УДК 004.9

А.Ю. Гриненко, А.В. Петрашенко, Д.С. Замятін
НТУУ “КПІ”

МОДИФІКАЦІЯ АЛГОРИТМУ ПОШУКУ НЕЧІТКИХ ДУБЛІКАТІВ У ТЕКСТАХ УКРАЇНСЬКОЮ МОВОЮ

Розглянуті найпопулярніші алгоритми пошуку нечітких дублікатів. Було запропоновано модифікацію алгоритму пошуку нечітких дублікатів для текстових ресурсів, що ґрунтується на методі шинглів. Розроблений метод вирішує проблему чутливості до перестановки слів у класичному алгоритмі шинглів, що існує для текстів українською та іншими мовами слов'янської групи. Результати експериментів, що наведені роботі, показали приріст швидкості пошуку нечітких дублікатів.

Ключові слова: пошук нечітких дублікатів, метод шинглів.

In this paper, are considered most popular algorithms of near-duplicates detecting. Was proposed modified algorithm for detecting near-duplicates for text resources based on shingles. New method solves problem of sensitivity to words inversion for classical shingles method that takes place for Ukrainian and other Slavonic languages. The results of experiments shown in the work demonstrate that the speed of the duplicate detection algorithms is greatly increased.

Key words: near-duplicates detecting, shingles method.

Вступ

Стрімкий розвиток Internet, даючи змогу користувачам безперешкодно розміщувати інформацію, породив проблему появи у глобальній мережі велику кількість публікацій, що дублюють одна одну. Інформаційно-пошукові системи при формуванні видачі результату за певним запитом, враховують “схожість” документів з метою непотрапляння в результат видачі дубльованого контенту. Наявність дублікатів засмічує індекс пошукових систем, тим самим знижуючи швидкість обробки запиту і видачу результату.

Однією з основних перешкод для розв’язання задачі пошуку дублікатів є великий обсяг даних, що зберігаються в базах пошукових машин. За статистикою 28 % [1] інформації, що зберігається в Internet, становлять дублікати. Така кількість даних робить неможливим в розумний час розв’язання цієї задачі шляхом попарного порівняння текстів документів. Тому останнім часом увага науковців приділяється розробці методів зниження обчислювальної складності створених алгоритмів за рахунок застосування спеціалізованих алгоритмічних підходів (зокрема, хешування певного фіксованого набору “значущих” слів або пропозицій документа, семплювання набору підрядків тексту, використання дактилограм тощо) [2].

Сьогодні існує велика кількість методів і алгоритмів, які ґрунтуються на синтаксичних та лексичних принципах побудови документів. Алгоритми пошуку нечітких дублікатів широко використовуються в оптимізації роботи інформаційно-пошукових систем і автоматичної класифікації і кластеризації документів. Але їх практичне застосування виявило низку недоліків. Ці недоліки передусім пов'язані з високими вимогами до апаратних ресурсів, низькою ефективністю при обробці коротких документів, а також великою чутливістю до мінімальних змін у тексті документа.

Тобто задача підвищення ефективності методів пошуку нечітких дублікатів серед великої кількості електронних документів є доволі актуальною.

Постановка задачі

Проблему пошуку нечітких дублікатів розглядали ще на початку 90-х років. До цього часу вже було запропоновано доволі багато різних підходів та алгоритмів, які можна умовно розділити на дві групи: лексичні та синтаксичні.

Одними з перших досліджень у галузі знаходження нечітких дублікатів є роботи U. Manber [3] і N. Heintze [4]. У цих роботах для побудови вибірки використовуються підпослідовності сусідніх букв. Контрольна сума файла або документа включає всі текстові підрядки фіксованої довжини. Числове значення контрольної суми обчислюється за допомогою алгоритму випадкових поліномів Карпа-Рабіна. В якості міри схожості двох документів використовується відношення числа однакових підрядків до розміру файла або документа. Також в цих роботах пропонують методи, спрямовані на зниження обчислювальної складності алгоритму. U. Manber використовував цей підхід для знаходження схожих файлів (утиліта sif), а N. Heintze — для виявлення нечітких дублікатів документів (система Koala).

