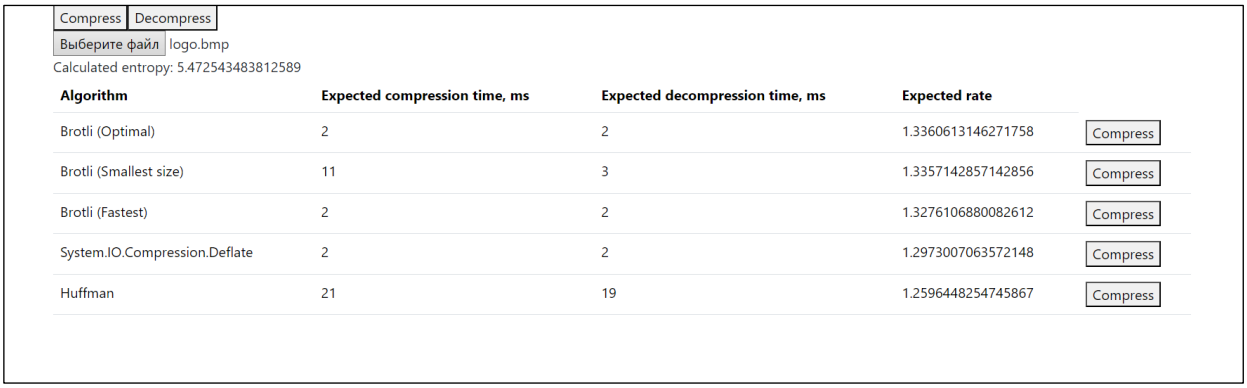


Рисунок 3.1 – Приклад інтерфейсу додатку (рисунок виконано самостійно)

Після обрання методу стиснений файл зберігається на комп’ютері користувача в папку із завантаженнями браузера. Нижче наведено код для збереження файлу:

```
const handleCompressSelected = async (id: number) => {
```

```
  let { compressionResult } = await compressFile(id, props.fileBytes);
  const pdfBlob = new Blob([compressionResult], {
    type: "application/octet-stream",
  });
  const url = window.URL.createObjectURL(pdfBlob);
  const tempLink = document.createElement("a");
  tempLink.href = url;
  tempLink.setAttribute(
    "download",
    `compressed.${
      props.data.name.toLowerCase().charAt(0) +
      props.data.name.toLowerCase().charAt(1) +
      props.data.name.toLowerCase().charAt(2)
    }`
  );
  document.body.appendChild(tempLink);
  tempLink.click();
  document.body.removeChild(tempLink);
  window.URL.revokeObjectURL(url);
};
```

За необхідності стиснутий файл можна відновити, проте для цього необхідно вказати бажане розширення, з яким файл буде збережено після декомпресії, а також алгоритм, який повинен збігатись із тим, яким першочергово було стиснено файл. На рисунку 3.2 зображено приклад інтерфейсу додатку з обраним режимом відновлення.

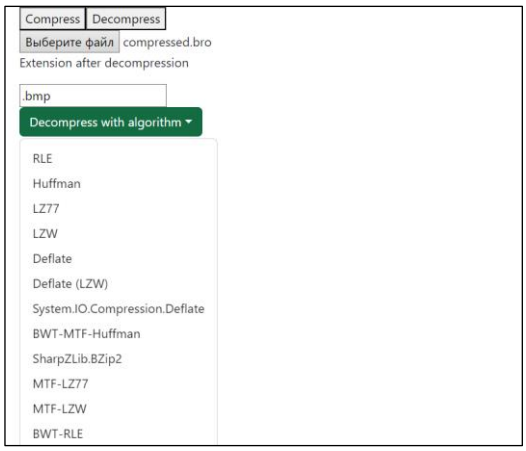


Рисунок 3.2 – Приклад інтерфейсу для відновлення файлу (рисунок виконано самостійно)

Після відновлення файл також зберігається на пристрої користувача.

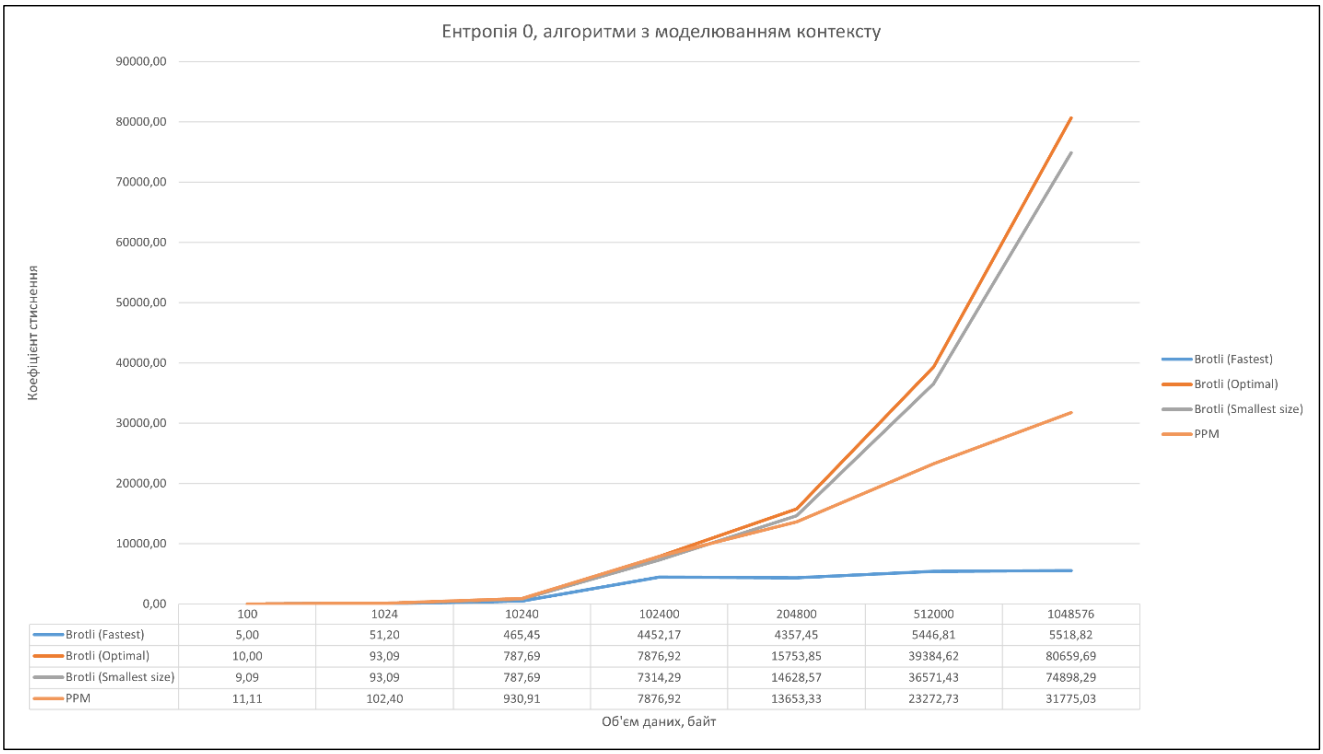
4 ОПИС ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

4.1 Результати експериментальних досліджень

Згідно з планом, було проведено експерименти зі згенерованими та з реальними даними. Було проведено тестування загалом 17 алгоритмів та їхніх поєднань, результати було розділено за значеннями ентропії та об’єму даних і відображено на графіках. Алгоритми на графіках розділено за групами:

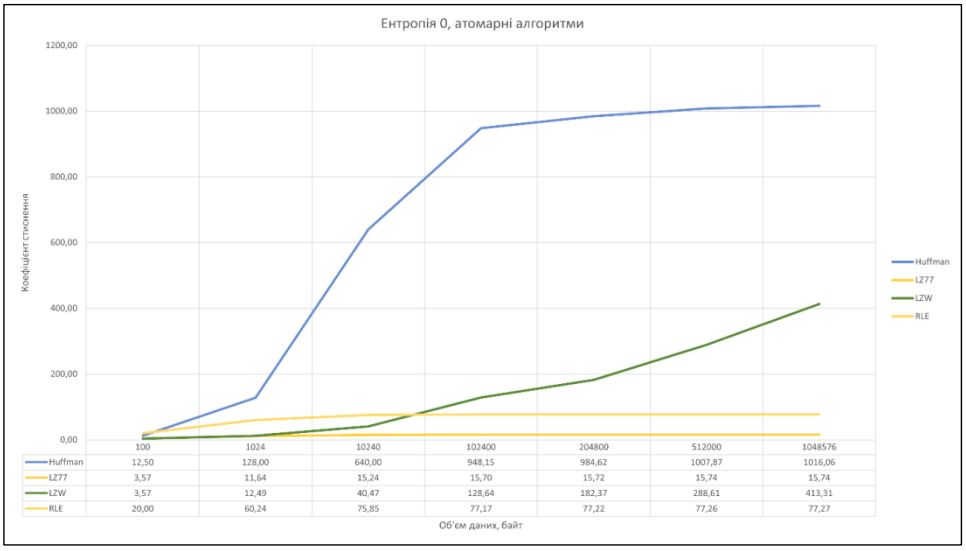
– алгоритми з використанням моделювання контексту;
– атомарні алгоритми (неподільні завершені алгоритми);
– складні алгоритми (поєднання атомарних алгоритмів, окрім тих, що використовують моделювання контексту).

На рисунках 4.1-4.3 показано результати роботи алгоритмів із тестовими даними з ентропією, що дорівнює 0.



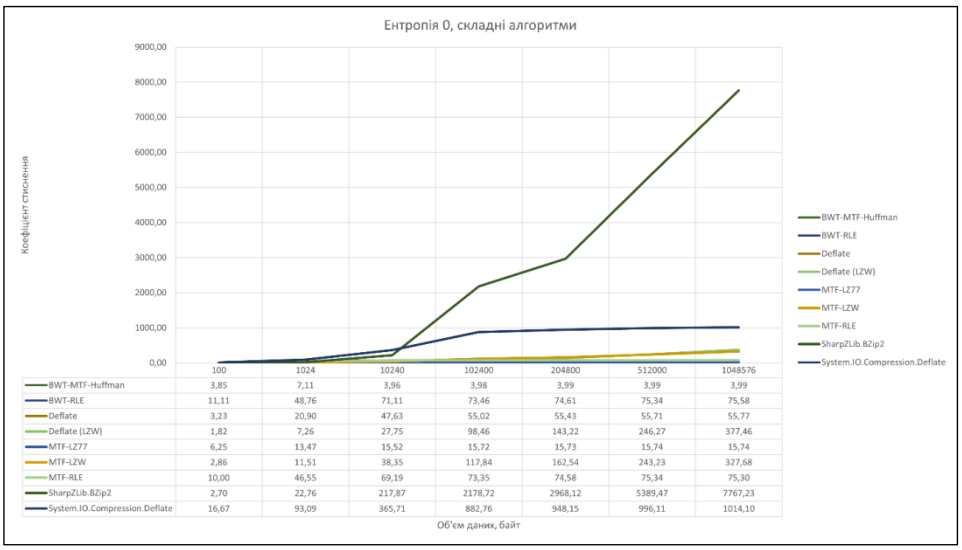
	100	1024	10240	102400	204800	512000	1048576
Brotli (Fastest)	5,00	51,20	465,45	4452,17	4357,45	5446,81	5518,82
Brotli (Optimal)	10,00	93,09	787,69	7876,92	15753,85	39384,62	80659,69
Brotli (Smallest size)	9,09	93,09	787,69	7314,29	14628,57	36571,43	74898,29
PPM	11,11	102,40	930,91	7876,92	13653,33	23272,73	31775,03

Рисунок 4.1 – Робота алгоритмів з використанням моделювання контексту на даних з ентропією 0 (рисунок виконано самостійно)



	100	1024	10240	102400	204800	512000	1048576
Huffman	12,50	128,00	640,00	948,15	984,62	1007,87	1016,06
LZ77	3,57	11,64	15,24	15,70	15,72	15,74	15,74
LZW	3,57	12,49	40,47	128,64	182,37	288,61	413,31
RLE	20,00	60,24	75,85	77,17	77,22	77,26	77,27

Рисунок 4.2 – Робота атомарних алгоритмів на даних з ентропією 0 (рисунок виконано самостійно)



	100	1024	10240	102400	204800	512000	1048576
BWT-MTF-Huffman	3,85	7,11	3,96	3,98	3,99	3,99	3,99
BWT-RLE	11,11	48,76	71,11	73,46	74,61	75,34	75,58
Deflate	3,23	20,90	47,63	55,02	55,43	55,71	55,77
Deflate (LZW)	1,82	7,26	27,75	98,46	143,22	246,27	377,46
MTF-LZ77	6,25	13,47	15,52	15,72	15,73	15,74	15,74
MTF-LZW	2,86	11,51	38,35	117,84	162,54	243,23	327,68
MTF-RLE	10,00	46,55	69,19	73,35	74,58	75,34	75,30
SharpZLib.BZip2	2,70	22,76	217,87	2178,72	2968,12	5389,47	7767,23
System.IO.Compression.Deflate	16,67	93,09	365,71	882,76	948,15	996,11	1014,10

Рисунок 4.3 – Робота складних алгоритмів на даних із ентропією 0 (рисунок виконано самостійно)

З отриманих результатів можна побачити, що дані, які мають середню ентропію 0 бітів на символ, тобто складаються з одного символу, стискаються з досить великими коефіцієнтами.

Варто зауважити, що для даних розміром 100 байтів найкраще спрацювало стиснення RLE, що може бути пов’язане з меншою кількістю метаданих, що кодуються разом зі стисненими даними. З більшими файлами краще всього

працювали алгоритми Brotli, BZip2 та PPM. Очевидно, що алгоритми MTF та BWT лише погіршують стиснення, що можна помітити на прикладі RLE, оскільки вони залишають додаткові дані про блоки, на які розбивались нестиснені дані, але самі дані фактично не змінюються.

На рисунку 4.4 показано час стиснення кожного файлу алгоритмами.

