УДК 004.627 DOi 10.30837/ bi.2021.1(96).07

О.С. Валлас[1], О.В. Вечур[2]

[1]ХНУРЕ, м. Харків, Україна, oleh.vallas@nure.ua, ORCID iD: 0000-0003-4043-3946

[2]ХНУРЕ, м. Харків, Україна, alexander.vechur@nure.ua, ORCID iD: 0000-0001-9605-1475

МЕТОДИ ПОШУКУ ТА КОДУВАННЯ СХОЖИХ ПОСЛІДОВНОСТЕЙ ДАНИХ В АЛГОРИТМАХ СТИСНЕННЯ ДАНИХ БЕЗ ВТРАТ

Розглянуто методи пошуку та кодування схожих послідовностей даних, та їх використання для покращення алгоритмів стиснення даних без втрат. Досліджено сучасні підходи до пошуку послідовностей з неточним збігом – тривіальні та евристичні методи, індексні методи та методи, що базуються на N-грамах. Розглянуто підходи кодування відмінностей з використанням відстані Левенштейна та Геммінга. Запропонована розширена структура алгоритму стиснення даних. Комбінації вищезазначених методів у складі запропонованої структури було протестовано на двох датасетах – датасеті англійського тексту «enwik8» та комбінованому датасеті «Silesia Corpus». При тестування оцінювались ступінь стиснення, швидкість кодування та декодування, та загальний баланс. У результаті було розроблено нову структуру алгоритмів стиснення даних та виявлено найбільш ефективні комбінації методів для компресії різних типів даних.

СТИСНЕННЯ ДАНИХ БЕЗ ВТРАТ, СХОЖІ ПОСЛІДОВНОСТІ ДАНИХ, ІНДЕКСНІ МЕТОДИ, N-ГРАМИ, ВІДСТАНЬ ЛЕВЕНШТЕЙНА, ВІДСТАНЬ ГЕММІНГА, КОДУВАННЯ ВІДМІННОСТЕЙ

Валлас О.С., Вечур О.В. Методы поиска и кодирования похожих последовательностей данных в алгоритмах сжатия данных без потерь. Рассмотрены методы поиска и кодирования похожих последовательностей данных и их использование для улучшения алгоритмов сжатия данных без потерь. Исследованы современные подходы к поиску последовательностей с неточным совпадением – тривиальные и эвристические методы, индексные методы и методы, основанные на N-граммах. Рассмотрены подходы кодирования различий с использованием расстояния Левенштейна и Хэмминга. Предложена расширенная структура алгоритма сжатия данных. Комбинации вышеупомянутых методов в составе предложенной структуры были протестированы на двух датасетах – датасете английского текста «enwik8» и комбинированном датасете «Silesia Corpus». При тестировании оценивались степень сжатия, скорость кодирования и декодирования, а также общий баланс. В результате была разработана новая структура алгоритмов сжатия данных, и выявлены наиболее эффективные комбинации методов для компрессии различных типов данных.

СЖАТИЕ ДАННЫХ БЕЗ ПОТЕРЬ, ПОХОЖИЕ ПОСЛЕДОВАТЕЛЬНОСТИ ДАННЫХ, ИНДЕКСНЫЕ МЕТОДЫ, N-ГРАММЫ, РАССТОЯНИЕ ЛЕВЕНШТЕЙНА, РАССТОЯНИЕ ХЭММИНГА, КОДИРОВАНИЕ РАЗЛИЧИЙ

Vallas O.S., Vechur O.V. Methods of search and encoding similar data sequences in lossless data compression algorithms. Considered methods for searching and encoding approximate string matchings and their application to lossless data compression algorithms improvement. Studied modern approaches to the search for similar data sequences – trivial and heuristic methods, indexing methods and methods based on N-grams. Considered approaches of encoding differences using Levenshtein and Hamming distances. Proposed an extended structure of the data compression algorithm. Combinations of the above methods as part of the proposed structure were tested on two datasets – the English text dataset «enwik8» and the combined dataset «Silesia Corpus». During testing, ccompression ratio, encoding and decoding speed and overall balance were evaluated. As a result, a new structure of data compression algorithms was developed and the most effective combinations of methods for compression were identified for different data types.

LOSSLESS DATA COMPRESSION, APPROXIMATE STRING MATCHING, INDEXING METHODS, N-GRAMS, LEVENSHTEIN DISTANCE, HAMMING DISTANCE, DIFFERENCE ENCODING

Вступ

Стиснення даних – одне з найстаріших і фундаментальних напрямків у всій сфері інформаційних технологій. На поточний момент обсяг даних в світі оцінюють більш ніж 10^{18} байт і щороку ця кількість збільшується на 30%. Ці дані представляють собою найрізноманітнішу інформацію – текст на різних мовах, зображення, мультимедіа, відеоролики, документи, файли вихідного коду і безліч інших поширених або вузько спеціалізованих форматів. І практично для всіх способів зберігання і типів даних використовується стиснення. Стиснення даних дозволяє значно скоротити обсяг пам'яті, займаний одними або іншими даними і використовується повсюдно у всіх без винятку сферах діяльності людини – від побутових приладів до космічних апаратів.

Усі підходи та алгоритми стиснення даних можна поділити на дві категорії – стиснення даних без втрат та стиснення з втратами. Стиснення без втрат (англ. lossless compression) — метод стиснення даних, при використанні якого закодована інформація може бути повністю відновлена зі стиснутих даних. Навпаки, стиснення з втратами дозволяє лише відновлення даних, які є тільки наближенням до початкових даних. Найчастіше використовується саме стиснення без втрат, адже користувачеві потрібно отримати інформацію саме в такому вигляді, якою вона була до стиснення. Стиснення з втратами

використовується в основному в медіафайлах – зображеннях, аудіо, відео тощо. Тому що в цих файлах часто важливо передати лише загальний огляд і не треба відновлювати інформацію з точністю до окремого пікселю або ноти. В цій роботі будуть розглянуть підходи саме стиснення даних без втрат.