У 1997 р. A. Broder [4] запропонували новий, синтаксичний метод оцінки схожості документів, заснований на поданні документа у вигляді множини всіх можливих послідовностей фіксованої довжини k, які складаються з сусідніх слів. Такі послідовності були названі “шинглами”. Два документи вважалися схожими, якщо їх множини шинглів істотно перетиналися. Оскільки кількість шинглів приблизно дорівнює кількості слів у документі, тобто є достатньо великою, пізніше автори запропонували два методи зменшення розміру образу для отримання репрезентативних підмножин.

Інший сигнатурний підхід, заснований вже не на синтаксичних, а на лексичних принципах, запропонував A. Chowdhury [6] в 2002 і удосконалив у 2004 р. Основна ідея такого підходу полягає в обчисленні дактилограми I-Match для подання змісту документів. Два документи вважають схожими, якщо у них збігаються I-Match сигнатури. Алгоритм має високу обчислювальну ефективність, яка перевершує показники алгоритму A. Broder. Іншою перевагою алгоритму (також порівняно з A. Broder) є його висока ефективність при порівняно невеликих за розміром документів. Основний недолік — відсутність стійкості до невеликих змін тексту документа.

Схожий підхід описаний в патенті US Patent 6,658,423 W. Pugh з Google [7]. Автор пропонує повний словник документа розбити на фіксовану кількість списків слів за допомогою будь-якої функції хешування (наприклад, за допомогою залишку від ділення хеш-коду на кількість списків). Потім для кожного списку обчислюється дактилограма і два документи вважають схожими, якщо вони мають хоча б одну загальну дактилограму. Автор наводить оцінки стійкості алгоритму до невеликих змін змісту вихідного документа. Але, на жаль, у роботі відсутні будь-які відомості про ефективність практичного застосування з міркувань конфіденційності.

Ще одним сигнатурним підходом, також основаним на принципах лексичних (тобто використанні словника), є метод “опорних” слів, запропонований С. Ільїнським [8] та ін. Суть алгоритму полягає в наступному. Спочатку з індексу по деякому правилу вибирається множина з N слів, що називають “опорними”. Потім кожен документ представляється N-мірним двійковим вектором, де *i*-та координата дорівнює 1, якщо і-те “опорне” слово має в документі відносну частоту вище від певного порогу (встановлюваного окремо для кожного “опорного” слова), і дорівнює 0 у іншому випадку. Цей двійковий вектор називається сигнатурою документа. Два документи розпізнаються схожими, якщо у них збігаються сигнатури.

Оскільки розглянуті лексичні методи ґрунтуються на побудові словника за цілим текстом документа, вони не дозволяють локалізувати фрагменти, які продубльовано, що знижує якість оцінки чутливості методу. Тому в подальшій роботі було використано алгоритм шинглів, як найперспективніший. Одним з істотних недоліків алгоритму шинглів є його чутливість до перестановки слів. Під чутливістю мається на увазі ситуація, коли зміна порядку слів призводить до значного погіршення ефективності роботи алгоритму. Зокрема зниження показників точності та повноти пошуку.

Одним з істотних недоліків алгоритму шинглів є його чутливість до перестановки слів. Під чутливістю мається на увазі ситуація, коли зміна порядку слів призводить до значного погіршення ефективності роботи алгоритму, зокрема зниження показників точності та повноти пошуку.

Зумовлено це тим, що шингли виділяються з тексту так, щоб вони накладалися один на інший. Це робиться для запобігання втрати інформації і є важливою особливістю цього алгоритму. Здебільшого значення відстані між шинглами d дорівнює одному символу (якщо кроком обрано слово). Саме тому заміна однієї літери в тексті документа одразу призводить до зміни значень одразу декількох шинглів. Тому порівнюючи контрольні суми двох документів, ми вже не отримаємо коректного результату.