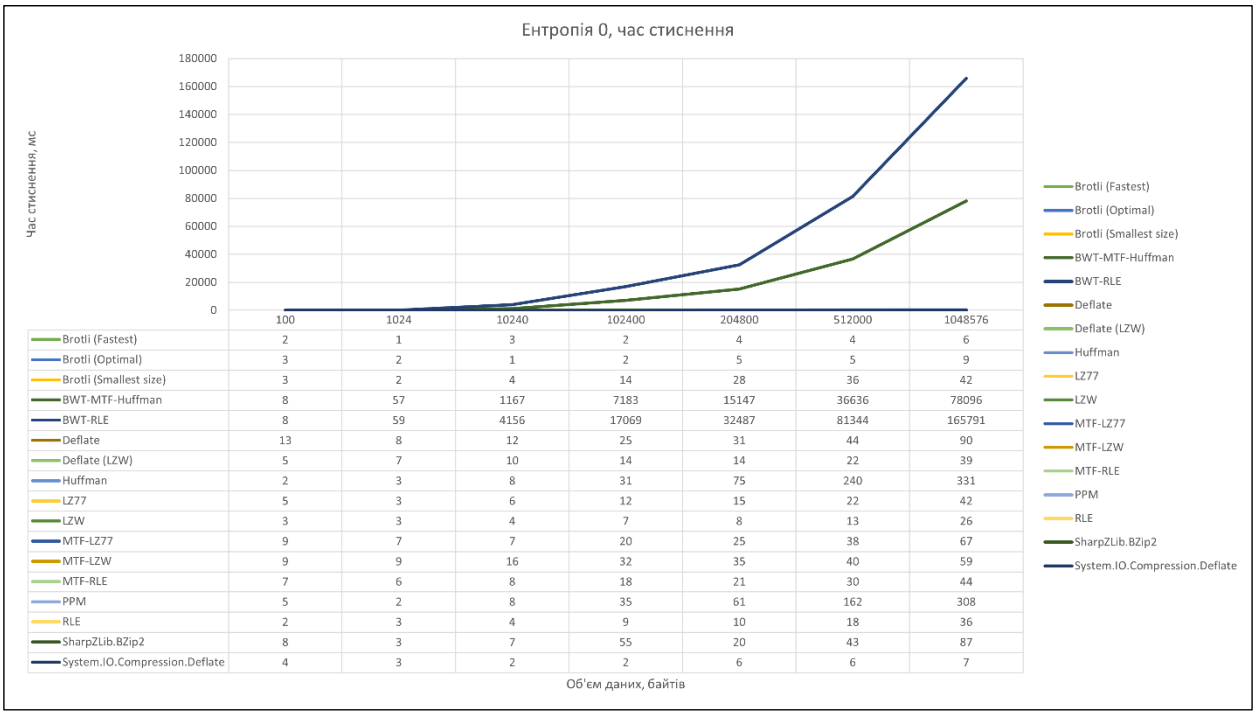


Рисунок 4.4 – Час стиснення даних (рисунок виконано самостійно)

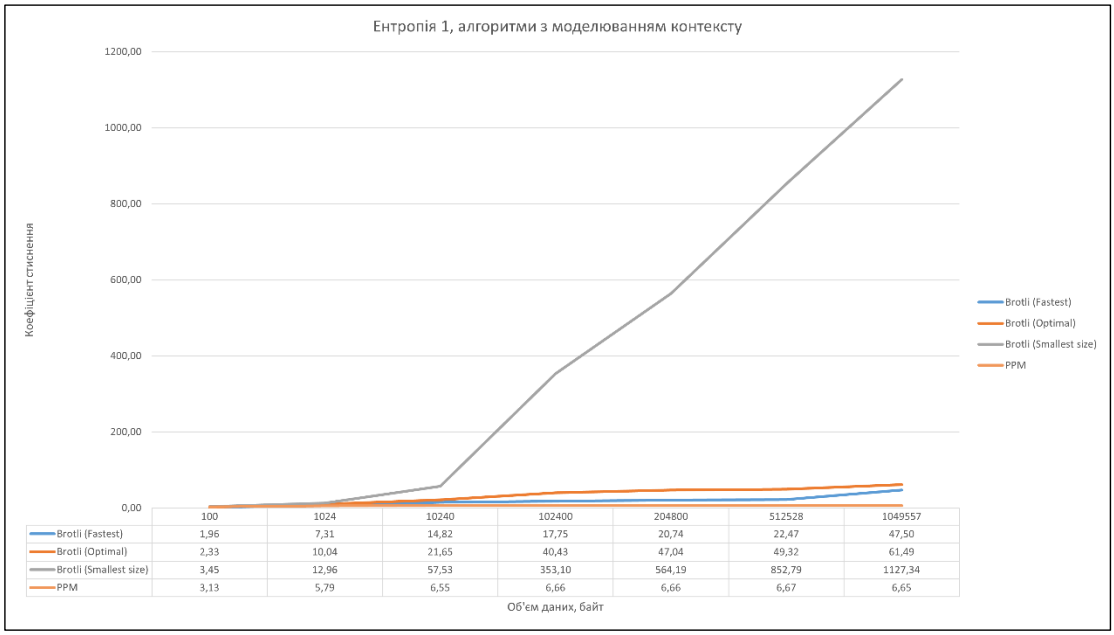
На рисунку 4.4 видно, що використання алгоритму BWT доволі сильно сповільнює процес стиснення, причому час зростає пропорційно до об’єму даних.

Це пов’язано з тим, що алгоритм BWT включає створення та сортування матриці зсувів, що займає багато часу навіть за умови використання паралелізму. Можна також помітити, що час стиснення алгоритмами PPM та BZip2 теж зростає, хоча й не так швидко. Натомість зростання часу стиснення за допомогою Brotli помітне лише для режиму “Smallest size”, але воно не є значно більшим, ніж для інших алгоритмів.

Порівнюючи роботу алгоритмів Deflate із використанням LZ77 та LZW можна помітити, що LZW перевершує за коефіцієнтом стиснення LZ77 починаючи з позначки об’єму даних у 200 Кб, причому за використання LZ77 коефіцієнт майже не зростає від 100 Кб.

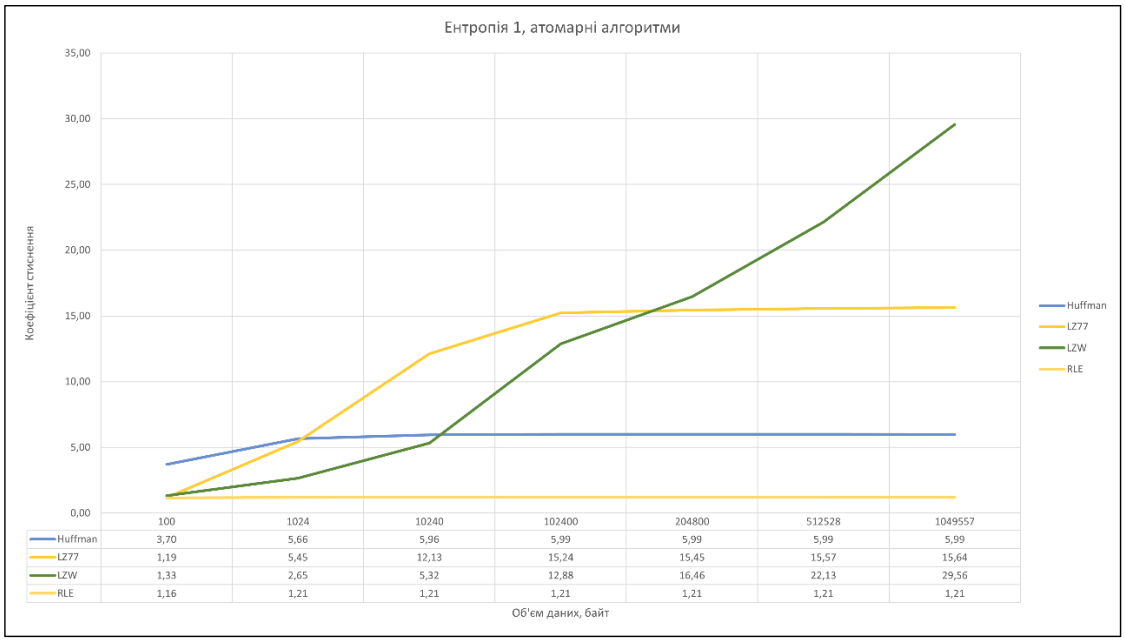
Варто також зауважити, що використання лише оптимального кодування Хафмана перевершує обидва методи Deflate, оскільки дані, що складаються з одного символа, кодуються лише одним числом, тоді як використання словникових методів порушує послідовність однакових символів.

На рисунках 4.5-4.7 зображено графіки, що ілюструють роботу алгоритмів із даними з ентропією, близькою до 1.



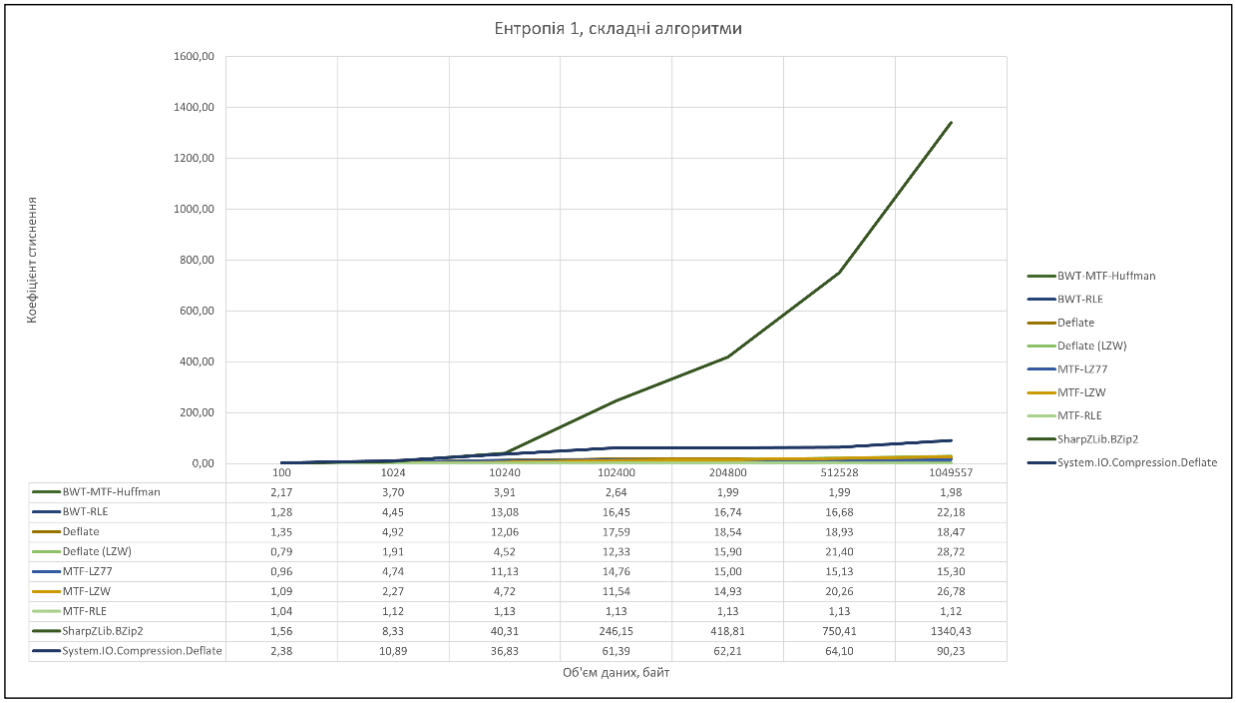
	100	1024	10240	102400	204800	512528	1049557
Brotli (Fastest)	1,96	7,31	14,82	17,75	20,74	22,47	47,50
Brotli (Optimal)	2,33	10,04	21,65	40,43	47,04	49,32	61,49
Brotli (Smallest size)	3,45	12,96	57,53	353,10	564,19	852,79	1127,34
PPM	3,13	5,79	6,55	6,66	6,66	6,67	6,65

Рисунок 4.5 – Робота алгоритмів з використанням моделювання контексту на даних з ентропією 1 (рисунок виконано самостійно)



	100	1024	10240	102400	204800	512528	1049557
Huffman	3,70	5,66	5,96	5,99	5,99	5,99	5,99
LZ77	1,19	5,45	12,13	15,24	15,45	15,57	15,64
LZW	1,33	2,65	5,32	12,88	16,46	22,13	29,56
RLE	1,16	1,21	1,21	1,21	1,21	1,21	1,21

Рисунок 4.6 – Робота атомарних алгоритмів на даних з ентропією 1 (рисунок виконано самостійно)



	100	1024	10240	102400	204800	512528	1049557
BWT-MTF-Huffman	2,17	3,70	3,91	2,64	1,99	1,99	1,98
BWT-RLE	1,28	4,45	13,08	16,45	16,74	16,68	22,18
Deflate	1,35	4,92	12,06	17,59	18,54	18,93	18,47
Deflate (LZW)	0,79	1,91	4,52	12,33	15,90	21,40	28,72
MTF-LZ77	0,96	4,74	11,13	14,76	15,00	15,13	15,30
MTF-LZW	1,09	2,27	4,72	11,54	14,93	20,26	26,78
MTF-RLE	1,04	1,12	1,13	1,13	1,13	1,13	1,12
SharpZLib.BZip2	1,56	8,33	40,31	246,15	418,81	750,41	1340,43
System.IO.Compression.Deflate	2,38	10,89	36,83	61,39	62,21	64,10	90,23

Рисунок 4.7 – Робота складних алгоритмів на даних із ентропією 1 (рисунок виконано самостійно)

З отриманих експериментальних даних видно, що найкраще дані було стиснуто за допомогою алгоритму Brotli в режимі «Smallest size” і BZip2. Хоча алгоритм PPM також використовує моделювання контексту, його ефективність не є такою високою. Перетворення BWT значно покращило роботу алгоритма RLE, на відміну від попередніх результатів, проте застосування MTF не показує такої ж властивості. Через те, що дані кодуються вже не одним символом, ефективність оптимального кодування Хафмана різко знизилась, і її різниця з Deflate зростає разом із об'ємом даних для стиснення. Проте кодування Хафмана все ж залишилось найефективнішим для 100-байтного файлу.

Загалом, можна зауважити, що більшість алгоритмів дають коефіцієнти, що не відрізняються настільки ж, наскільки відрізнялись коефіцієнти стиснення даних з нульовою ентропією, за винятком Brotli та BZip2. Варто також зауважити, що стиснення LZW починаючи з розміру файлу 512 Кб було кращим за Deflate з використанням LZW. На позначці розміру файлу 200 Кб LZW теж перевищив LZ77 за коефіцієнтом стиснення, проте на інших даних LZ77 помітно кращий.

На рисунку 4.8 зображено час, витрачений алгоритмами на стиснення даних.

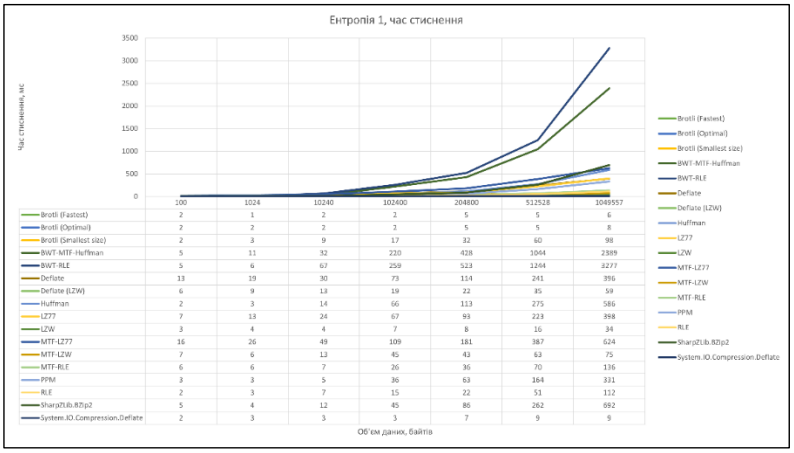


Рисунок 4.8 – Час стиснення даних (рисунок виконано самостійно)

Можна помітити, що відмінності між часом стиснення різних алгоритмів не є настільки значними, як при стисненні даних із ентропією 0. Це пов’язано з тим, що довжини послідовностей однакових символів зменшились, відповідно, зменшився й час їхньої обробки. Використання BWT помітно сповільнює роботу алгоритмів, хоча й не настільки помітно. Окрім цього, швидше зростає час стиснення алгоритмом LZ77, тоді як LZW працює за майже однаковий час. Помітно зріс час обробки алгоритмом BZip2, оскільки він також використовує BWT.

На рисунках 4.9-4.11 зображено результати експериментів із даними з ентропією 2.