На сьогодні існує дуже багато різних підходів для роботи з даними. Різноманітні алгоритми і структури даних дозволяють робити багатопрофільну обробку та пришвидшити різні операції роботи з даними. Сучасний напрямок обробки природньої мови (NLP) пропонує різні підходи аналізу даних на основі аналізу реальної мови людини. Популярні області аналізу даних та машинного навчання пропонують адаптивні методи будь-якої обробки даних на основі здатності комп'ютерів ітеративно покращувати результат, «навчаючись» на попередніх результатах. Усі ці сфери дають майже нескінченну множину комбінацій, які потенційно можуть бути використані для стиснення даних.

Метою даної роботи є дослідження методів та алгоритмів пошуку схожих послідовностей даних та можливостей застосування цих методів для покращення алгоритмів стиснення даних без втрат. Будуть проаналізовані різні комбінації цих методів у складі існуючих алгоритмів стиснення даних, зокрема алгоритмів родини LZ* [1]. Також будуть розглянуть підходи до оптимального кодування знайдених послідовностей для покращення ефективності стиснення і швидкості кодування та декодування даних. Для аналізу та порівняння результатів буде розроблена метрика та зібрані репрезентативні датасети, представлені різними типами даних. Для автоматизації порівняння буде розроблено набір програмних методів роботи з різними комбінаціями алгоритмів.

1. Базова структура алгоритму стиснення даних без втрат

Практично усі алгоритми стиснення даних без втрат базуються на простій загальній схемі, запропонованій [1], яка отримала назву родина алгоритмів LZ77». Ця схема складається з декількох послідовних етапів, зображених на рисунку 1 (зліва). На першому кроці відбувається пошук закономірностей в даних, далі ці закономірності кодуються так, щоб їх можна було відновити на етапі декодування. В кінці до отриманої послідовності застосовуються додаткові оптимізації, найчастіше – побайтове оптимальне кодування, в результаті якого виходить стиснена послідовність даних. В якості алгоритму оптимального кодування зазвичай вибирають кодування Гаффмана [2] або арифметичне кодування. Обидва алгоритми кодують байти оптимальним способом, тому цей крок не представляє інтересу і використовується однаково практично у всіх алгоритмах стиснення. Основна частина стиснення зосереджена на етапі 2 і 3, а саме в пошуку закономірностей в даних і їх кодуванні. Автори оригінального алгоритму LZ77 пропонують досить примітивні методи пошуку закономірностей – простий пошук однакових послідовностей в деякому фіксованому вікні даних. Знайдені повторення кодуються парами (відстань; довжина), а нові послідовності просто записуються копіюванням.

Рис. 1. Базова (зліва) та розширена (справа) схеми алгоритмів стиснення даних без втрат

Пізніше було запропоновано велику кількість більш розвинутих технік, що поліпшують ефективність оригінального алгоритму LZ77 – розбиття даних на незалежні блоки [3], оптимальне арифметичне кодування отриманої послідовності [4], застосування дельта-фільтрів [5], динамічна оптимізація параметрів пошуку. Сучасна версія алгоритму LZ77 і його ефективні оптимізація розглядаються в [6]. Однак всі ці алгоритми засновані на пошуку і кодування точних збігів послідовностей. У даній роботі ж розглядаються більш складні підходи з використанням кодувань схожих послідовностей, які не обов'язково повністю співпадають.

Вперше використання часткового співпадіння рядків для кодування даних було запропоновано в публікації «Universal data compression based on approximate string matching» [7]. Автори доповнюють вищевказану схему, пропонуючи шукати закономірності в даних шляхом реалізації динамічного словника з підтримкою неточного пошуку. У статті наведено доказ того, що при досить великих вхідних даних, алгоритм завжди сходиться до теоретично оптимального ступеня стиснення. Однак, в даній статті розглядається стиснення з втратами (lossy compression), тому в запропонованій схемі відсутній етап кодування відмінностей між послідовностями.

На основі проведеного аналізу, була розроблена конкретизація оригінальної схеми алгоритму стиснення даних, яка зображена на рисунку 1 (справа). Розроблена схема акцентує увагу на двох основних етапах – пошуку послідовностей з неточним збігом і ефективне кодування відмінностей. В якості останнього етапу було обрано кодування Гаффмана, яке дозволяє оптимально закодувати байти отриманої на попередніх етапах послідовності і має безліч застосувань і ефективних імплементацій у різних алгоритмах стиснення без втрат.

2. Методи пошуку схожих послідовностей даних

У публікації [7] розглядаються два основні підходи до пошуку схожих послідовностей – онлайн і офлайн підхід. Онлайн підхід полягає в ітеративній побудові словника по ходу опрацювання даних. Кодування нових послідовностей відбувається паралельно з побудуванням словника.

Офлайн підхід, на відміну від онлайн підходу, обробляє усі дані заздалегідь і будує так звані індекси схожості між послідовностями тексту. Після того, як індекси побудовані, відбувається кодування усього тексту з фіксованим словником. Онлайн підхід працює швидше і використовує менше пам'яті, але поступається офлайн підходу за ефективністю стиснення.