При вирішенні проблеми чутливості до перестановки слів потрібно врахувати той факт, що результати вдосконалень алгоритму значною мірою залежать від специфічних особливостей мови, якою створено документ.

Якщо потрібно скласти речення українською мовою, то слова у ньому можна поставити у будь-якій послідовності і від цього зміст значно не зміниться. Така відмінність з'являється, враховуючи те, що англійський синтаксис є набагато жорсткішим і передбачає стійкішим порядком слів, ніж речення в українській мові. Хоча необхідно мати на увазі, що свобода розташування членів речення в українській мові є відносною. Порядок слів завжди підкоряється певним нормам і завжди виконує ті чи інші граматичні, смислові чи стилістичні функції. Частково проблему чутливості до перестановки слів було розглянуто в статті А.Ф. Кузнєцова [9]. Так, автор визначав хеш-суму шинглу як суму числових ідентифікаторів кожного слова, що входили до цього шинглу. При цьому значення ідентифікатора окремого слова вибиралося зі словника базових форм слів. Якщо ж слово було відсутнє в словнику, то в якості ідентифікатора бралося значення числових кодів символів, піднесене в четвертий степінь.

Такий підхід дозволяв покращити результати, але він мав один значний недолік, а саме, через використання функції додавання, під час роботи з середніми та великими документами, виникало доволі багато колізій навіть у межах одного й того ж документа. Це означає, що для шинглів, які складаються навіть з абсолютно різних слів, обчислюються однакові контрольні суми. Це призводило до спотворення результатів роботи алгоритму.

Постановка задачі. Отже, в роботі розв'язується задача зменшення чутливості алгоритму шинглів для текстів українською та іншими мовами слов'янської групи.

Модифікація алгоритму шинглів

Основна ідея алгоритму шинглів полягає в розбитті тексту на послідовності слів однакової довжини – шингли (від англ. shingle – черепиця). Причому перед цим початковий документ приводиться до канонічного вигляду. Тобто з нього видаляються усі знаки пунктуації, стоп слова, HTML розмітка. Важливою особливістю цього алгоритму є те, що шингли виділяються не один за одним, а накладаються для запобігання втрати інформації.

Щоб зробити висновок про те чи насправді два документи є нечіткими дублікатами, або один з документів включає в себе інший, використовуються математичні поняття схожості (resemblance) та входження (containment).

Схожість двох документів A та B – це число між 0 та 1, таке, що у разі схожості, близької до одиниці, два документи можна вважати нечіткими дублікатами. Так само входження документа A в B є число між 0 та 1, таке що при його наближенні до одиниці, можна зробити висновок, що документ B включає документ A. Для обчислення схожості та/або входження двох документів достатньо зберігати для кожного документа його короткий образ у декілька сотень байт. Такий короткий образ може бути ефективно побудований за час, прямо пропорційний довжині документа. Отже, схожість та входження для двох документів може бути обчислена за час, прямо пропорційний до розміру образу.

Розглянемо текстовий документ як послідовність слів приведених до канонічної форми (токенів). Під канонічною формою мається на увазі видалення елементів форматування, HTML-тегів та знаків пунктуації. Тоді з кожним документом D можна пов'язати підмножину таких токенів S(D,w). Суміжні підпослідовності слів w, що входять в D, називаються шинглами. Отже, множиною шинглів для певного документа є набір всіх унікальних шинглів, що входять до цього документа.

Для заданої довжини шинглу, схожість r двох документів A та B можна визначити так:

$$r(A,B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}, \quad (1)$$

Тобто, це є відношення числа однакових для двох документів шинглів до загальної їх кількості. Входження документа A в документ B визначається як відношення числа однакових шинглів до числа шинглів A:

$$c(A,B) = \frac{|S(A) \cap S(B)|}{|S(A)|}, \quad (2)$$

При розпізнаванні нечітких дублікатів також використовується поняття відстані схожості (resemblance distance) документів, що визначається як d(A, B) = 1 – r(A, B).