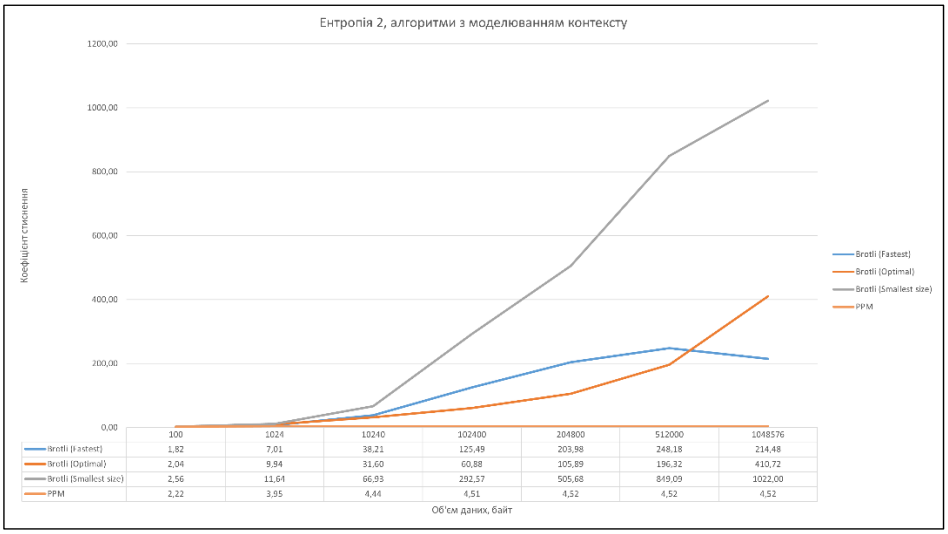
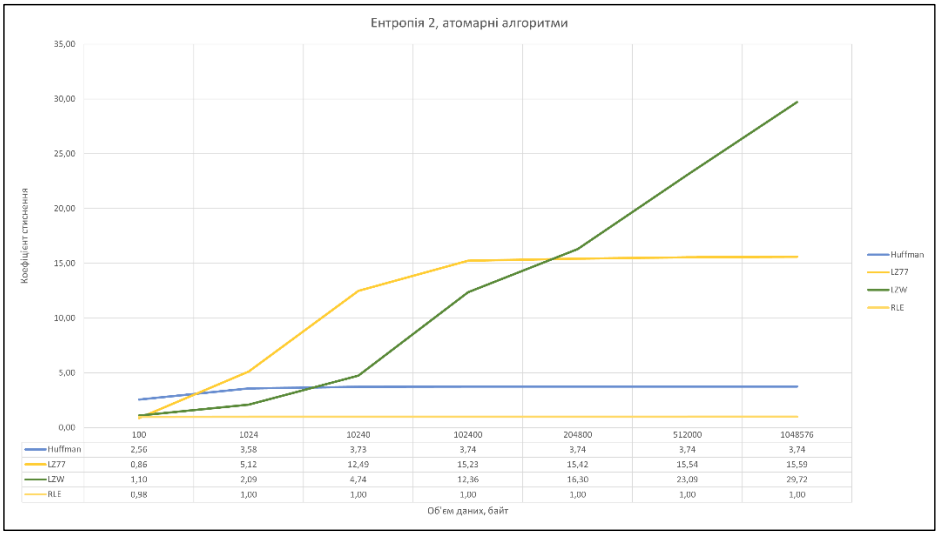
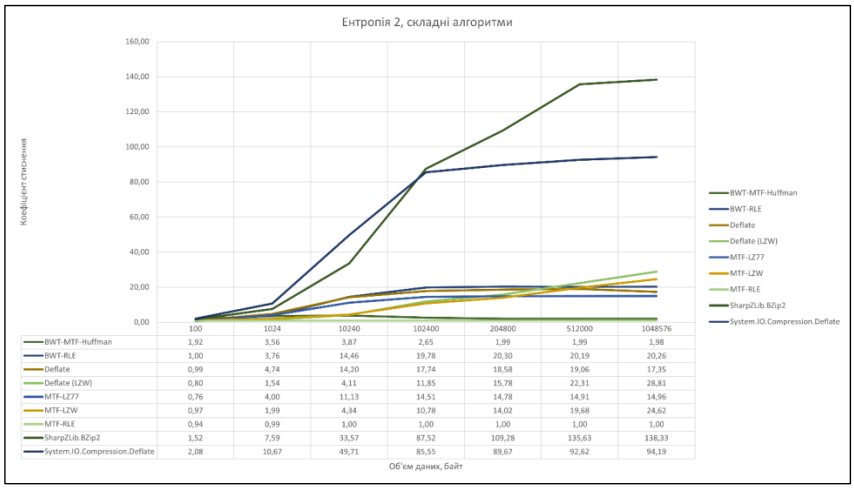


Рисунок 4.9 – Робота алгоритмів з використанням моделювання контексту на даних з ентропією 2 (рисунок виконано самостійно)



	100	1024	10240	102400	204800	512000	1048576
Huffman	2,56	3,58	3,73	3,74	3,74	3,74	3,74
LZ77	0,86	5,12	12,49	15,23	15,42	15,54	15,59
LZW	1,10	2,09	4,74	12,36	16,30	23,09	29,72
RLE	0,98	1,00	1,00	1,00	1,00	1,00	1,00

Рисунок 4.10. Робота атомарних алгоритмів на даних з ентропією 2 (рисунок виконано самостійно)



	100	1024	10240	102400	204800	512000	1048576
BWT-MTF-Huffman	1,92	3,56	3,87	2,65	1,99	1,99	1,98
BWT-RLE	1,00	3,76	14,46	19,78	20,30	20,19	20,26
Deflate	0,99	4,74	14,20	17,74	18,58	19,06	17,35
Deflate (LZW)	0,80	1,54	4,11	11,85	15,78	22,31	28,81
MTF-LZ77	0,76	4,00	11,13	14,51	14,78	14,91	14,96
MTF-LZW	0,97	1,99	4,34	10,78	14,02	19,68	24,62
MTF-RLE	0,94	0,99	1,00	1,00	1,00	1,00	1,00
SharpZLib.BZip2	1,52	7,59	33,57	87,52	109,28	135,63	138,33
System.IO.Compression.Deflate	2,08	10,67	49,71	85,55	89,67	92,62	94,19

Рисунок 4.11 – Робота складних алгоритмів на даних із ентропією 2 (рисунок виконано самостійно)

Можна помітити, що під час стиснення даних із середньою ентропією 2 помітно знизилась ефективність BZip2, хоча він працює приблизно на рівні Brotli в режимі «Optimal».

Перетворення BWT покращує стиснення RLE в 20 разів, тоді як MTF майже не впливає на нього. Для невеликих файлів, зокрема розміром 100 байтів та 1 кілобайт, з'являються випадки «шкідливого» стиснення, тобто його коефіцієнт менший за 1. Ефективність стиснення Хафмана без використання додаткових алгоритмів не перевищує 3,8, на відміну від попередніх експериментів.

На даному етапі можна спостерігати зменшення ефективності ентропійного кодування взагалі, а також збільшення ефективності словникових методів.

На рисунку 4.12 зображено графік витрат часу на стиснення даних із ентропією 2.

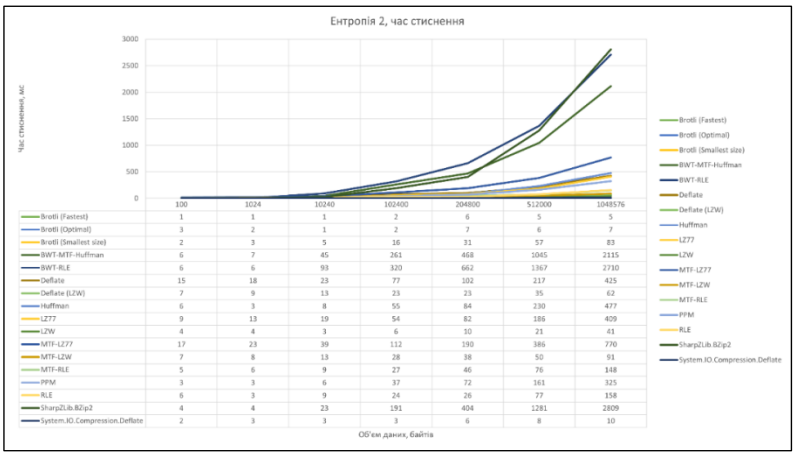


Рисунок 4.12 – Час стиснення даних (рисунок виконано самостійно)

Порівняно зі стисненням даних із ентропією 1, часові витрати значно збільшились, зокрема для BWT та BZip2. Так, стиснення BZip2 займає майже 3 секунди для файлу 1 Мб, тоді як Brotli в режимі «Smallest size», що дав найбільший коефіцієнт стиснення, стискає цей файл за 83 мс. Помітно зменшилась і швидкість роботи LZ77, тоді як LZW майже не сповільнився.

На рисунках 4.13-4.15 зображено результати експериментів із даними з ентропією, рівною 3.

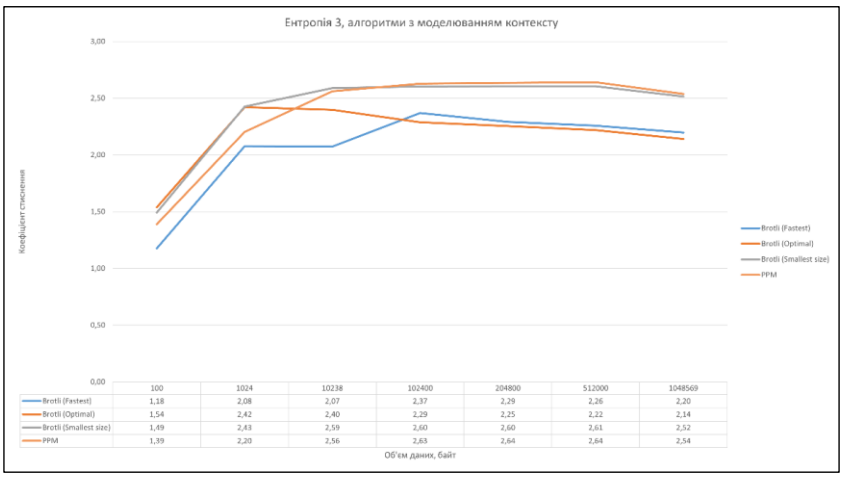
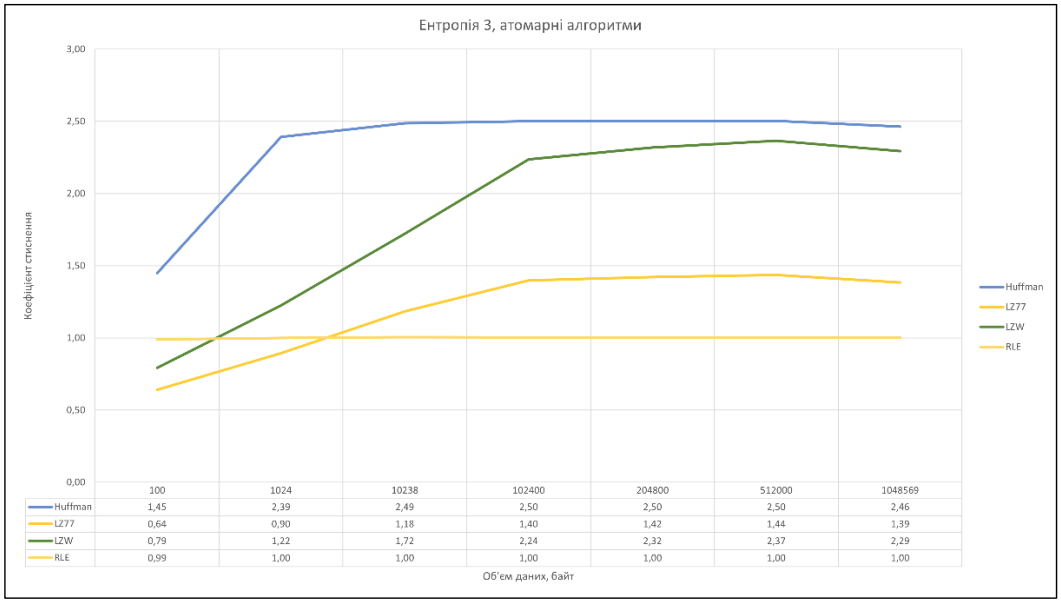
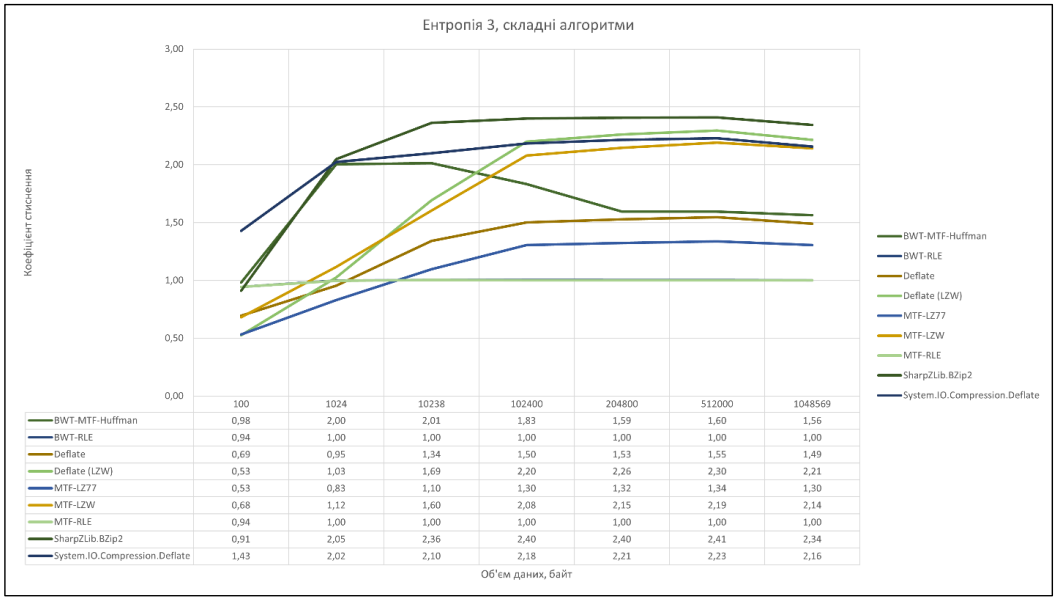


Рисунок 4.13 – Робота алгоритмів з використанням моделювання контексту на даних з ентропією 3 (рисунок виконано самостійно)



	100	1024	10238	102400	204800	512000	1048569
Huffman	1,45	2,39	2,49	2,50	2,50	2,50	2,46
LZ77	0,64	0,90	1,18	1,40	1,42	1,44	1,39
LZW	0,79	1,22	1,72	2,24	2,32	2,37	2,29
RLE	0,99	1,00	1,00	1,00	1,00	1,00	1,00

Рисунок 4.14 – Робота атомарних алгоритмів на даних з ентропією 3 (рисунок виконано самостійно)