У роботі «A Comparison of Approximate String Matching Algorithms» [8] були розглянуті неасимптотичні оптимізації, які пришвидшують тривіальний підхід. Асимптотична складність тривіального алгоритму, який обчислює відстань редагування з усіма послідовностями зі словника дорівнює $O(m*n)$, де m – загальна довжина усіх слів словника, а n – довжина слова, яке шукають. Представлені у цій публікації алгоритми асимптотично теж працюють за час, залежний від розміру словника, але на практиці показують значне прискорення. Основна ідея таких алгоритмів полягає у тому, що можна робити пошук у 2 етапи. На першому етапі проводиться швидка статистична перевірка кандидатів, а повна відстань редагування рахується лише на тих кандидатах, які пройшли перший етап. Завдяки тому, що більшість кандидатів значно відрізняються від патерну, повільна перевірка буде обчислена відносно невелику кількість раз. Основний же об'єм словника буде опрацьовано лише на етапі швидкої перевірки. Автори роботи порівняли ефективність основних евристичних алгоритмів на словнику розміром 100000 англійських слів. Результат зображений на рис. 2.

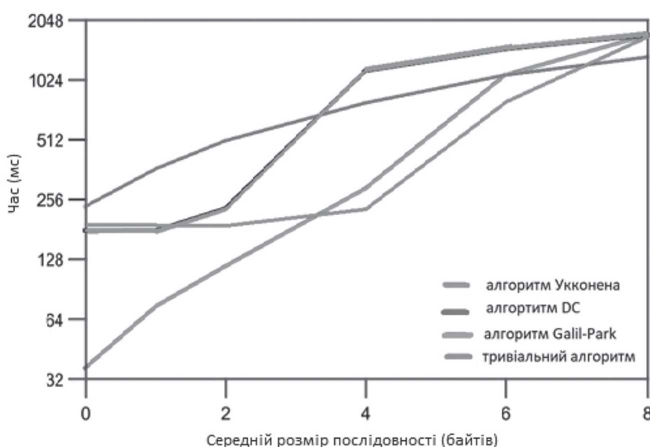

Рис. 2. Ефективність евристичних оптимізацій тривіального пошуку

Алгоритм «Galil-Park» [9] базується на монотонності таблиці пошуку і використовую так звані «посилання переходів», щоб швидко пропускати оброблені раніше послідовності.

Представлений у роботі [10] алгоритм «DC» використовує частоту зустрічання символів у словах словника і відфільтровує слова, частотний розподіл яких сильно відрізняється від шуканого.

Еско Укконен у своїй роботі [11] відзначив, що частотний підхід має значний недолік – він ніяк не враховує відносний порядок слів і має однакове значення для усіх перестановок заданого рядка. Він запропонував разом із частотним розподілом враховувати спільні послідовності, що забезпечило збіг не тільки множини символів, а і їх порядку.

За результатами тестування, проведеного авторами роботи [8] найбільш ефективним серед евристичних алгоритмів виявився метод Укконена, описаний у роботі [11].

Альтернативний підхід до пошуку схожих послідовностей у наборі даних був запропонований у публікації «Approximate string matching using n-grams technique» [12]. Автори роботи погоджуються з тим, що обчислення відстані редагування між поточною послідовністю та усіма іншими послідовностями, наявними у словнику, працює дуже повільно та пропонують поетапний алгоритм, який базується на так званих n-грамах. N-грама – це неперервна підстрока з N послідовних символів, які є складовими частинами рядка. Алгоритм складається з декількох етапів, на кожному з яких залишаються лише ті слова, які містять необхідну кількість n-грам, співпадаючих з n-грамами оброблюваного рядка. Ключовою ідеєю алгоритму є те, що на кожному етапі простір потенційних кандидатів значно зменшується, тому що аби не бути відфільтрованим, кандидат повинен містити не тільки необхідну кількість n-грам поточної довжини, але і всі менші n-грами. Автори стверджують, що достатнього всього 4 етапи навіть для послідовностей великої довжини і експериментально показують, що порівняння відбувається за лінійний час, на відміну від квадратичного у методі обчислення відстані підходом динамічного програмування.

У роботі «Indexing Methods for Approximate Dictionary Searching: Comparative Analysis»[13] представлений детальний аналіз більш сучасних та ефективних методів пошуку схожих послідовностей, які базуються на індексних структурах даних. Завдяки ефективним структурам даних пошуку строк, до яких відносяться суфіксні / префіксні структури, дерева пошуку, строкові автомати, та інші, вирішується основна проблема розглянутих вище методів – необхідність аналізувати увесь словник для пошуку кандидатів. Пошук інформації у індексних структурах може бути виконаний за суб-лінійний час, тобто значно швидше, ніж переглядати усі входження у словник. Але для того, щоб мати можливість робити

запити до таких структур, їх потрібно спочатку побудувати і підтримувати. Деякі методи, зокрема робота [14], пропонують заздалегідь побудувати структури даних на усьому тексті, але це не підходить до задачі компресії, адже під час стиснення необхідно шукати схожі послідовності лише на опрацьовану префіксу даних, а не на усьому тексті. Тому для цієї задачі необхідні «онлайн» індексні структури, тобто ті, які можуть підтримувати два типи запитів:

– додати нову послідовність у структуру;

– знайти усі послідовності, відстань редагування яких до заданої є не більшою за k.

Індексні методи можна поділити на дві категорії – префіксні дерева та методи генерації околиць. Автори публікації [13] аналізують ефективність кожної категорії окремо та визначають алгоритми, які показують кращу ефективність серед усіх індексних методів.

До категорії префіксних дерев відносять алгоритми, які знаходять схожі послідовності шляхом рекурсивного обходу дерева переходів, розглядаючи заміни, вставки та видалення вздовж шляху пошуку. Кловстат та Мондштей у своїй роботі [15] показали алгоритм пошуку, який базується на префіксному дереві «String trie», та довели, що переходячи за посиланнями автомату Левенштейна, цей алгоритм коректно відновіть шлях усіх кандидатів на схожу послідовність. Міхов та Шульц запропонували альтернативну структуру – FB-trie [16]. Основна відмінність полягає у тому, що метод підтримує два дерева – одне побудоване на оригінальному словнику, а інше – на словнику зворотних послідовностей. Це дозволяє комбінувати пошук у двох деревах, порівнюючи одночасно префікси і суфікси. У роботі [17] була запропонована ще одна суфіксна структура – k-errata trie. Вона націлена на оптимізацію найбільш складних випадків, впроваджуючи так звані centroid-шляхи дерева.