Реалізація класичного алгоритму шинглів передбачає кілька основних етапів:

1. Канонізація тексту

На початковому етапі вхідні тексти приводяться до канонічного вигляду. Тобто з них видаляються усі знаки пунктуації, елементи форматування, зайві пробіли, HTML теги та стоп-слова (сполучники, частки, займенники, вигуки тощо), а також всі слова приводяться до початкової форми.

2. Розбиття тексту на шингли

Цей етап передбачає розбиття канонізованого тексту на підрядки однакової довжини (шингли). Найчастіше в якості кроку обираються символи або слова, і значно рідше – речення. Вибір кроку розміром у речення підходить лише для доволі великих документів і передбачає деякі зміни на етапі канонізації текстів. Шингли повинні виділятися з тексту не один за одним, а накладатися. Мінімальна відстань (d), між двома сусідніми шинглами дорівнює 1 слово/символ і гарантує, що при розбитті тексту на шингли не буде жодних втрат інформації. При збільшенні відстані d зменшується розмір вибірки, що позитивно впливає на швидкодію алгоритму.

3. Знаходження контрольних сум шинглів

Після того як ми маємо множину шинглів для текстового документа, необхідно обчислити їхні контрольні суми. Для цього використовують хеш-функції, які забезпечують мінімальну кількість колізій. Найвживанішими є криптографічні хеш-функції md5, crc, sha.

4. Побудова короткого образу документа

У класичному алгоритмі шинглів процес побудови образу передбачає відбір лише тих контрольних сум, які будуть використовуватися під час порівняння двох документів. Це можна зробити декількома способами.

Можна відбирати задану кількість мінімальних за значенням контрольних сум. У цьому випадку отримаємо фіксовану за розміром вибірку (що є перевагою для коротких документів), але не можна буде судити про входження документів.

Інший спосіб полягає в відборі контрольних сум, які діляться без залишку на деяке число з діапазону від 10 до 50. Такий метод називають відбором за модулем m (де m N). Розмір вибірки в такому випадку буде пропорційний до розміру документа.

Найпростішим є відбір абсолютно всіх шинглів. Тоді для зберігання образу потрібно виділяти більше пам’яті, а порівняння двох документів займатиме більше часу.

5. Пошук однакових підпослідовностей

На останньому етапі відбувається порівняння двох образів шляхом визначення ступеня їх схожості і входження. Якщо значення r (c) перевищує заданий поріг, то документи можна вважати нечіткими дублікатами. Значення порогу схожості визначається експериментально і здебільшого лежить у діапазоні 0,75–0,95.

Проаналізувавши цей алгоритм, було виявлено можливість модифікувати його з метою зменшення чутливості до перестановки слів на етапі розбиття тексту на шингли. Припустимо, що на цьому етапі в якості кроку було обрано речення, а не слово. Тоді в якості переставлених місцями фрагментів можна розглядати речення. Хоча прикладів таких змін, що не порушують викладену думку в тексті, підібрати значно складніше.

Спочатку сформулюємо задачу так. Визначимо S1, як один з шинглів, сформованих для деякого текстового документа, а S2 – як шингл, отриманий з S1 шляхом зміни місцями двох довільних слів.

Тоді можна записати, що

S1 = {w1,…,wi,…,wj,…,wn}

S2 = {w1,…,wj,…,wi,…,wn},

де wi, wj – переставлені місцями слова.

Нехай SH1 і SH2 хеш-суми обчислені для шинглів S1 та S2 за допомогою деякої хеш-функції hf. При порівнянні двох текстів, необхідно, щоб SH1 та SH2 набували однакових значень. Це означатиме, що перестановка слів не впливатиме на результати обчислення схожості r та входження c.