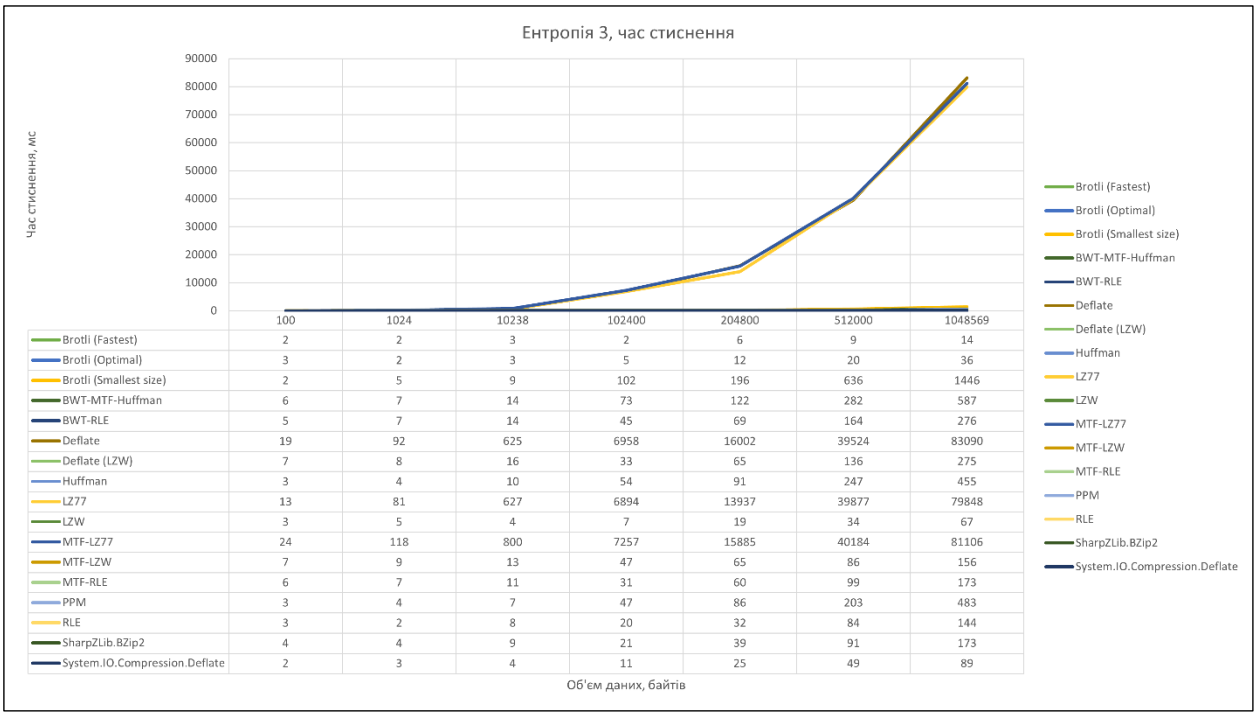
	100	1024	10238	102400	204800	512000	1048569
BWT-MTF-Huffman	0,98	2,00	2,01	1,83	1,59	1,60	1,56
BWT-RLE	0,94	1,00	1,00	1,00	1,00	1,00	1,00
Deflate	0,69	0,95	1,34	1,50	1,53	1,55	1,49
Deflate (LZW)	0,53	1,03	1,69	2,20	2,26	2,30	2,21
MTF-LZ77	0,53	0,83	1,10	1,30	1,32	1,34	1,30
MTF-LZW	0,68	1,12	1,60	2,08	2,15	2,19	2,14
MTF-RLE	0,94	1,00	1,00	1,00	1,00	1,00	1,00
SharpZLib.BZip2	0,91	2,05	2,36	2,40	2,40	2,41	2,34
System.IO.Compression.Deflate	1,43	2,02	2,10	2,18	2,21	2,23	2,16

Рисунок 4.15 – Робота складних алгоритмів на даних із ентропією 3 (рисунок виконано самостійно)

Можна помітити, що дані з ентропією 3 були стиснуті зі значно меншими коефіцієнтами, ніж попередні. Так, різниця між двома найбільшими коефіцієнтами складає близько 0,2. Також можна помітити, що дані розміром 512 Кб були стиснуті краще, ніж дані розміром 1 Мб. Багато алгоритмів утратили ефективність при роботі з даними, меншими за 10 Кб.

Також можна помітити, що перетворення BWT і MTF не покращили роботу алгоритма RLE. LZW був кращий за LZ77 під час роботи з усіма категоріями файлів, проте не давав набагато більших коефіцієнтів. Таким чином, можна помітити тенденцію до зменшення коефіцієнтів стиснення при збільшенні ентропії, а збільшення коефіцієнтів зі збільшенням об'єму даних стає менш значним.

На рисунку 4.16 зображено часові витрати на стиснення даних із ентропією 3:



	100	1024	10238	102400	204800	512000	1048569
Brotli (Fastest)	2	2	3	2	6	9	14
Brotli (Optimal)	3	2	3	5	12	20	36
Brotli (Smallest size)	2	5	9	102	196	636	1446
BWT-MTF-Huffman	6	7	14	73	122	282	587
BWT-RLE	5	7	14	45	69	164	276
Deflate	19	92	625	6958	16002	39524	83090
Deflate (LZW)	7	8	16	33	65	136	275
Huffman	3	4	10	54	91	247	455
LZ77	13	81	627	6894	13937	39877	79848
LZW	3	5	4	7	19	34	67
MTF-LZ77	24	118	800	7257	15885	40184	81106
MTF-LZW	7	9	13	47	65	86	156
MTF-RLE	6	7	11	31	60	99	173
PPM	3	4	7	47	86	203	483
RLE	3	2	8	20	32	84	144
SharpZLib.BZip2	4	4	9	21	39	91	173
System.IO.Compression.Deflate	2	3	4	11	25	49	89

Рисунок 4.16 – Час стиснення даних (рисунок виконано самостійно)

Можна помітити, що час стиснення за використання BWT значно зменшився порівняно з попереднім експериментом. Натомість набагато збільшився час стиснення за допомогою методу LZ77 і, відповідно, усіх методів, пов'язаних із ним. Також алгоритм Brotli в режимі «Smallest size» стиснув 1 Мб даних за приблизно 1,5 с, тоді як BZip2 спрацював набагато швидше.

На рисунках 4.17-4.19 зображено результати експериментів із даними з середньою ентропією 4 біти на символ.

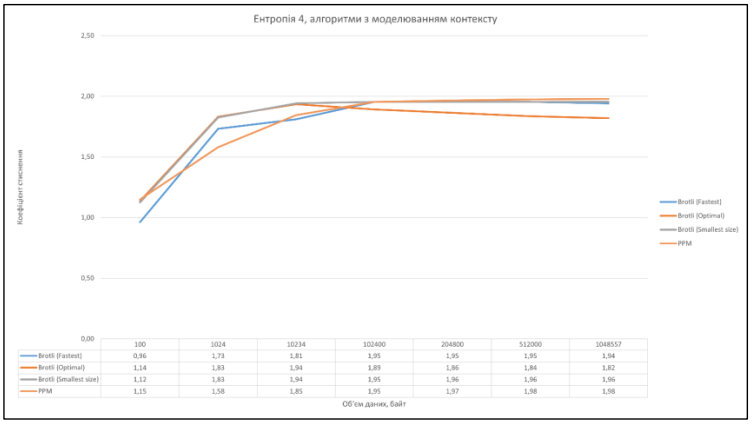


Рисунок 4.17 – Робота алгоритмів з використанням моделювання контексту на даних з ентропією 4 (рисунок виконано самостійно)

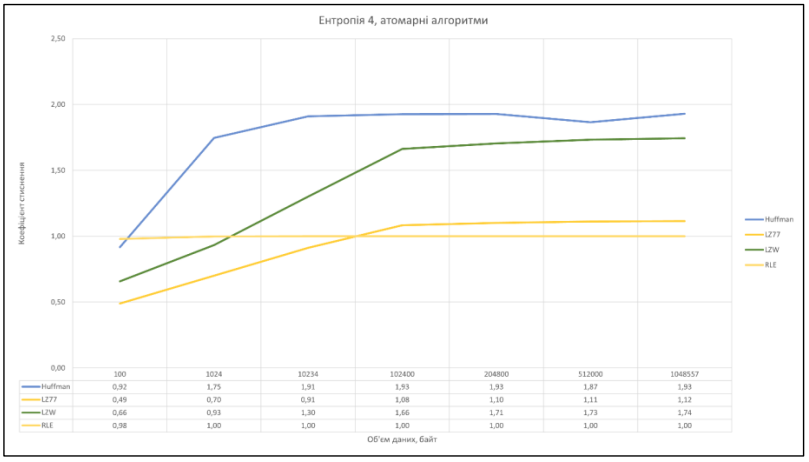


Рисунок 4.18 – Робота атомарних алгоритмів на даних з ентропією 4 (рисунок виконано самостійно)

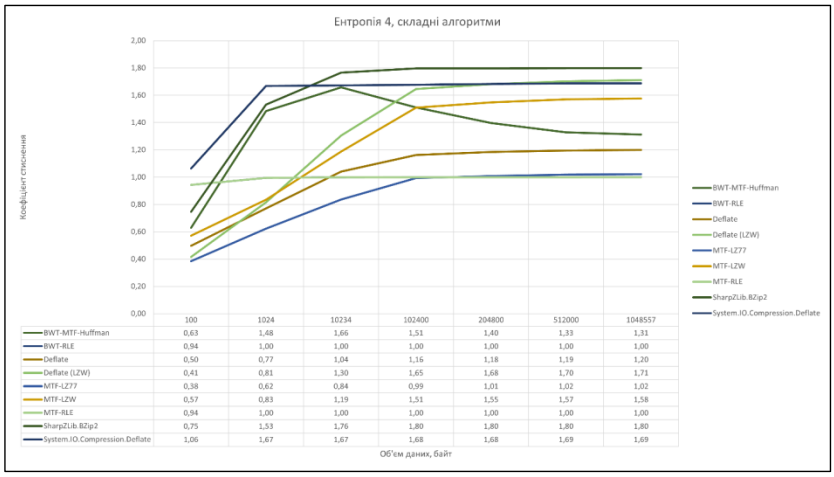


Рисунок 4.19 – Робота складних алгоритмів на даних із ентропією 3 (рисунок виконано самостійно)

З отриманих під час вищезазначених експериментів даних маємо, що коефіцієнти стиснення знизились, і жоден з алгоритмів не дав коефіцієнту, більшого за 2, при стисненні файлів із усіх категорій. Варто зауважити, що кодування Хафмана та алгоритми, що його використовують, спрацювали з майже тією ж ефективністю, що й Brotli та PPM. При використанні з даними об’ємом у 100 байтів лише вони та реалізація Deflate із бібліотеки System.IO.Compression змогли стиснути дані. При застосуванні з даними розміром 1 Мб оптимальне кодування Хафмана показало один із найкращих результатів.

Різниця між LZW та LZ77 також стала менш значущою, причому LZW дав більші коефіцієнти стиснення, ніж LZ77, хоча файли розміром 100 та 1024 байти не було стиснуто жодним із них. Використання разом із ними оптимального кодування Хафмана (алгоритми Deflate та Deflate (LZW) відповідно) для файлів меншого розміру дещо покращило коефіцієнти, але починаючи з даних розміром 100 Кб, LZW стискав дані краще, ніж Deflate із використанням LZW.

Приблизно таке ж порівняння можна провести й із реалізацією Deflate із бібліотеки System.IO.Compression, оскільки починаючи з даних об’ємом 200 Кб стиснення за допомогою LZW також давало дещо кращі коефіцієнти.

На рисунку 4.20 показано витрати часу на стиснення даних.

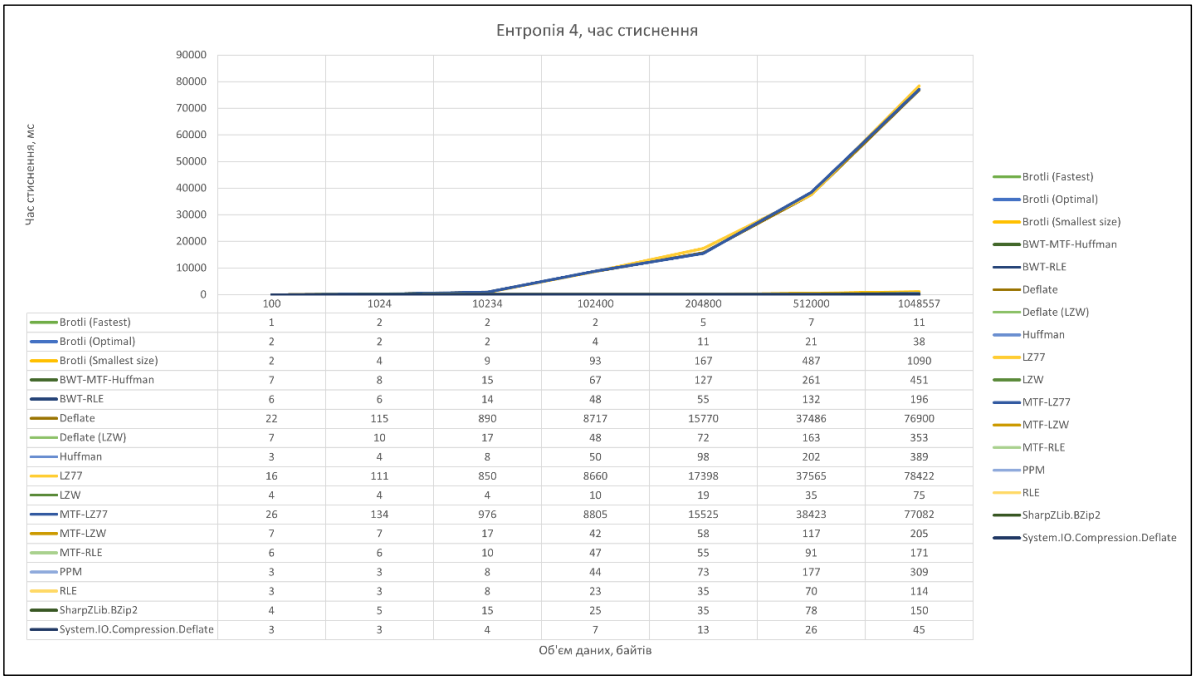


Рисунок 4.20 – Час стиснення даних (рисунок виконано самостійно)

Отримані дані не показують значних змін. Алгоритм LZ77 також помітно вирізняється за часом стиснення, проте працює на 5-7 секунд швидше, ніж під час стиснення даних із ентропією 3. Алгоритм BWT майже не сповільнює роботу RLE, проте й не дає значного збільшення коефіцієнтів стиснення.

Порівнюючи витрати часу на стиснення із коефіцієнтами, що дають відповідні алгоритми, можна зробити висновок, що більший час роботи не означає автоматично більшого коефіцієнту стиснення. Так, алгоритм Brotli працював близько секунди, проте не дав набагато більшого коефіцієнту, ніж BZip2.

На рисунках 4.21-4.23 зображено роботу алгоритмів із даними з ентропією 5.

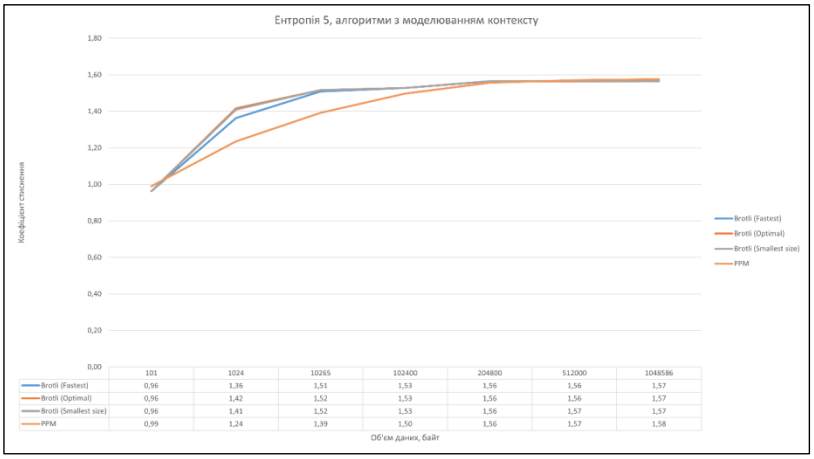


Рисунок 4.21 – Робота алгоритмів з використанням моделювання контексту на даних з ентропією 5 (рисунок виконано самостійно)

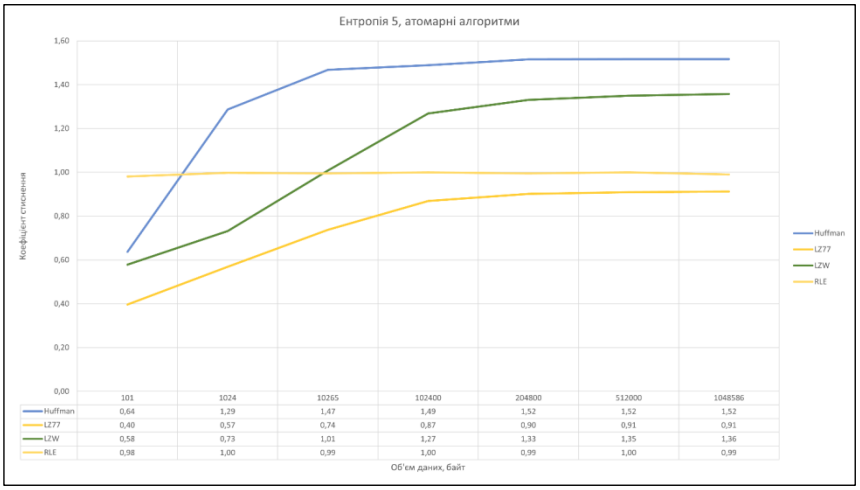


Рисунок 4.22 – Робота атомарних алгоритмів на даних з ентропією 5 (рисунок виконано самостійно)

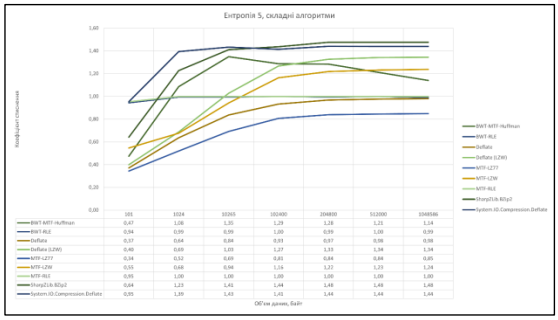


Рисунок 4.23 – Робота складних алгоритмів на даних із ентропією 5 (рисунок виконано самостійно)

Дані, отримані під час експерименту, показують, що жоден алгоритм не зміг стиснути файл розміром 100 байтів та ентропією 5 бітів на символ. Найвищі коефіцієнти дали алгоритми, що використовують моделювання контексту, тобто Brotli та PPM, а також кодування Хафмана. Можна помітити, що, оскільки алгоритм BZip2 включає в себе кодування Хафмана, його коефіцієнт стиснення також є близьким до нього. Інші алгоритми не показують суттєвих змін: їхні коефіцієнти стиснення порівняно з даними з ентропією 4 біти на символ зменшуються.