Найбільш ефективною категорією індексних структур пошуку вважають методи генерації околиці. Замість того, щоб шукати схожі послідовності у словнику, вони генерують околицю шуканого слова – фактично емулюють усі можливі відмінності, які можуть бути допущені від поточного рядка. Множину таких відмінностей з відстанню редагування не більше за k називають повною k-околицею. Після генерації такої околиці, кожен її елемент може бути дуже ефективно перевірений на наявність у словнику базовими методами, наприклад, хешуванням, або суфіксним деревом. Тривіальний алгоритм генерації такої околиці був вперше представлений Укконеном у роботі [18]. Пізніше Мейєрс у своїй роботі [19] представив так звані конденсовані околиці. Вони представляють собою повні околиці, з яких видалені неоптимальні елементи. Фактично в конденсованій околиці залишають лише елементи, які є суфіксами або префіксами яких-небудь слів зі словника.

Досить широкі тестові дослідження, проведені у роботі [13] показують, що найбільш ефективним алгоритмом серед індексних методів є алгоритм генерації околиць [18], тому саме він і буде протестований у складі алгоритму стиснення даних.

3. Методи кодування відмінностей

Наступним етапом в алгоритмі стиснення даних без втрат є кодування відмінностей. Як відомо, основна мета алгоритмів стиснення – зменшити фінальний розмір стисненого файлу. Використання схожих послідовностей дозволяє не записувати нову послідовність повністю, а записати лише посилання на попереднє входження та саме відмінність. Маючи ці дані, декодер зможе застосувати вказані операції до попереднього рядка і коректно відновити нову послідовність. Тому, не менш важливим етапом є максимально стиснути інформацію стосовно відмінностей, а саме – знайти спосіб закодувати відмінність мінімальною кількістю байт.

Базові алгоритми стиснення даних, такі як LZ77 [1] використовують лише точні збіги, тому їх можна закодувати досить просто – представити у вигляді пари (відстань; довжина). Декодеру потрібно буде просто скопіювати позначену підстроку. Але нам необхідно розглянути більш загальний випадок схожих послідовностей, тому окрім відстані і довжини необхідно також закодувати відмінності. На жаль, на теперішній час ця область розглядалася дуже стисло. Досить тривіальний підхід був запропонований у роботі [7], який базується на використанні відстані Левенштейна. Альтернативний метод, що ґрунтується на кодування відстані Геммінга, запропонували автори роботи [20] для використання у сфери стиснення зображень.

У якості першого методу буде розглянуто кодування відстані Левенштейна. Відстань Левенштейна дорівнює мінімальній кількості операцій, які необхідно щоб отримати з рідка А рядок В. У нашій задачі важлива не тільки кількість, а й самі операції для того, щоб можна було відновити новий рядок. Формально, мається оригінальний рядок А та послідовність операцій додавання, видалення або заміни символів у деяких позиціях. Приклад відстані Левенштейна та операцій для її досяжності зображений на рис. 3.

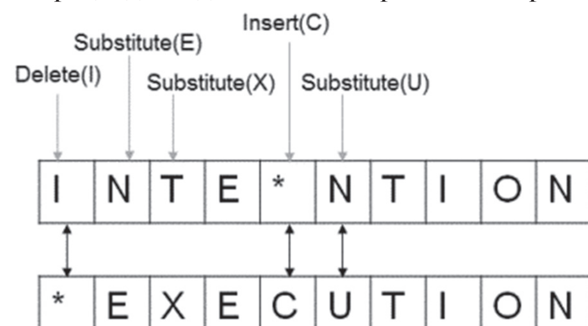

Рис. 3. Приклад перетворення рядка «INTENTION» у EXECUTION» методом Левенштейна

Автори роботи [7] пропонують розбити рядок на групи двох типів – група точного збігу та операція зміни. Для того, щоб групи чергувалися, між двома групами другого типу необхідно вставити пусті групи першого. Чергування груп необхідно для того, щоб забезпечити однакову кількість груп першого і другого типу. Після такого перетворення повне кодування схожого рядка можна зобразити наступною послідовністю: $(shift; length_1; length_2; \ldots; length_n; delimeter; operation_1; operation_2; \ldots; operation_n)$, де $shift$ – відстань до схожого рядка, $length_i$ – довжина групи повного збігу, $delimeter$ – спеціальний символ, який позначає закінчення опису груп першого типу, а $operation_i$ – операція зміни, яку необхідно зробити після копіювання групи під номером i. У свою чергу операція представляє собою два елементи – тип операції (заміна, додавання, або видалення символу) та новий символ для перших двох типів операцій.

Автори публікації зазначають, що такий підхід показує більш ефективну компресію на відміну від базового підходу. Основною перевагою цього підходу є те, що не потрібно кодувати відстань до кожної групи окремо, а достатньо лише один раз зазначити відстань до початку рядку, а потім просто записувати усі точні збіги один за одним, розділюючи їх операціями другого типу.