Тоді задача зменшення чутливості до зміни порядку слів зводиться до задоволення такого виразу:

SH1 = hf(S1) = hf(S2) = SH2

або

SH1 = hf(w1,…,wi,…,wj,…,wn) = hf(w1,…,wj,…,wi,…,wn) = SH2

Такого результату можна досягти на етапі розбиття канонізованого тексту на шингли, виконавши сортування за алфавітом усіх слів у межах кожного шинглу.

Це означає, що у разі будь-якої зміни порядку слів wi у межах шинглу, він завжди матиме сталий вигляд: S = { wa,wб,wв,…,wя }.

Отже, контрольні суми двох шинглів S1 та S2 (який отриманий з S1 методом перестановки слів) матимуть рівні значення, оскільки їх розраховують для однакових аргументів:

S1 = sort(w1,…,wi,…,wj,…,wn),

S2 = sort(w1,…,wj,…,wi,…,wn),

hf(S1) = hf(sort(w1,…,wi,…,wj,…,wn)),

hf(S2) = hf(sort(w1,…,wj,…,wi,…,wn)),

S1 = S2 = S => SH1 = hf(S1) = hf(S2) = SH2.

Важливою перевагою цього методу є відсутність будь-яких колізій, пов'язаних з операціями обчислення контрольних сум. Це означає, що хеш-суми двох шинглів S1 та S2 будуть рівними тільки в тому випадку, якщо вони складаються з однакового набору слів.

Також потрібно відмітити простоту програмної реалізації. Цей метод не передбачає кардинальних змін класичного алгоритму шинглів. Його можна розглядати як додаткове розширення базового конвеєра. До того ж реалізація функції сортування слів не викликає ніякої складності.

Як вже зазначалося, результати вдосконалень алгоритму шинглів багато в чому залежать від граматики конкретної мови. Чим довільнішим є порядок слів у реченні, тим краще проявить себе запропонований метод зменшення чутливості. Тому експерименти проводились окремо на вибірках документів українською та англійською мовами.

Приблизно 50 % усіх документів колекції становили нечіткі дублікати. Тобто практично для кожного оригіналу було створено відповідну копію документа. Кожен з цих дублікатів, був отриманий з оригіналу шляхом незначних модифікацій та обов'язкової зміни порядку слів у межах речення. Для того, щоб мати можливість оцінити зміну результатів алгоритмів залежно від мови, перестановка слів виконувалась лише за дотримання граматичної цілісності та збереження змісту речення. При цьому чутливість алгоритму визначалась як відношенням кількості правильно розпізнаних дублікатів до загальної кількості документів у вибірці:

$$Sens = \frac{N_{found}}{N_{existing}}, \tag{3}$$

де Nfound – кількість правильно розпізнаних дублікатів; Nexisting – загальна кількість наявних дублікатів, отриманих внаслідок зміни порядку слів.

Результати експерименту подані в вигляді діаграми, що зображена на рис. 1.