Помітним є той факт, що більше половини складних методів стиснення не змогли виконати стиснення файлів, менших за 10 Кб, також деяка частина алгоритмів виконала «шкідливу роботу». Оскільки ентропія близько 5 бітів на символ є досить розповсюдженою, можна попередньо зробити висновок про меншу універсальність таких алгоритмів.

На рисунку 4.24 показано час, використаний для стиснення даних алгоритмами.

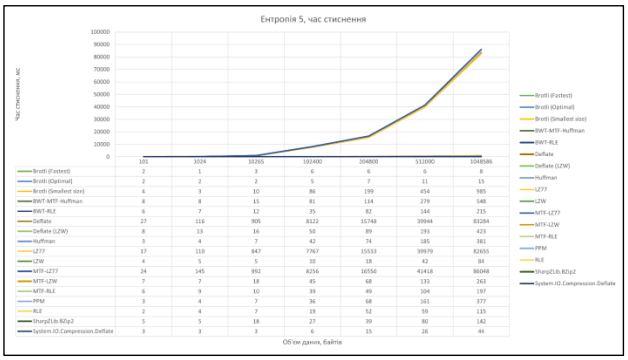
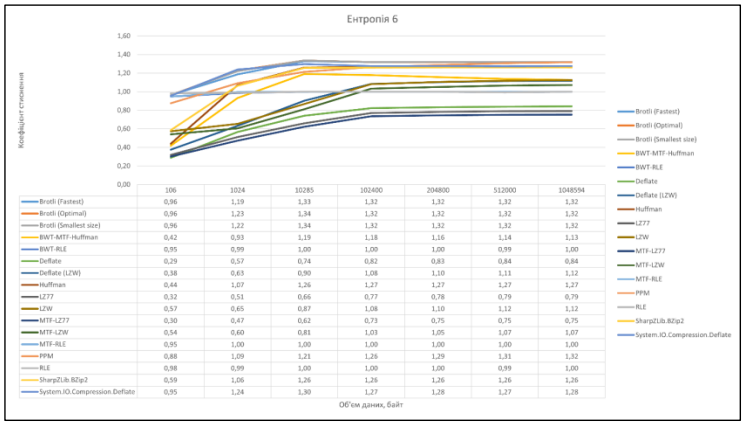


Рисунок 4.24 – Час стиснення даних (рисунок виконано самостійно)

Дані про час стиснення також не показують суттєвих змін порівняно із попередніми експериментами. Помітно зростає час стиснення за допомогою алгоритму LZ77, причому LZW залишається доволі швидким, хоча й не дає високих коефіцієнтів.

На рисунку 4.25 зображено роботу алгоритмів із даними з середньою ентропією 6 бітів на символ.

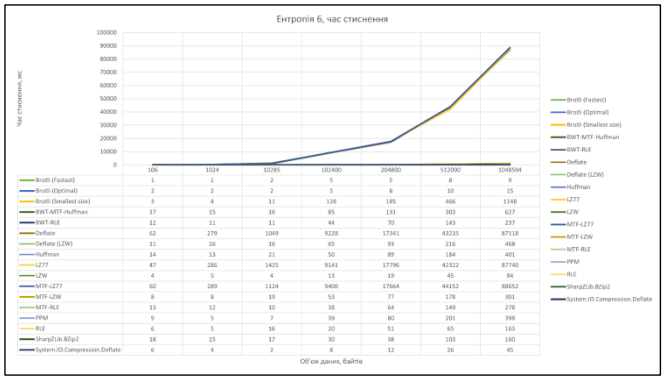


	106	1024	10285	102400	204800	512000	1048594
Brotli (Fastest)	0,96	1,19	1,33	1,32	1,32	1,32	1,32
Brotli (Optimal)	0,96	1,23	1,34	1,32	1,32	1,32	1,32
Brotli (Smallest size)	0,96	1,22	1,34	1,32	1,32	1,32	1,32
BWT-MTF-Huffman	0,42	0,93	1,19	1,18	1,16	1,14	1,13
BWT-RLE	0,95	0,99	1,00	1,00	1,00	0,99	1,00
Deflate	0,29	0,57	0,74	0,82	0,83	0,84	0,84
Deflate (LZW)	0,38	0,63	0,90	1,08	1,10	1,11	1,12
Huffman	0,44	1,07	1,26	1,27	1,27	1,27	1,27
LZ77	0,32	0,51	0,66	0,77	0,78	0,79	0,79
LZW	0,57	0,65	0,87	1,08	1,10	1,12	1,12
MTF-LZ77	0,30	0,47	0,62	0,73	0,75	0,75	0,75
MTF-LZW	0,54	0,60	0,81	1,03	1,05	1,07	1,07
MTF-RLE	0,95	1,00	1,00	1,00	1,00	1,00	1,00
PPM	0,88	1,09	1,21	1,26	1,29	1,31	1,32
RLE	0,98	0,99	1,00	1,00	1,00	0,99	1,00
SharpZLib.BZip2	0,59	1,06	1,26	1,26	1,26	1,26	1,26
System.IO.Compression.Deflate	0,95	1,24	1,30	1,27	1,28	1,27	1,28

Рисунок 4.25 – Робота алгоритмів із даними з ентропією 6 (рисунок виконано самостійно)

Найбільший коефіцієнт дало стиснення за допомогою алгоритмів Brotli та PPM. Варто зауважити, що в усіх трьох режимах Brotli давав майже однакові коефіцієнти, а коефіцієнт стиснення алгоритмом Хафмана повторював коефіцієнт реалізації Deflate у стандартній бібліотеці .NET. Інші алгоритми показали менші коефіцієнти, не набагато вищі за 1, зокрема LZW та поєднання BWT, MTF та кодування Хафмана.

На рисунку 4.26 зображено графік витрат часу алгоритмами під час стиснення даних.

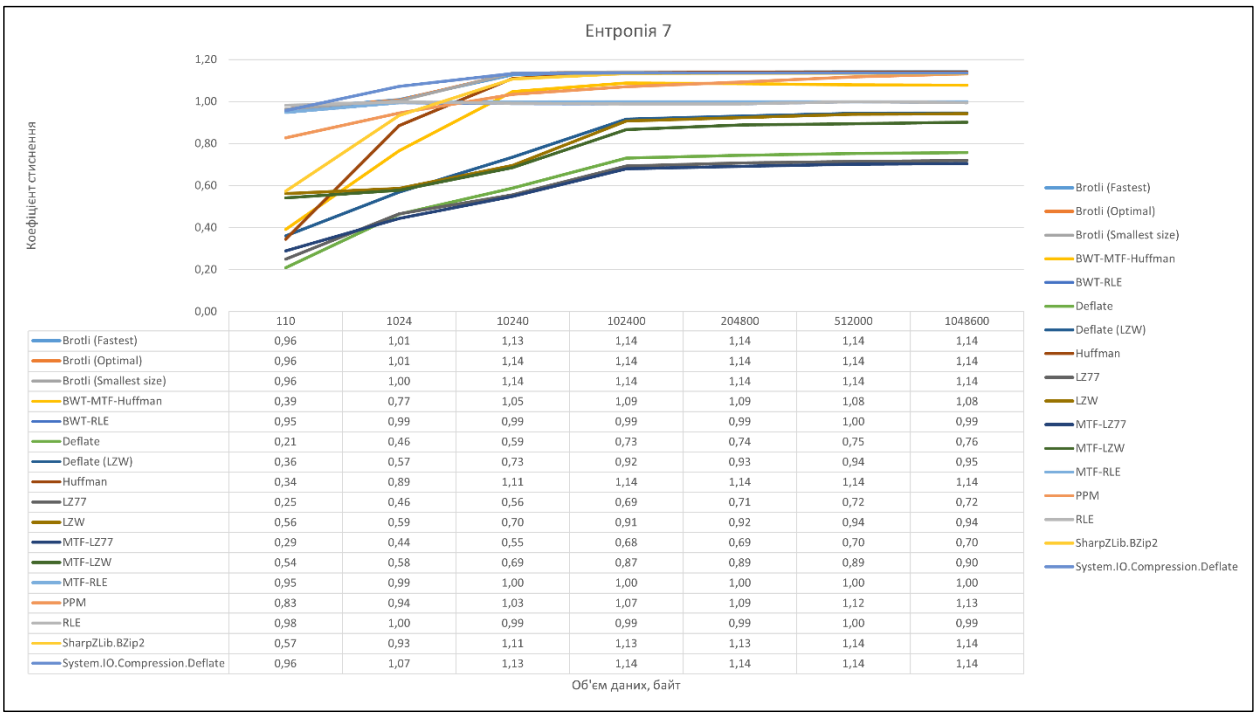


	106	1024	10285	102400	204800	512000	1048594
Brotli (Fastest)	1	1	2	5	8	8	9
Brotli (Optimal)	2	2	2	5	8	10	15
Brotli (Smallest size)	3	4	11	128	185	466	1148
BWT-MTF-Huffman	17	15	16	85	131	303	627
BWT-RLE	12	11	11	44	70	143	237
Deflate	62	279	1069	9228	17341	43235	87118
Deflate (LZW)	11	16	16	65	93	216	468
Huffman	14	13	21	50	89	184	401
LZ77	47	286	1425	9141	17796	43222	87740
LZW	4	5	4	13	19	45	94
MTF-LZ77	60	289	1124	9400	17664	44152	88652
MTF-LZW	8	8	19	53	77	178	301
MTF-RLE	13	12	10	38	64	149	278
PPM	9	5	7	39	80	201	398
RLE	6	5	16	20	51	65	163
SharpZLib.BZip2	18	15	17	30	38	103	160
System.IO.Compression.Deflate	6	4	2	8	12	26	45

Рисунок 4.26 – Час стиснення даних (рисунок виконано самостійно)

На рисунку 4.26 помітне збільшення часу, що був витрачений усіма алгоритмами, проте суттєві зміни відсутні. Видно, що Brotli в режимі «Smallest size» витрачає близько 1 секунди на стиснення даних із ентропією, більшою за 2 біти на символ.

На рисунку 4.27 зображено результати проведення експериментів із даними з ентропією 7 бітів на символ. Можна помітити, що найменший файл має розмір 110 байтів, щоб забезпечити ентропію, близьку до 7.



	110	1024	10240	102400	204800	512000	1048600
Brotli (Fastest)	0,96	1,01	1,13	1,14	1,14	1,14	1,14
Brotli (Optimal)	0,96	1,01	1,14	1,14	1,14	1,14	1,14
Brotli (Smallest size)	0,96	1,00	1,14	1,14	1,14	1,14	1,14
BWT-MTF-Huffman	0,39	0,77	1,05	1,09	1,09	1,08	1,08
BWT-RLE	0,95	0,99	0,99	0,99	0,99	1,00	0,99
Deflate	0,21	0,46	0,59	0,73	0,74	0,75	0,76
Deflate (LZW)	0,36	0,57	0,73	0,92	0,93	0,94	0,95
Huffman	0,34	0,89	1,11	1,14	1,14	1,14	1,14
LZ77	0,25	0,46	0,56	0,69	0,71	0,72	0,72
LZW	0,56	0,59	0,70	0,91	0,92	0,94	0,94
MTF-LZ77	0,29	0,44	0,55	0,68	0,69	0,70	0,70
MTF-LZW	0,54	0,58	0,69	0,87	0,89	0,89	0,90
MTF-RLE	0,95	0,99	1,00	1,00	1,00	1,00	1,00
PPM	0,83	0,94	1,03	1,07	1,09	1,12	1,13
RLE	0,98	1,00	0,99	0,99	0,99	1,00	0,99
SharpZLib.BZip2	0,57	0,93	1,11	1,13	1,13	1,14	1,14
System.IO.Compression.Deflate	0,96	1,07	1,13	1,14	1,14	1,14	1,14

Рисунок 4.27 – Робота алгоритмів із даними з ентропією 7 (рисунок виконано самостійно)

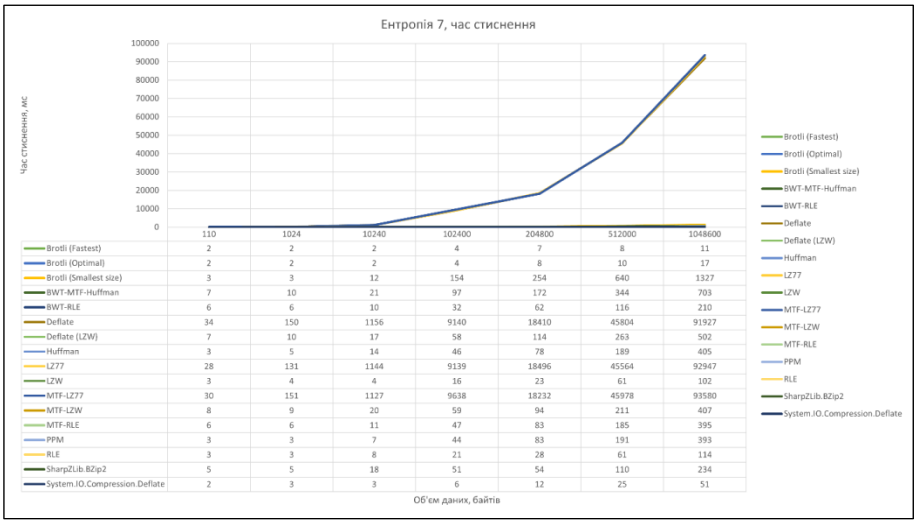
На графіку можна помітити, що більшість алгоритмів не змогла стиснути жоден із файлів, лише збільшивши його розмір на виході.

Можна помітити, що зі збільшенням ентропії після 5 бітів на символ алгоритм Хафмана та Deflate працюють із такими ж коефіцієнтами, що й Brotli та PPM, причому алгоритм PPM не зміг стиснути дані об’ємом 100 і 1024 байти.