У роботі «Image Compression using Approximate Matching and Run Length»[20] пропонується альтернативний метод кодування відмінностей схожих послідовностей. Автори розглядають кодування відмінностей у контексті стиснення зображень, але їх метод можна розширити на компресію будь-яких даних. Основна ідея цього методу – використовувати відстань Геммінга замість відстані Левенштейна. Особливість відстані Геммінга полягає у тому, що вона є звуженням відстані Левенштейна і дозволяє тільки операції заміни символу. Більш того, у своєму оригінальному формулювання ця відстань розглядається лише у полі бінарних рядків, але вона може бути досить просто узагальнена на рядки будь-якого алфавіту. Також у цьому методі пропонується альтернативний спосіб кодування груп з точним збігом та замін. Замість кодування довжини групи, як це було зроблено у попередньому методі, алгоритм кодує одне бітове значення для кожного символу – «1», якщо символ співпадає, та «0» якщо він відрізняється. У разі, коли символ відрізняється, після нього кодується також новий символ, на який його потрібно замінити. Після цього послідовні повторення нулів та одиниць додатково стискаються алгоритмом RLE [21].

Автори публікації стверджують, що на практиці схожі послідовності досить малі, та операція заміни використовується частіше, тому їх алгоритм дає кращі результати ніж попередній. Нижче буде експериментально перевірено обидва запропонованих методи.

4. Метрики оцінювання якості алгоритмів

Для оцінки алгоритмів стиснення даних без втрат враховуються три основні показники:

ступінь стиснення – співвідношення між оригінальним розміром файлу та розміром файлу після стиснення;

швидкість компресії (МБ/секунду) – швидкість роботи алгоритму стиснення;

швидкість декомпресії (МБ/секунду) – швидкість роботи алгоритму відновлення оригінального файлу.

Розрахунок вказаних показників проводиться за наступними формулами:

$$compression\,ratio = \frac{original\ file\,size}{compressed\ file\,size},$$

$$compression\,speed = \frac{original\ file\,size}{compression\,time},$$

$$decompression\,speed = \frac{original\ file\,size}{decompression\,time}.$$

Найважливішим показником звісно є саме ступінь стиснення, тому що він характеризує, наскільки добре алгоритм стиснення здатен зменшити розмір файлу відносно його початкового розміру. В деяких роботах до оригінального розміру файлу також додається розмір програмного коду, який використовується для компресії і декомпресії. Але в нашому випадку тестування буде проводитись на великому об’ємі даних, тому розміром програмного коду можна знехтувати.

5. Огляд існуючих датасетів

У якості текстового датасету був обраний досить популярний датасет enwik8, який містить перші 10^8 символів англійської Вікіпедії. Аналіз датасету показав, що він містить 709,405 унікальних слів. На рис. 4 зображено розподіл 100 найчастіших слів у датасеті.

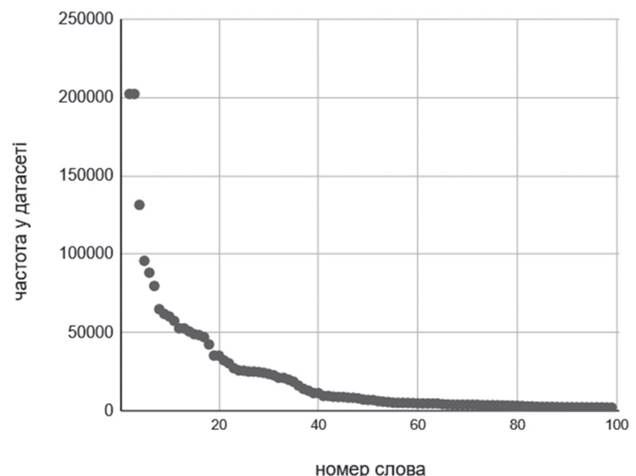

Рис. 4. Розподіл 100 самих частих слів у датасеті

З графіку можна переконатися, що текст містить велику кількість дуже частих слів, які багато повторюються, що посприяє ефективності роботи алгоритму.

В якості загального датасету був обраний розширений датасет Silesia Corpus, який складається з набору файлів популярних типів. Типи файлів у датасеті розділені на 6 категорій – виконувані файли, зображення, мультимедіа, документи, файли розмітки та файли баз даних. Кожна категорія містить декілька файлі різних форматів. Детальний аналіз вмісту датасету наведений у табл. 1.

Таблиця 1

Розподіл даних у датасеті Silesia Corpus

Тип файлів	Формати	Кількість файлів	Загальний розмір, МБ
Виконувані файли	EXE	5	16.8 МБ
Зображення	PNG, JPEG, BMP, TIFF	16	31.1 МБ
Мультимедіа	MP4, WEBM, MPEG, AVI	7	45.6 МБ
Документи	PDF, DOC, DOCX, PPTX	10	4.2 МБ
Файли розмітки	HTML, XML	6	0.7 МБ
Файли баз даних	DB, SQL	3	8.2 МБ

Загалом датасет enwik8 містить 1 файл розміром 100МБ, а датасет Silesia Corpus містить 47 файлів сумарним розміром 106.6 МБ.

6. Опис експериментальних досліджень

Виходячи з аналізу, проведеного у попередніх пунктах, найбільший інтерес представляють три алгоритми пошуку схожих послідовностей даних – алгоритм Укконена [11], що відноситься до евристичних алгоритмів, метод N-грам [12], та індексний метод генерації околиць [18]. Перш за все був реалізований алгоритм Укконена за схемою, описаною у оригінальній публікації. Алгоритм був реалізований на мові програмування C++ та вбудований у загальну реалізацію алгоритму стиснення даних LZ77 [1]. Як визначалося вище, цей алгоритм є евристичним тому його асимптотична складність складає $O\left(N^2 * k\right)$, де N – кількість слів у датасеті. Датасет enwik8 містить приблизно 10 мільйонів слів, тому навіть зі статистичними оптимізаціями реалізований алгоритм працює дуже довго. Для того, щоб отримати реалістичний час, алгоритм був допрацьований за схемою, описаною у [21]. Фактично, датасет був розділений на невеликі блоки і пошук схожої послідовності проводився тільки на останньому блоці. На рис. 5 зображена залежність ступеню стиснення даних від розміру блоку. У якості розмірів блоку були розглянуті ступені двійки, а саме 16, 64, 256, 1024, 4096, 16384, 65536, 262144 та 1048576 байтів. Для кращого сприйняття дані наведені на логарифмічній шкалі. З маленьким розміром блоку алгоритм працював дуже швидко, наприклад опрацювання всього словнику з блоком розміру 16 зайняло усього 5 секунд. Але через малий розмір блоку ступінь стиснення виявилася досить малою. Зі збільшенням розміру блоку датасет стискався краще, але алгоритм працював повільніше. На розміру блоку 1 МБ обробка датасету зайняла 28 хвилин.