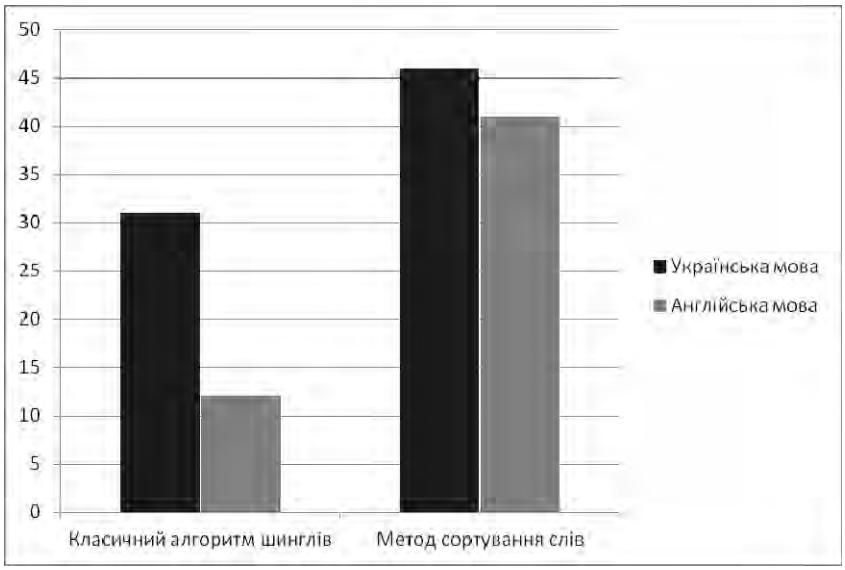

Рис. 6. Експериментальне дослідження зменшення чутливості алгоритму шинглів до перестановки слів у реченні

Проведений експеримент демонструє, що запропонований метод зменшення чутливості до перестановки слів показав значно кращі результати порівняно з класичним алгоритмом шинглів. У середньому кількість знайдених дублікатів зросла на 10–30 %.

Приріст кількості знайдених дублікатів для україномовної колекції документів становить близько 30 %, для англомовної колекції – лише 10–12 %. Це пояснюється тим, що в українській мові порядок слів у реченні є вільнішим, і при створенні експериментальної колекції дублікатів було зроблено значно більше змін у текстових документах. Значення міри чутливості досліджуваних алгоритмів наведені в табл. 1.

Таблиця 1

Показники міри чутливості до перестановки слів у реченні

	Українська мова	Англійська мова
Класичний алгоритм	0,24	0,62
Метод сортування слів	0,82	0,92

Як видно з табл. 1, значення міри чутливості для вдосконалених алгоритмів наближається до одиниці. Це значить, що майже всі дублікати були знайдені.

Реалізація на GPU

Під час реалізації було помічено, що деякі етапи алгоритму шинглів (видалення стоп-слів, генерація шинглів, порівняння образів) придатні до розпаралелювання. Був використаний підхід обчислень на GPU (graphics processing unit) за допомогою технології CUDA. CUDA (Compute Unified Device Architecture) – технологія, яка розроблена фірмою Nvidia[10] для паралельних обчислень, що дозволяє істотно збільшити обчислювальну продуктивність завдяки багатоядерній обчислювальній потужності графічних процесорів.

Тестування було проведено на декількох комп’ютерах з центральними процесорами від AMD Athlon(tm) до Intel(R) Core(TM) i7 CPU. Значний внесок у обчислювальну складність робить алгоритм хешування, як описувалось раніше – MD5 добре розпаралелюється, тому в результаті маємо більше прискорення. Хоч реалізація на CPU послідовна і виконується на одному ядрі, а відтак до уваги варто брати тактову частоту одного ядра. Проте, як бачимо (табл. 2) на 8-ядерному процесорі час найкращий серед представлених CPU. Це можна пояснити тим, що i7 має більш ефективну архітектуру:

певні інструкції виконуються за меншу кількість тактів
конвеєри з більшим числом рівнів
більший і швидший кеш
швидша пам'ять

Таблиця 2

Порівняльний аналіз

Пристрій		CRC32		MD5	
Назва	Частота, MHz	Час, мс	Прискорення	Час, мс	Прискорення
AMD Athlon(tm)	1102	14217,36	43,95	69073,60	150,12
AMD Athlon(tm) 64 Processor 3000+	1809	4852,79	15	11727,28	25,49
AMD Athlon(tm) 64 X2 Dual Core Processor 6000+	3015	3156,40	9,76	5206,54	11,32
Intel(R) Core(TM)2 CPU 4400	1994	2746,32	8,49	3395,50	7,38
Intel(R) Core(TM)2 Duo CPU T6600	2194	2522,01	7,80	3170,56	6,89
Intel(R) Core(TM) i7 CPU 920	2664	1855,82	5,74	2028,86	4,41
GeForce 9800 GT	1500	323,46	–	460,13	–

З наведеної таблиці бачимо, що швидкодії під час обчислення на GPU істотно (на два порядки) переважає швидкодію центрального процессора. Також оцінювали швидкодію алгоритму для різних розмірів вхідного документа. Експеримент проводився з відеокартою GeForce 9800 GT і центральним процесором AMD Athlon(tm) 64 X2 Dual Core Processor 6000+.