BWT та MTF фактично не дають збільшення коефіцієнту для алгоритмів Хафмана та RLE, оскільки за такого значення ентропії дані є майже випадковими. Відповідно, стає складніше знайти повторювані послідовності, і, якщо вони все ж знаходяться, вони компенсуються більшими витратами на метадані для

визначення блоків, кодування дерев Хафмана тощо. Варто зауважити, що за великих значень ентропії алгоритм RLE дає коефіцієнти стиснення, близькі до 1, незалежно від розміру файлу.

На рисунку 4.28 зображено графік витрат часу на стиснення даних.



	110	1024	10240	102400	204800	512000	1048600
Brotli (Fastest)	2	2	2	4	7	8	11
Brotli (Optimal)	2	2	2	4	8	10	17
Brotli (Smallest size)	3	3	12	154	254	640	1327
BWT-MTF-Huffman	7	10	21	97	172	344	703
BWT-RLE	6	6	10	32	62	116	210
Deflate	34	150	1156	9140	18410	45804	91927
Deflate (LZW)	7	10	17	58	114	263	502
Huffman	3	5	14	46	78	189	405
LZ77	28	131	1144	9139	18496	45564	92947
LZW	3	4	4	16	23	61	102
MTF-LZ77	30	151	1127	9638	18232	45978	93580
MTF-LZW	8	9	20	59	94	211	407
MTF-RLE	6	6	11	47	83	185	395
PPM	3	3	7	44	83	191	393
RLE	3	3	8	21	28	61	114
SharpZLib.BZip2	5	5	18	51	54	110	234
System.IO.Compression.Deflate	2	3	3	6	12	25	51

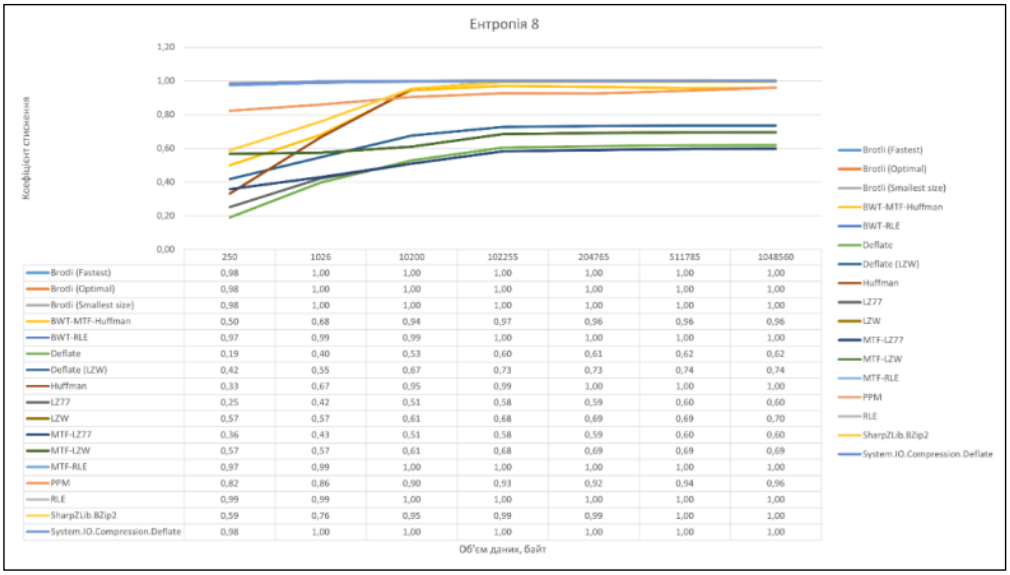
Рисунок 4.28 – Час стиснення даних (рисунок виконано самостійно)

На графіку можна побачити, що стиснення LZ77 займає більше часу, досягаючи 1,5 хв. Це можна пояснити, зокрема, особливостями конкретної реалізації даного алгоритму.

Проаналізувавши час, що витрачається алгоритмом Deflate зі стандартної бібліотеки .NET, можна зрозуміти, що стиснення цим алгоритмом займає часом менше, ніж LZW. Оскільки Deflate включає в себе LZ77, такий же висновок можна зробити й про нього.

Загалом ситуація, зображена на графіку, доволі подібна до попередніх експериментів. Можна спостерігати певну залежність часу стиснення від об'єму даних, причому для словникових алгоритмів показники часу часто зростають швидше, ніж для ентропійних.

На рисунку 4.29 зображено результати проведення експериментів із даними з ентропією 8 бітів на символ. Аналогічно, для забезпечення ентропії, близької до 8, найменший файл має розмір 250 байтів, проте такий розмір файлу все одно достатньо далекий від 1024 байтів для того, аби бути поміщеним у наступну категорію за розміром.

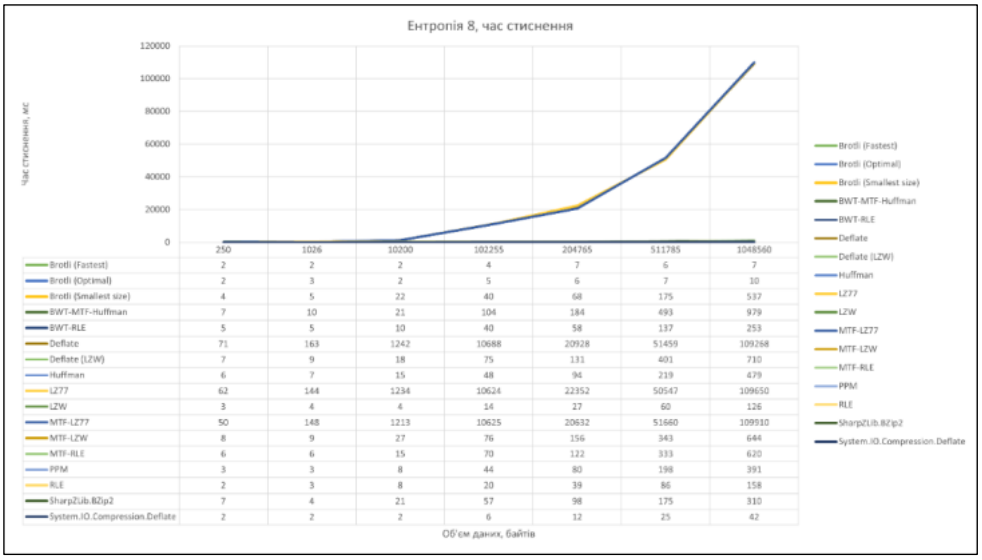


	250	1026	10200	102255	204765	511785	1048560
Brotli (Fastest)	0,98	1,00	1,00	1,00	1,00	1,00	1,00
Brotli (Optimal)	0,98	1,00	1,00	1,00	1,00	1,00	1,00
Brotli (Smallest size)	0,98	1,00	1,00	1,00	1,00	1,00	1,00
BWT-MTF-Huffman	0,50	0,68	0,94	0,97	0,96	0,96	0,96
BWT-RLE	0,97	0,99	0,99	1,00	1,00	1,00	1,00
Deflate	0,19	0,40	0,53	0,60	0,61	0,62	0,62
Deflate (LZW)	0,42	0,55	0,67	0,73	0,73	0,74	0,74
Huffman	0,33	0,67	0,95	0,99	1,00	1,00	1,00
LZ77	0,25	0,42	0,51	0,58	0,59	0,60	0,60
LZW	0,57	0,57	0,61	0,68	0,69	0,69	0,70
MTF-LZ77	0,36	0,43	0,51	0,58	0,59	0,60	0,60
MTF-LZW	0,57	0,57	0,61	0,68	0,69	0,69	0,69
MTF-RLE	0,97	0,99	1,00	1,00	1,00	1,00	1,00
PPM	0,82	0,86	0,90	0,93	0,92	0,94	0,96
RLE	0,99	0,99	1,00	1,00	1,00	1,00	1,00
SharpZLib.BZip2	0,59	0,76	0,95	0,99	0,99	1,00	1,00
System.IO.Compression.Deflate	0,98	1,00	1,00	1,00	1,00	1,00	1,00

Рисунок 4.29 – Робота алгоритмів із даними з ентропією 8 (рисунок виконано самостійно)

За отриманими результатами можна зрозуміти, що жодному з алгоритмів не вдалося стиснути файли. Це пов’язано, зокрема, з тим, що ентропія 8 бітів на символ означає, що дані не мають надлишковості, або вона є мінімальною. За теоремою про джерело кодування, будь-яка кількість байтів N із ентропією 8 бітів, або 1 байт на символ, може бути закодована лише в N байтів без утрат даних. Отже, даними з ентропією 8 бітів на символ досягнуто ліміту на стиснення без втрат, що й підтверджується результатами експериментів.

На рисунку 4.30 зображено графік часу роботи алгоритмів.



	250	1026	10200	102255	204765	511785	1048560
Brotli (Fastest)	2	2	2	4	7	6	7
Brotli (Optimal)	2	3	2	5	6	7	10
Brotli (Smallest size)	4	5	22	40	68	175	537
BWT-MTF-Huffman	7	10	21	104	184	493	979
BWT-RLE	5	5	10	40	58	137	253
Deflate	71	163	1242	10688	20928	51459	109268
Deflate (LZW)	7	9	18	75	131	401	710
Huffman	6	7	15	48	94	219	479
LZ77	62	144	1234	10624	22352	50547	109650
LZW	3	4	4	14	27	60	126
MTF-LZ77	50	148	1213	10625	20632	51660	109310
MTF-LZW	8	9	27	76	156	343	644
MTF-RLE	6	6	15	70	122	333	620
PPM	3	3	8	44	80	198	391
RLE	2	3	8	20	39	86	158
SharpZLib.BZip2	7	4	21	57	98	175	310
System.IO.Compression.Deflate	2	2	2	6	12	25	42

Рисунок 4.30 – Час стиснення даних (рисунок виконано самостійно)

За даними графіку помітно, що алгоритм Brotli в режимі «Smallest size» працював удвічі швидше за попередні експерименти.

Також можна помітити пришвидшення алгоритму Deflate із бібліотеки System.IO.Compression, а також алгоритму LZW, хоча вони не виконали корисної роботи.

4.2 Тестування роботи алгоритмів із реальними даними

Для тестування алгоритмів також було використано такі файли:

– текст «Lorem ipsum»;
– зображення у форматі BMP;
– зображення у форматі SVG;
– веб-сторінка у форматі HTML;
– програмний код на мові програмування C#;
– відео в форматі AVI;
– файл MS Excel у форматі XLSX.

Ці дані мають різний розмір, середню ентропію та формат. Таким чином, відбулось експериментальне стиснення даних, що представляють різні формати та є згенерованими реальними програмами для подальшого використання, або є наближеними до натуральної мови.

Для кожного файлу було проаналізовано час, витрачений кожним алгоритмом на стиснення, і коефіцієнт стиснення. Із кожним файлом, як і в випадку тестових даних, було проведено експерименти тричі. З трьох експериментів для кожного файлу було обрано найменші значення часу для мінімізації впливу сторонніх чинників.

Експерименти зі стисненням реальних даних мають на меті продемонструвати відмінності між форматами тестових даних та тих, що застосовуються користувачами або додатками в реальних випадках бізнес-логіки, розробки тощо.

На рисунку 4.31 зображено графік коефіцієнтів стиснення текстового файлу за допомогою різних алгоритмів. Розмір файлу – 100620 байтів, формат – TXT.

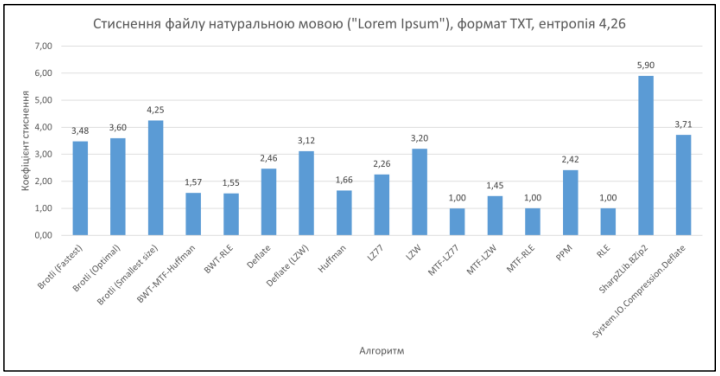


Рисунок 4.31 – Стиснення текстового файлу (рисунок виконано самостійно)

Виходячи з результатів експерименту, найвищі коефіцієнти були отримані за використання методів BZip2 та Brotli. Можна помітити, що коефіцієнти є значно вищими за ті, що були отримані під час стиснення тестових даних. Це може бути пов'язано з тим, що в тексті використовуються частоти символів, що значно відрізняються від тестових та відповідають частотам у реальних текстах.

На рисунку 4.32 зображено витрати часу на стиснення файлу.

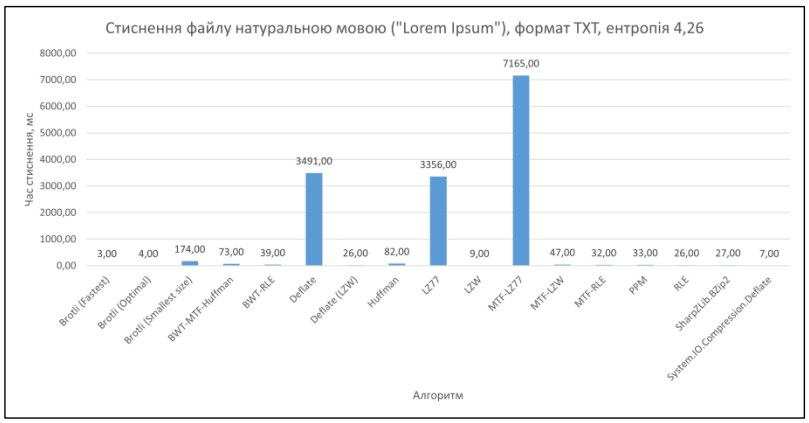


Рисунок 4.32 – Час стиснення файлу (рисунок виконано самостійно)

Графік часових витрат на стиснення файлу не відрізняється суттєво від графіку стиснення тестових даних зі схожою ентропією та розміром. Можна побачити, що алгоритм LZ77 працює значно довше за інші, також помітно виділяється Brotli в режимі «Smallest size».

Наступний файл для стиснення – зображення в форматі BMP розміром близько 320 Кб та з середньою ентропією 7,58 бітів на символ. Графік результатів експериментів із ним зображено на рисунку 4.33.

Рисунок 4.33 – Стиснення файлу BMP (рисунок виконано самостійно)

На графіку помітно, що через доволі високу ентропію дані не було стиснуто більше ніж у 1,55 рази. Крім Brotli та BZip2 доволі високий у даному контексті коефіцієнт показали також алгоритми Deflate та PPM. Різниця між Brotli в режимах «Fastest» та «Optimal» є набагато нижчою, ніж між ними ж та «Smallest size».

Можна також помітити, що в цьому випадку алгоритми BWT та MTF покращили стиснення RLE навіть за ентропії 7,6 бітів на символ.

Це доводить, що коефіцієнт стиснення залежить також від інших факторів, зокрема формату файлу: для подібного випадку в тестових даних алгоритми мали набагато нижчі коефіцієнти стиснення.

На рисунку 4.34 зображено графік часових витрат на стиснення файлу.