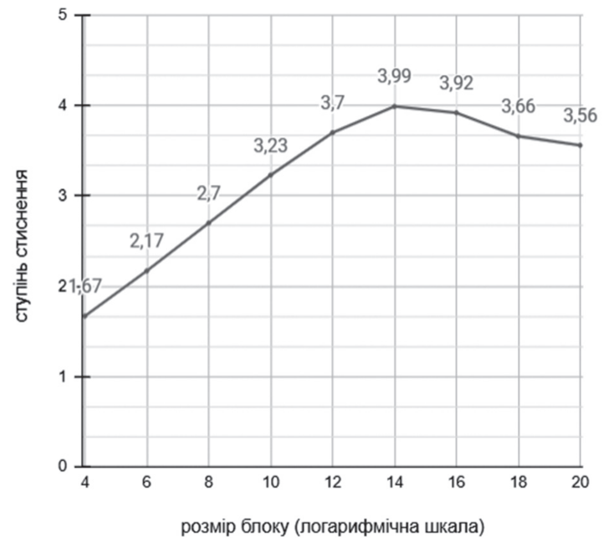

Рис. 5. Ступінь стиснення алгоритмом Укконена залежно від розміру блоку

Цікаво, що починаючи з розміру блоку 2^{16} якість стиснення почала спадати. Це пов'язано з тим, що коли блок дуже довгий, алгоритм знаходить схожі слова досить далеко позаду і кодує велику відстань, що є менш оптимальною, ніж закодувати декілька маленьких ближче чи не кодувати це слово взагалі. Оптимальним розміром блоку було визначено 16 КБ.

Наступним протестованим алгоритмом був метод N-грам [12]. Алгоритм складається з декількох етапів, на кожному з яких перевіряються лише той набір слів, який пройшов попередні перевірки. З цього набору слів обирається підмножина, яка містить достатню кількість N-грам поточної довжини. Автори стверджують, що після четвертого етапу множина потенційних кандидатів стиснеться до дуже малого розміру і усі його елементи можна перевірити стандартною метрикою. Алгоритм був реалізований на мові програмування C++ та протестований на датасеті enwik8. У табл. 2 наведено відповідність між номером етапу та відсотком кандидатів від загальної кількості слів у словнику.

Таблиця 2

Розмір множин кандидатів на етапах методу N-грам

Номер етапу	Відсоток потенційних кандидатів
1-грами	13.32%
2-грами	4.77%
3-грами	2.36%
4-грами	1.29%

Наведені результати були отримані шляхом тестування реалізованого методу N-грам у складі онлайн

схеми алгоритму, описаної у розділі 1. Відсоток був усереднений по всіх словах датасету. Як можна побачити з таблиці, множина кандидатів звужується дуже швидко і після чотирьох етапів залишається лише 1.3% кандидатів порівняно до оригінальної множини. Для пришвидшення алгоритму, так як і в попередньому випадку, датасет був поділений на блоки по 16КБ. Метод N-грам працює швидше евристичного методу, і обробка усього датасету зайняла 105 секунд.

Найбільш сучасним та ефективним методом пошуку схожих послідовностей є метод генерації околиць. У роботі [13] приводиться алгоритм онлайн підтримки словнику та пошуку послідовностей, редакційна відстань до яких не перевищує k. На основі цього алгоритму була протестована залежність розміру околиці від кількості зроблених змін у шуканому слові, результати чого зображені на рис. 6. Можна побачити, що допустимий розмір не повинен перевищувати 5, тому що для більших значені розмір околиці є занадто великим.

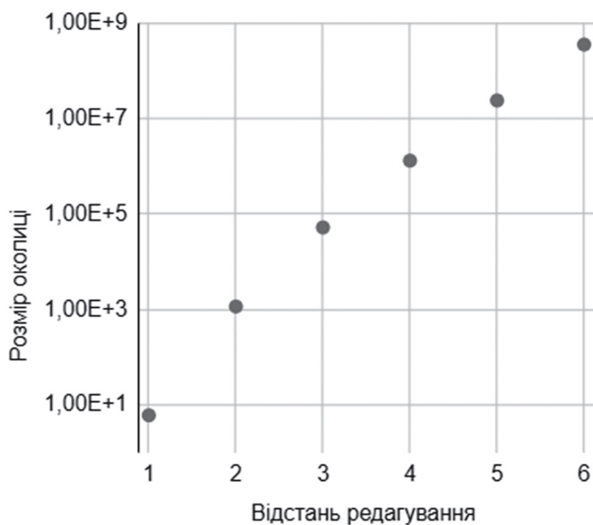

Рис. 6. залежність розміру околиці від відстані редагування

Для виконання перевірки префіксного чи суфіксного збігу буде використане структура даних префіксне дерево, побудоване на множині вже оброблених слів. Для того, щоб використовувати пошук суфіксів, необхідно побудувати два таких дерева – для прямих слів та перевернутих.

У фінальній частині дослідження було проведено тестування комбінацій розроблених методів у складі алгоритму стиснення даних. Для проведення тестування був реалізований застосунок на мові програмування Python. Також була використана утиліта lzbench для автоматизації тестування. Результати проведеного тестування зображені на рис. 7.