Розглядались розміри файлів від 0.5 до 4 Мб, на яких можна легко прослідкувати закономірність. Нижче наведені діаграми порівняння часу роботи програми на CPU і GPU, з використанням алгоритмів хешування CRC32 і MD5 окремо.

Таблиця 3

Час роботи програми на основі алгоритму CRC32

	500 Kb	1 Mb	2 Mb	3 Mb	4 Mb
CPU	880,90	3074,82	8584,07	16602,24	26553,79
GPU	200,80	306,67	827,67	1550,90	2560,68

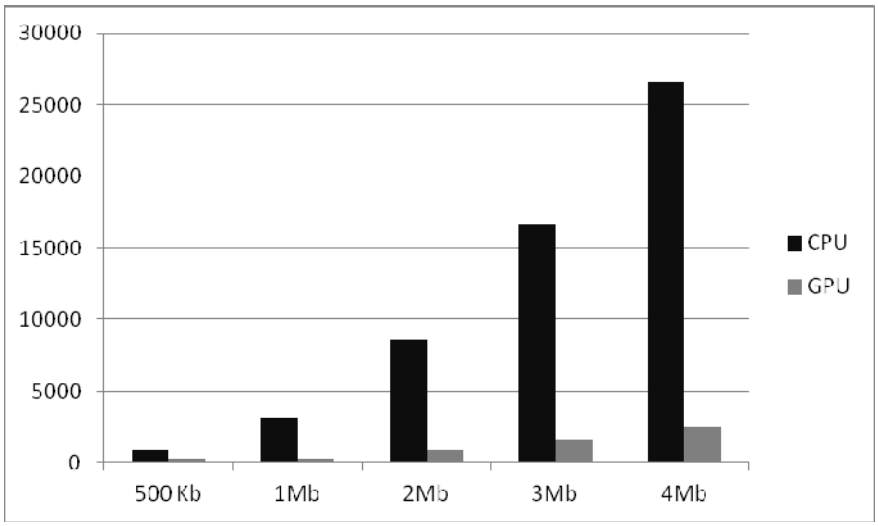

Рис. 7. Діаграма роботи програми на основі алгоритму CRC32

Таблиця 4

Час роботи програми на основі алгоритму MD5

	500 Kb	1 Mb	2 Mb	3 Mb	4 Mb
CPU	1227,92	4805,88	27016,30	48024,88	84341,52
GPU	230,21	463,29	1285,24	2473,88	4152,92

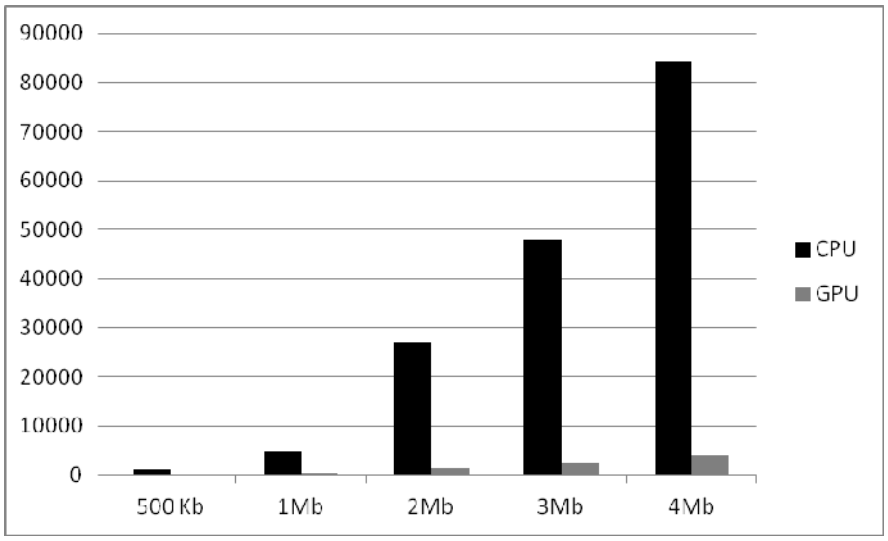

Рис. 8. Діаграма роботи програми на основі алгоритму MD5

Отже, використання технології розпаралелювання з застосуванням GPU дає змогу значно збільшити продуктивність розглянутого алгоритму.

Висновки

У роботі було запропоновано метод зменшення чутливості алгоритму шинглів до перестановки слів у межах речення та проведено його експериментальне дослідження.

Було показано, що результати запропонованого методу значною мірою залежать від граматики мови. Якщо під час роботи з документами англійською мовою спостерігається підвищення коефіцієнта чутливості на 30 %, то для україномовної вибірки документів коефіцієнт чутливості зростає на 60 %.

Для підвищення продуктивності було запропоновано розпаралелювання з використанням технології GPGPU та експериментально показано ефективність такого підходу.