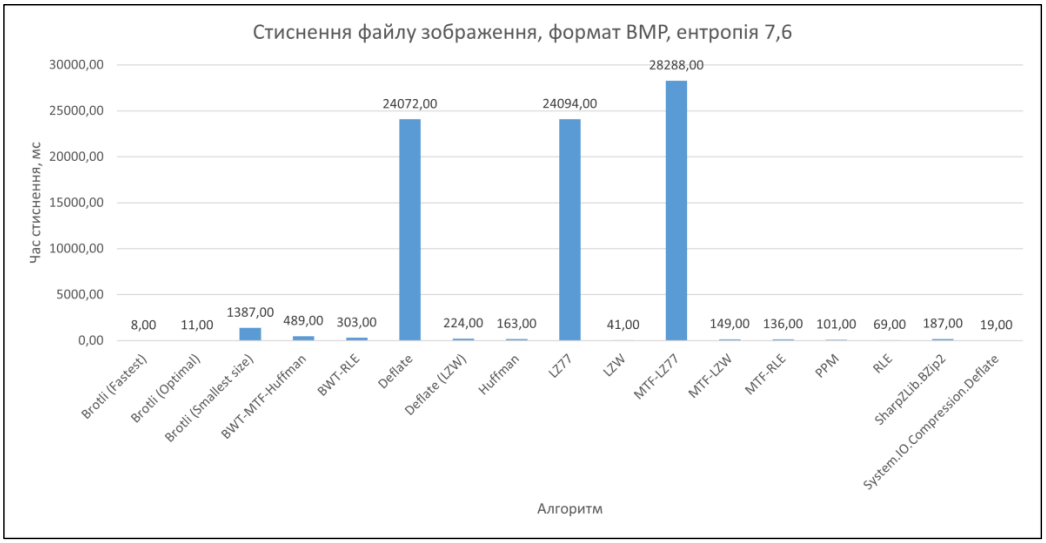


Рисунок 4.34 – Час стиснення файлу (рисунок виконано самостійно)

З даних на графіку можна спостерігати, що часові витрати алгоритму LZ77 є меншими ніж з експериментальними даними. Також можна помітити збільшення часу роботи Brotli та алгоритмів, що використовують BWT.

На рисунку 4.35 зображено результати експериментів зі стиснення іншого файлу – векторного зображення в форматі SVG. Воно має розмір близько 21 Кб та середню ентропію 5,19 бітів на символ.

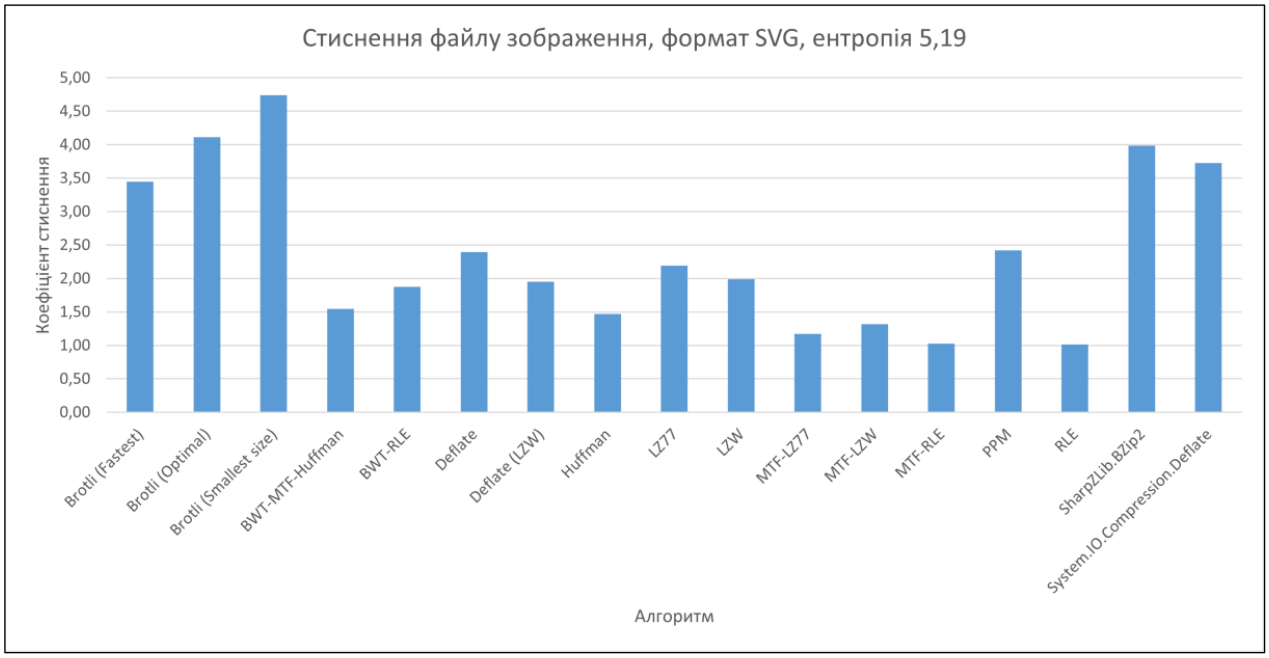


Рисунок 4.35 – Стиснення векторного зображення SVG (рисунок виконано самостійно)

З даних на графіку можна зауважити значну перевагу алгоритму Brotli. Це можна пояснити тим, що цей алгоритм першочергово розроблявся для стиснення саме веб-сторінок та файлів CSS. Формат SVG має структуру, що базується на XML, як і сторінки формату HTML, саме тому Brotli працює з такими даними краще.

Алгоритм BWT покращив роботу RLE для стиснення цих даних, але алгоритм MTF майже не зробив корисної роботи.

Алгоритм LZ77 спрацював краще, ніж LZW, причому при додатковому застосуванні алгоритму Хафмана коефіцієнти стиснення обох алгоритмів також зросли.

На рисунку 4.36 зображено графік часу стиснення файлу.

Рисунок 4.36 – Графік часу стиснення файлу (рисунок виконано самостійно)

Окрім LZ77, часові витрати загалом не відрізняються. Помітно вирізняється алгоритм Brotli в режимі «Smallest size», а також поєднання алгоритмів BWT, MTF та кодування Хафмана. На порівняно невеликий час стиснення даних вплинув, зокрема, й їхній розмір.

На рисунку 4.37 зображено графік стиснення HTML-сторінки. Її розмір – 360 Кб, середня ентропія – 5,74 біти на символ.

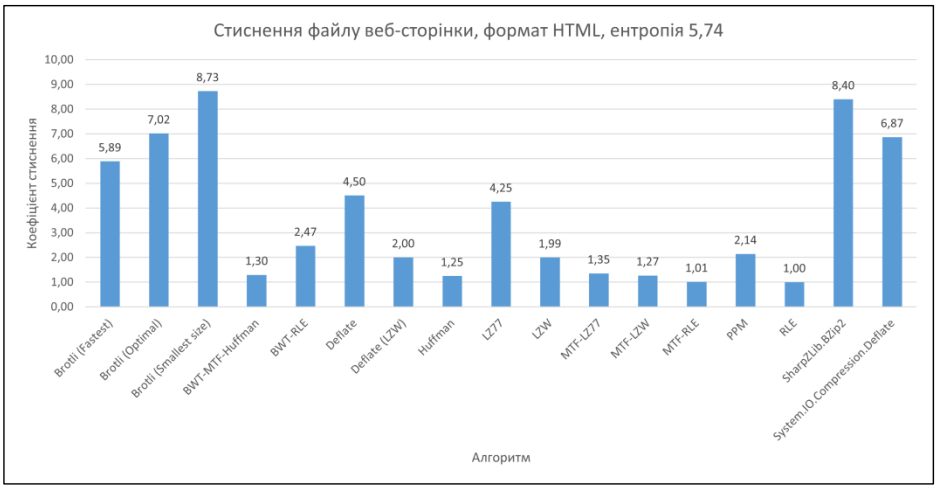


Рисунок 4.37 – Стиснення HTML-сторінки (рисунок виконано самостійно)

За отриманими даними, Brotli та BZip2 дають найвищі коефіцієнти стиснення. Brotli було розроблено саме для стиснення таких форматів, як HTML та програмний код, тому, як і в випадку з SVG, він показує значні результати.

BZip2, натомість, сильніше залежить від ентропії та розміру словника, використаного для створення даних. Варто також зауважити, що алгоритм BWT теж покращив стиснення RLE, проте на стиснення за допомогою оптимального кодування Хафмана майже не вплинув. Попередня обробка за допомогою MTF на стиснення RLE мала незначний вплив, а LZW дав менший коефіцієнт стиснення.

Також можна помітити значне підвищення ефективності алгоритму LZ77 порівняно з LZW, що може свідчити про більш сильну залежність цього алгоритму від формату даних та розміру словника.

Алгоритм PPM стиснув дані приблизно в два рази, що доводить певний зв’язок між символами у форматі HTML.

На рисунку 4.38 зображено графік часу стиснення файлу різними алгоритмами.

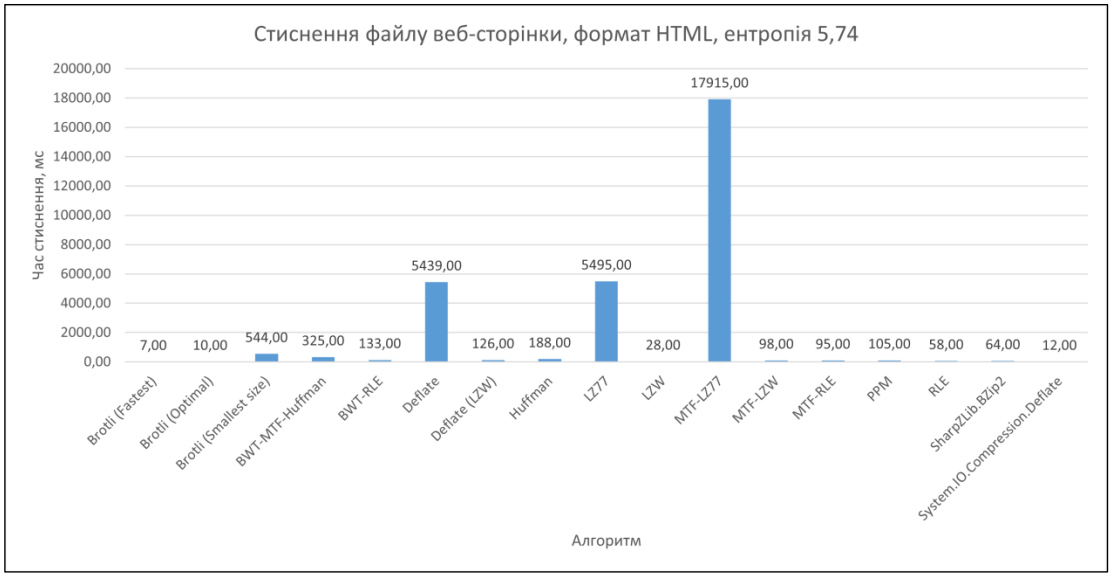


Рисунок 4.38 – Час стиснення даних (рисунок виконано самостійно)

З даних про час стиснення файлу можна зрозуміти, що загалом пропорції часових витрат для даних зі схожою ентропією зберігаються. Помітним є стиснення Brotli та поєднання алгоритмів BWT, MTF та кодування Хафмана які, окрім LZ77, виконувались найдовше. Причому використання алгоритмів BWT та MTF збільшило часові витрати майже вдвічі.

На рисунку 4.39 зображено графік стиснення програмного коду на мові C#, об'єм якого – 12 Кб, середня ентропія – 4,32 біта на символ.

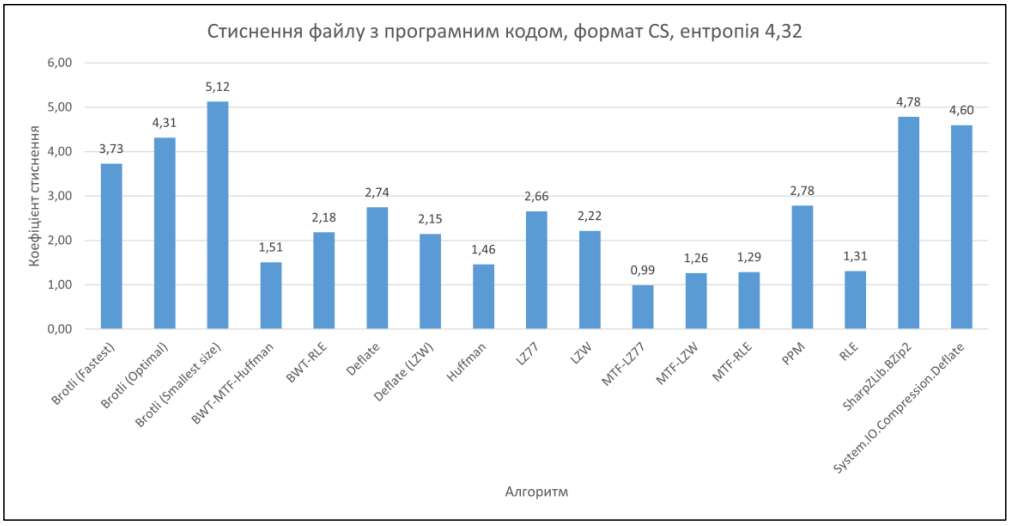


Рисунок 4.39 – Стиснення коду мовою C# (рисунок виконано самостійно)

З отриманих даних можна помітити, що найвищі коефіцієнти відрізняються менше, ніж при стисненні попередніх файлів. Так, алгоритми Brotli, BZip2 та Deflate стиснули дані з найвищими коефіцієнтами. Можна також помітити алгоритми PPM та LZ77, що теж виділяються. Алгоритм BWT зміг покращити стиснення методом RLE, проте майже не вплинув на стиснення методом Хафмана. Вплив додавання до LZ77 та LZW стиснення Хафмана менш значний, ніж у попередніх експериментах.

На рисунку 4.40 зображено графік, що представляє результати експериментів зі стиснення відеофайлу формату AVI розміром 740 Кб та середньою ентропією 3,71 бітів на символ.

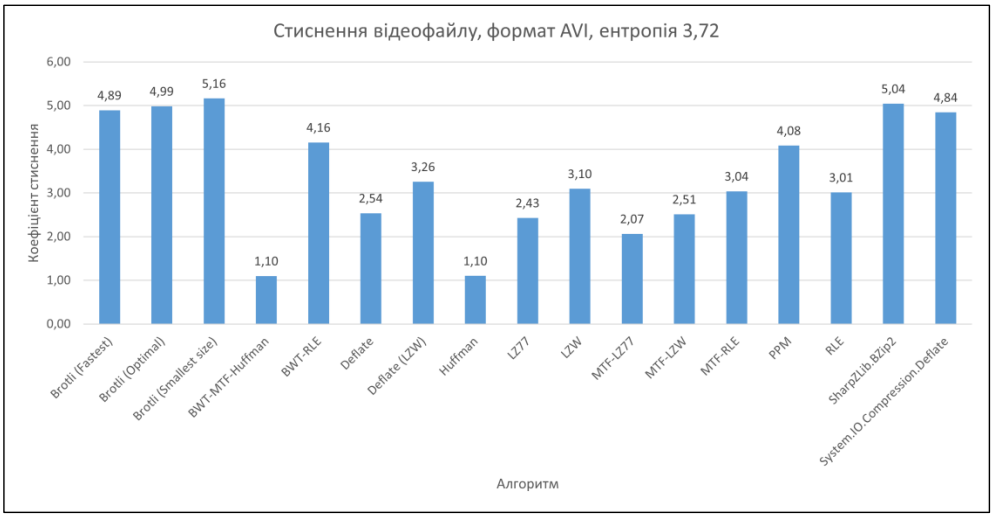


Рисунок 4.40 – Стиснення файлу формату AVI (рисунок виконано самостійно)

З отриманих під час експериментів даних помітно, що різниця між трьома режимами Brotli зменшилась порівняно з попередніми експериментами. Можна помітити суттєвий рівень стиснення, досягнутий за допомогою поєднання алгоритмів BWT та RLE. Також високі рівні стиснення показали алгоритми BZip2 та Deflate.