Всього було протестоване 6 комбінацій:

– евристичний метод Укконена з кодуванням методом Левенштейна;

– евристичний метод Укконена з кодуванням методом Геммінга;

– метод N-грам з кодуванням методом Левенштейна;

– метод N-грам з кодуванням методом Геммінга;

– індексний метод генерації околиць з кодуванням методом Левенштейна;

– індексний метод генерації околиць з кодуванням методом Геммінга.

На рис. 8 представлені результати тестування на комбінованому датасеті Silesia Corpus. Отримані результати експериментів показали, що найбільш ефективним методом пошуку схожих послідовностей на обох датасетах виявився евристичний метод Укконена, проте він є самим повільним. Цьому алгоритму вдається досягти такої високої ступені стиснення завдяки тому, що він тривіальна перебирає усіх можливих кандидатів. В той же час індексний метод генерації околиць працює майже в п’ять разів швидше, при тому не сильно поступаючись у ступені стиснення. Метод N-грам теж показав конкурентні

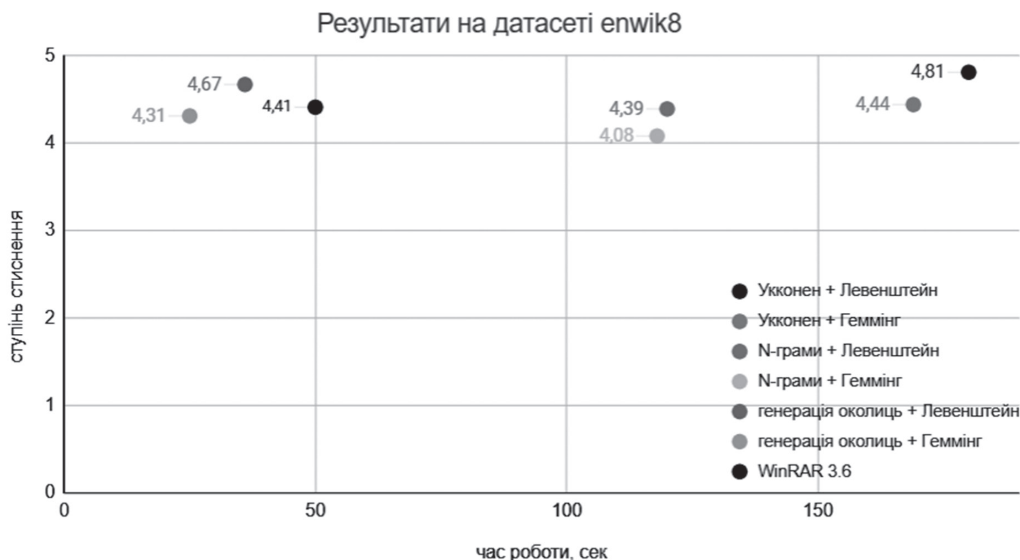

Рис. 7. Результати роботи алгоритмів на датасеті enwik8

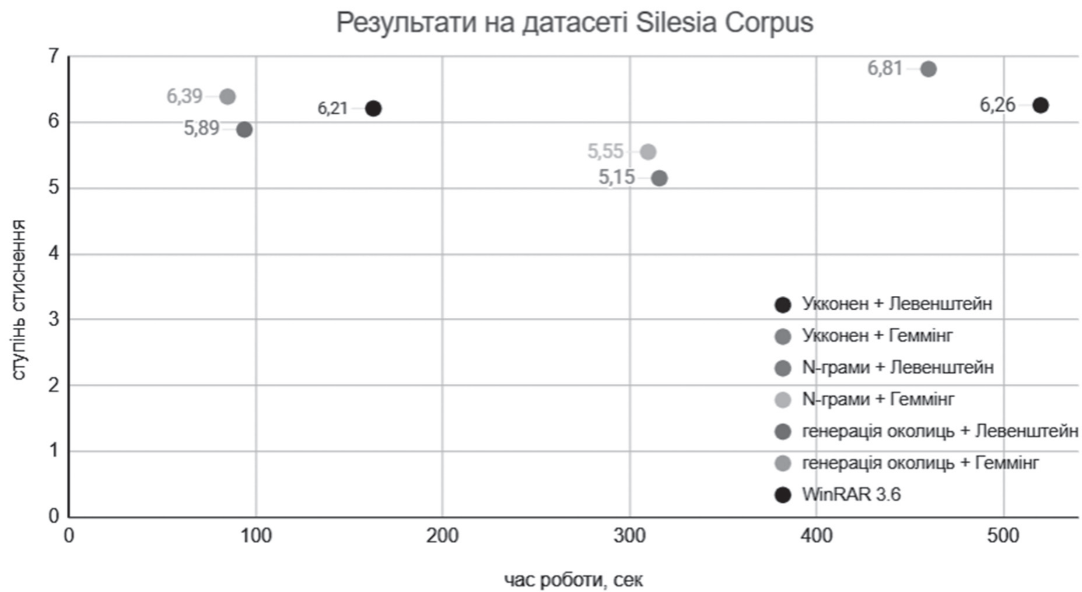

Рис. 8. Результати роботи алгоритмів на датасеті enwik8

результати, проте, поступився методу генерації околиць як по швидкості, так і по ступеню стиснення, що робить його не ефективним на практиці.

З іншого боку, методи кодування відмінностей показали себе неоднозначно. На датасеті enwik8 більш ефективним виявився метод Левенштейна, проте на датасеті Silesia Corpus – метод Геммінга. Це пов'язано з тим, що enwik8 складається зі статей природною мовою, де часто трапляються схожі послідовності з різними варіантами відмінностей – додаванням символу, видаленням, та заміною. Як раз це і забезпечив алгоритм Левенштейна. В той же час датасет Silesia Corpus містить більшість даних машинного формату – виконувані файли, зображення, мультимедіа, тощо. На таких даних частіше трапляються відмінності пов'язані лише с заміною символу, та зустрічаються багато дуже схожих послідовностей. Тому на таких даних метод Геммінга досяг більшого степеню стиснення. Також на обох датасетах підхід кодування методом Гемінга працював швидше, тому що він розглядає меншу кількість операцій і, відповідно, кандидатів.