1. Duplicate and Near Duplicate Documents Detection: A Review.[Електронний ресурс] / J. Prasanna Kumar, P. Govindarajulu // European Journal of Scientific Research, EuroJournals Publishing, pp. 514–527: http://www.eurojournals.com/ejsr_32_4_08.pdf 2. Сравнительный анализ методов определения нечетких дубликатов для Web-документов. [Електронний ресурс] / Зеленков Ю.Г., Сегалович И.В.: http://rcdl2007.pereslavl.ru/papers/paper_65_v1.pdf 3. Finding Similar Files in a Large File System / U. Manber // Winter USENIX Technical Conference, 1994. – pp. 1–10. 4. Scalable Document Fingerprinting.[Електронний ресурс] / Nevin Heintze // Proc USENIX Workshop on Electronic Commerce (1996): http://www.mendeley.com/research/scalable-document-fingerprinting 5. Syntactic clustering of the Web. [Електронний ресурс] / A. Broder, S. Glassman, M. Manasse and G. Zweig. // Proc. of the 6th International World Wide Web Conference, April 1997. – pp. 1–13: http://citeseer.ist.psu.edu/viewdoc/download;jsessionid=6E6FE4B04E54CD09C7F59CBCDDA530BE?doi=10.1.1.134.4842&rep=rep1&type=pdf 6. Improved Robustness of Signature-Based Near-Replica Detection via Lexicon Randomization / A. Kolcz, A. Chowdhury, J. Alspector // KDD 2004. – pp. 1–6.: http://citeseer.ist.psu.edu/viewdoc/download;jsessionid=6E6FE4B04E54CD09C7F59CBCDDA530BE?doi=10.1.1.134.4842&rep=rep1&type=pdf 7. Detecting duplicate and near-duplicate files.[Електронний ресурс] / W. Pugh // US Patent 6,658,423 pp. 1–25: http://www.cs.umd.edu/~pugh/google/Duplicates.pdf 8. An efficient method to detect duplicates of Web documents with the use of inverted index. / S. Ilyinsky, M. Kuzmin, A. Melkov, I. Segalovich. // WWW Conference 2002. – pp. 1–6. 9. Проблема дублирования страниц и поиска нечетких дубликатов в сайтах по экономической тематике. [Електронний ресурс] / Руслан Ф. Кузнецов: http://st.free-lance.ru/users/rusl_ir/upload/f_4a97d3d360d0e.doc 10. What is CUDA. [Електронний ресурс] // NVIDIA Developer Zone: http://developer.nvidia.com/what-cuda.