Алгоритм LZW, що на практиці використовується для стискання медіафайлів (зокрема форматів GIF та TIFF [https://www.martinreddy.net/gfx/2d/GIF-comp.txt]), показав вищий коефіцієнт стиснення, ніж LZ77. Через доволі низьке значення середньої ентропії порівняно високе значення стиснення надав і PPM.

На рисунку 4.41 зображено графік часу стиснення файлу різними алгоритмами.

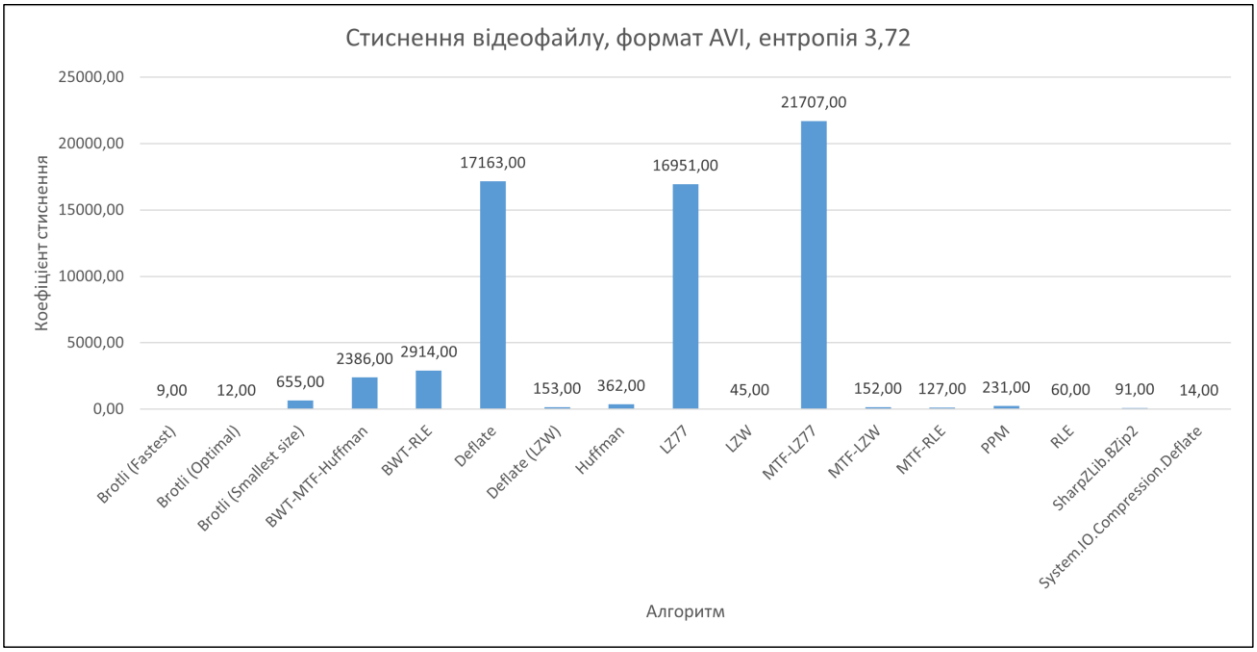


Рисунок 4.41 – Час стиснення даних (рисунок виконано самостійно)

На графіку можна помітити, що алгоритм BWT працює помітно довше за попередні експерименти, зокрема ті, в яких він мав менший вплив на коефіцієнт стиснення. Алгоритм LZ77 працює набагато довше порівняно з іншими алгоритмами, проте час роботи реалізації алгоритму Deflate зі стандартної бібліотеки .NET є одним із найменших. Незважаючи на те, що три режими роботи

Brotli дають майже однакові коефіцієнти, режим «Smallest size» все ж працює набагато довше, ніж «Optimal» та «Fastest».

На рисунку 4.42 зображено графік стиснення даних у форматі MS Excel із середньою ентропією 7,61 та розміром 3 Кб.

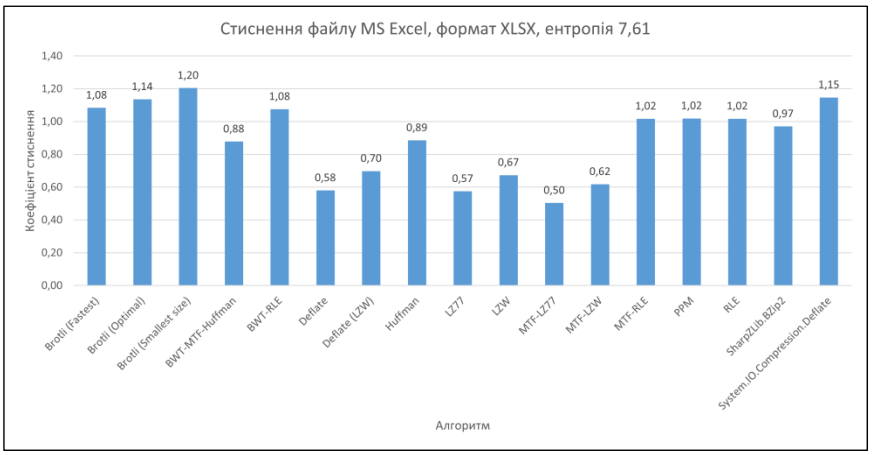


Рисунок 4.42. Стиснення файлу XLSX (рисунок виконано самостійно)

З отриманих даних видно, що найвищі коефіцієнти є майже однаковими: 1,20 за допомогою алгоритму Brotli та 1,15 за допомогою алгоритму Deflate. Можна також помітити порівняно велику кількість алгоритмів, що не виконали саме стиснення даних, зокрема BZip2 та кодування Хафмана, які в попередніх експериментах часто давали одні з найвищих коефіцієнтів. Це пов’язано з тим, що файл має доволі високу ентропію і вже є стиснутим майже максимально.

На рисунку 4.43 зображено графік часу роботи алгоритмів із даним файлом.

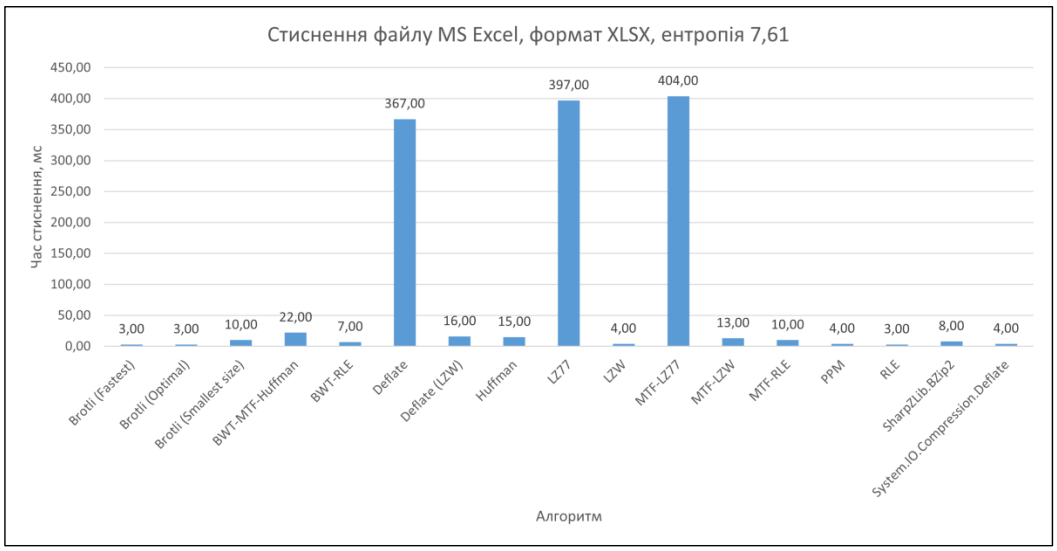


Рисунок 4.43 – Час стиснення файлу (рисунок виконано самостійно)

Маємо, що більшість алгоритмів працювала доволі швидко, порівняно з іншими експериментами, оскільки незважаючи на високу ентропію, розмір файлу невеликий і для більшості алгоритмів займає менш ніж один блок.

4.3 Висновки з результатів експериментів

Проаналізувавши результати експериментів, можна дійти висновку, що коефіцієнт стиснення деякого повідомлення дійсно залежить від його середньої ентропії, проте також експерименти показали, що формат даних є важливим, особливо для алгоритмів із застосуванням моделювання контексту, як Brotli або PPM. Так, Brotli дав набагато вищий коефіцієнт стиснення файлу у форматі HTML, ніж згенерованого тестового файлу з подібною ентропією.

Алгоритми з застосуванням моделювання контексту (Brotli, PPM) добре показували себе під час стиснення різноманітних даних, проте Brotli давав особливо високі коефіцієнти під час стиснення веб-сторінок та коду, що відповідає його прямому призначенню. Алгоритм PPM показував кращі результати під час використання його з текстовими даними, близькими до натуральних мов.

Варто зазначити, що алгоритм Хафмана доволі часто перевершував алгоритми LZW та LZ77 за більших розмірів даних та значень ентропії, більших за 3. Це може бути пов’язано з тим, що кодування Хафмана стискає дані на рівні бітів, майже повністю нівелюючи надлишковість, тоді як обидва словникових алгоритми часто можуть формувати надлишкові дані з великою кількістю повторів символів. При стисненні реальних даних поєднання оптимального кодування Хафмана з LZW або LZ77 підвищує коефіцієнт стиснення за порівняно невеликого збільшення витрат часу. Велику різницю в часі стиснення та загалом великі його значення для LZ77 можна пояснити особливостями реалізації, оскільки вона не адаптована для обробки даних у паралельних блоках.

Алгоритм RLE в чистому вигляді дав найвищий коефіцієнт стиснення для тестового файлу з ентропією 0 розміром 100 байтів, що доводить його простоту, але часом ефективну реалізацію. Помітно, що RLE, на відміну від інших компресорів, майже не давав коефіцієнтів, нижчих за 1 навіть під час стиснення даних із

ентропією, близькою до 8. Зазвичай під час стиснення реальних даних RLE поступався коефіцієнтом стиснення алгоритмам LZW та LZ77, проте застосування BWT часто підвищувало коефіцієнт стиснення. Натомість MTF мав значно менший вплив під час стиснення як згенерованих, так і реальних даних.

Реалізація BZip2 із бібліотеки SharpZipLib показала високу ефективність порівняно з Brotli, проте вона в багатьох випадках витрачає більше оперативної пам’яті та часу. Відповідно, це робить BZip2 менш придатним для використання браузерами, на відміну від Brotli.

ВИСНОВКИ

В ході виконання науково-дослідницької роботи було розглянуто алгоритми стиснення без втрат, що найактивніше використовуються в сучасних застосунках для стиснення даних. Деякі з цих алгоритмів є складовими частинами більш складних методів, таких як Deflate, що працюють у декілька етапів. Також було з'ясовано, що незважаючи на те, що деякі алгоритми, як, наприклад, LZ77, були розроблені доволі давно й мають новіші варіації, вони все одно продовжують використовуватись у сучасних алгоритмах.

Було складено план експериментальних досліджень, де визначено основні характеристики роботи алгоритмів: коефіцієнт стиснення, час кодування й декодування інформації, використання оперативної пам'яті. Також було визначено критерії порівняння тестових даних, такі як середня ентропія повідомлення та розмір файлу. Тестові файли було згенеровано за відповідними категоріями, а також узято реальні текстові файли, веб-сторінки, зображення та відео.

Для проведення експериментальних досліджень було складено сценарії використання алгоритмів та їхніх поєднань із різними даними, розроблено базу даних для зберігання результатів. Дані, отримані під час експериментів, було використано в програмній реалізації. Також із отриманих даних зроблено висновки про особливості роботи алгоритмів, що досліджувались.

За результатами наукового дослідження було опубліковано тези доповіді «Дослідження алгоритмів оптимального кодування і стиснення даних» на 28-й Міжнародний молодіжний форум «Радіоелектроніка і молодь у XXI столітті», що наведені в Додатку Б.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Large Text Compression Benchmark. URL: http://mattmahoney.net/dc/text.html (дата звернення: 23.01.2024).

2. FLAC Documentation. URL: https://xiph.org/flac/documentation_format_overview.html (дата звернення: 25.01.2024).

3. Byzkrovnyi, O., Chupryna, A., Smelyakov, K., Sharonova, N., Repikhov, V. Comparison of Object Detection Algorithms for the Task of Detecting Possible Road Accident. CEUR Workshop Proceedings, 2023, 3387, pp. 13–28

4. A Universal Algorithm for Sequential Data Compression. URL: https://courses.cs.duke.edu/spring03/cps296.5/papers/ziv_lempel_1977_universal_algorithm.pdf (дата звернення: 25.01.2024).

5. Cleary J., Witten I. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*. 1984. Vol. 32, no. 4. P. 396–402. URL: https://doi.org/10.1109/tcom.1984.1096090 (дата звернення: 25.01.2024).

6. Alakuijala J. Comparison of Brotli, Deflate, Zopfli, LZMA, LZHAM and Bzip2 compression algorithms. URL: https://cran.r-project.org/web/packages/brotli/vignettes/brotli-2015-09-22.pdf (дата звернення: 05.02.2024).

7. RFC 1951: DEFLATE Compressed Data Format Specification version 1.3. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc1951 (date of access: 12.06.2024).

8. Khairi N. A., Jambek A. B. Run-Length encoding (RLE) data compression algorithm. *International Journal of Nanoelectronics and Materials*. 2021. Vol. 14.

9. Khairi N. A., Jambek A. B. Run-Length encoding (RLE) data compression algorithm. *International Journal of Nanoelectronics and Materials*. 2021. Vol. 14.

10. Burrows M., Wheeler D. A block sorting lossless data compression algorithm. Digital Equipment Corporation, 1994.

11. A locally adaptive data compression scheme / J. L. Bentley et al. *Communications of the ACM*. 1986. Vol. 29, no. 4. P. 320–330. URL: https://doi.org/10.1145/5684.5688 (date of access: 12.06.2024).

12. Alakuijala J., Szabadka Z. Brotli compressed data format. RFC Editor, 2016. URL: https://doi.org/10.17487/rfc7932 (дата звернення: 02.02.2024).

13. Jang H., Kim C., Lee J. W. Practical speculative parallelization of variable-length decompression algorithms. *ACM SIGPLAN Notices*. 2013. Vol. 48, no. 5. P. 55–64. URL: https://doi.org/10.1145/2499369.2465557 (date of access: 12.06.2024).

14. Cleary J. G. Unbounded length contexts for PPM. *The Computer Journal*. 1997. Vol. 40, no. 2 and 3. P. 67–75. URL: https://doi.org/10.1093/comjnl/40.2_and_3.67 (дата звернення: 15.02.2024).

15. Wilmhoff R. C. An information interchange code for communications. *Proceedings of the IEEE*. 1981. Vol. 69, no. 7. P. 842–844. URL: https://doi.org/10.1109/proc.1981.12086 (дата звернення: 18.02.2024).

16. McKay D. J. C. Information theory, inference & learning algorithms. Cambridge, UK : Cambridge University Press, 2003. 640 p.

17. Shannon C. E. A mathematical theory of communication. *Bell System Technical Journal*. 1948. Vol. 27, no. 4. P. 623–656. URL: https://doi.org/10.1002/j.1538-7305.1948.tb00917.x (дата звернення: 19.02.2024).

18. Gorman B. L. Asynchronous Data Operations and Multiple Database Contexts. *Practical Entity Framework*. Berkeley, CA, 2020. P. 575–623. URL: https://doi.org/10.1007/978-1-4842-6044-9_14 (date of access: 12.06.2024).

19. Metrics applicable for evaluating software at the design stage / I. Gruzdo та ін. *5th International Conference on Computational Linguistics and Intelligent Systems. Volume I: Main Conference*, м. Lviv, 22–23 квіт. 2021 р. 2021. С. 916–936. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85107237353&partnerID=40&md5=2acca556132c7c02af11bc95e7b5d4ec.