Також для порівняння розроблених комбінацій з так званою базовою лінією, вони були порівняні з відомим компресором WinRAR версії 3.6. Як можна побачити з графіків, метод Укконена на обох датасетах випереджує WinRAR за ступенем стиснення, а метод генерації околиці – не тільки за ефективністю, а й за швидкістю.

Отже, розглядаючи алгоритм стиснення даних загалом, найкращим варіантом виявився індексний метод генерацій околиць у комбінації з кодуванням Геммінга. У спеціальному випадку роботи з природньою мовою слід використовувати той же метод, але з кодуванням Левенштейна.

Висновки

В ході проведення теоретичного аналізу була запропонована розширена версія алгоритму компресії, досліджені різноманітні підходи до пошуку схожих послідовностей, розглянуті евристичні методи, методі N-грам та індексні методи. Також були проаналізовані методи кодування відмінностей Левенштейна та Геммінга.

На етапі проведення експериментальних досліджень було обрано два репрезентативних датасети: enwik8 – датасет статей англійською мовою, та Silesia Corpus – комбінований датасет різних типів даних. На основі результатів аналізу було визачено три кращі методи пошуку схожих послідовностей – метод Укконена, метод N-грам та метод генерації околиць. Обрані методи були реалізовані та протестовані на визначеному датасеті в комбінації алгоритмами кодування відмінностей, базованих на відстані Левенштейна та Геммінга.

У результаті проведених теоретичних та експериментальних досліджень було обрано дві найбільш ефективні комбінації підходів до стиснення даних без втрат. На датасеті природньої мови кращий результат показав метод генерації околиць у комбінації з кодуванням Левенштейна, який досяг ступеню стиснення 4,67 за найкоротший час. Для комбінованого датасету найефективнішим також виявився метод генерації околиць, але з кодуванням Геммінга, та показав ступень стиснення 6,39.

Перспективним продовженням дослідження є аналіз більш широкого набору методів для пошуку схожих послідовностей (зокрема, нейронні мережі та підходи NLP). Крім того, доцільно проаналізувати комбінації методів на окремих датасетах для різних типів і форматах даних.

Список літератури:

[1] Jacob Ziv, Abraham Lempel. (1977) A Universal Algorithm for Sequential Data Compression IEEE Transactions on Information Theory, pp. 337—343.

[2] Van Leeuwen, Jan (1976) On the construction of Huffman trees, pp. 382–410.

[3] Katz, Phillip W. (1991) String searcher, and compresor using same string patterns.

[4] Ranganathan, N, Henriques, S. (1993) High-speed VLSI designs for Lempel-Ziv-based data compression. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Vol-40, No.2, pp. 96–106.

[5] E.Jebamalar Leavline ,D.Asir Antony Gnana Singh (2013) Hardware Implementation of LZMA Data Compression Algorithm. International Journal of Applied Information Systems (IJAIS).

[6] Apoorv Gupta, Aman Bansal, Vidhi Khanduja, (2017) Modern Lossless Compression Techniques: Review, Comparison and Analysis.

[7] Ilan Sadeh (2009) Universal data compression based on approximate string matching.

[8] Petteri J, Jorma T, Ukkonen E. (1998) A Comparison of Approximate String Matching Algorithms.

[9] Z. Galil, K. Park (1990) An improved algorithm for approximate string matching, SIAM Journal on Computing, 19, pp. 989–999.

[10] R. Grossi, F. Luccio (2018) Simple and efficient string matching with k mismatches, Information Processing, pp. 113–120.

[11] E. Ukkonen (1992) Approximate string matching with q-grams and maximal matches, Theoretical Computer Science, 92, pp. 191–211.

[12] Narendra Kumar, Vimal Bibhu, Mohammad Islam, Shashank Bhardwaj (2017) Approximate string matching using n-grams technique.

[13] Boytsov L. (2011) Indexing Methods for Approximate Dictionary Searching: Comparative Analysis.

[14] A. Yerokhin, A. Nechyporenko, O. Turuta, A. Babii (2016) A new intelligence-based approach for rhinomanometric data processing 2016 IEEE 36th International Conference on Electronics and Nanotechnology (ELNANO), Kiev, 2016, pp. 198-201.

[15] Klovstad J., Mondshein L. (1975) The CASPERS linguistic analysis system. IEEE Transactions on Acoustics, Speech and Signal Processing 23, pp. 118 – 123.

[16] Mihov S., Schulz K. U. (2004) Fast approximate string search in large dictionaries. Computational Linguistics, 30, 4, pp. 451–477.

[17] Cole R., Gottlieb, L. A., AND Lewenstein, M. (2004) Dictionary matching and indexing with errors and don’t cares. In STOC ’04: Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing. ACM, pp. 91–100.

[18] Ukkonen, E. (1985) Algorithms for approximate string matching. Information and Control 64, 1-3, pp. 100–118.

[19] Myers, E. (1994) A sublinear algorithm for approximate keyword searching. Algorithmica 12, 4/5, pp. 345–374.

[20] Samir Kumar Bandyopadhyay, Tuhin Utsab Paul, Avishek Raychoudhury (2016) Image Compression using Approximate Matching and Run Length.

[21] Robinson, A. H. Cherry, C. (1967) Results of a prototype television bandwidth compression scheme, Proceedings of the IEEE, pp. 356–364.

Надійшла до редколегії 26.03.2021