POLITECHNIKA POZNAŃSKA
WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI
oraz
IBM Software Lab Kraków

Metody oceny różnorodności danych dla kompresji typu delta

Pod opieką prof. Roberta Wrembla z Politechniki Poznańskiej
oraz Pana Michała Bodzionego z IBM

Dokumentacja techniczna

Autorzy:
Tomasz Chudziak nr. 136691
Marek Kamiński nr. 136732
Wojciech Szczepaniak nr. 136808

Poznań, 2021

1. Wstęp

Celem projektu jest zbadanie wpływu wyboru miar podobieństwa, na jakość kompresji typu delta. Istotne jest również czy taka kompresja w ogóle może być opłacalna. Eksperymenty były przeprowadzane na danych tekstowych. Do celów badawczych zostały wybrane następujące miary podobieństwa:

- tri-gram,
- cosine,
- Levenstein,
- Monge Elkan,
- Winkler-Jaro.

Rozpatrywana metoda polega na policzeniu zmian pomiędzy kolejnymi elementami i zapisaniu ich (oraz początkowego elementu), zamiast zapisywać oryginały. W koncepcji powinno to pozwolić nam zredukować rozmiar zbioru. Miary służą do określania sposobu ułożenia danych. W ramach badania chcemy przetestować czy ułożenie ma wpływ na jakość kompresji oraz sposób sortowania z użyciem której miary daje najlepsze wyniki.

2. Środowisko uruchomieniowe

Język:

- Python 3.8

Biblioteki:

- Pandas 1.2.3
- Numpy 1.20.1
- Jupyter Notebook 6.3.0
- Matplotlib 3.3.4
- Seaborn 0.11.1
- diff-match-patch 20200713
- fuzzy-match 0.1.0
- textdistance 4.2.1

3. Test bibliotek liczących delty

Cel:

Głównym celem było przetestowanie różnych metod z wybranych bibliotek, a także porównanie ich wydajności i efektywności.

Wstęp:

Testy zostały przeprowadzone na zbiorach Amazon-Google, które zawierają opisy produktów. Przetestowane zostały 2 biblioteki:

- diff_match_patch,
- xdelta3.

Biblioteka diff_match_patch została opracowana przez pracowników Google, w celu prowadzenia zapisów zmian w Google Docs. Interesowały nas 2 główne grupy metod: do obliczania patchy oraz do liczenia różnic między plikami. Patchami nazywamy serię małych zmian, które trzeba wykonać, żeby z dokumentu A otrzymać dokument B. W xdelcie interesowała nas tylko metoda zwracająca deltę.

Metodyka:

Początkowym etapem jest zapisanie danych wejściowych jako słowa początkowego i kolejnych zmian. Zmiany te umożliwiają odtworzenie oryginalnych wartości, poprzez nałożenie ich na poprzedni ciąg znaków. Następnym etapem jest wykonanie odtwarzania danych wejściowych na podstawie tych zmian. W trakcie wykonywania procedury analizie została poddana reprezentacja delty. Ostatecznie porównane zostały także otrzymane rozmiary zbiorów wynikowych z rozmiarem danych wejściowych.

Wnioski:

Użyta przez nas wersja Xdelta3 wymaga użycia środowiska Linuksowego. Ze względu na porównywalną efektywność oraz łatwiejsze użycie, biblioteka diff_match_patch została wybrana do dalszej analizy zagadnienia.

4. Zbadanie wydajności wybranego algorytmu oraz ocenienie jego efektywności w zależności od zastosowanego progu

Cel I:

Zbadanie wydajności algorytmu oraz określenie czy pracuje jednowątkowo.

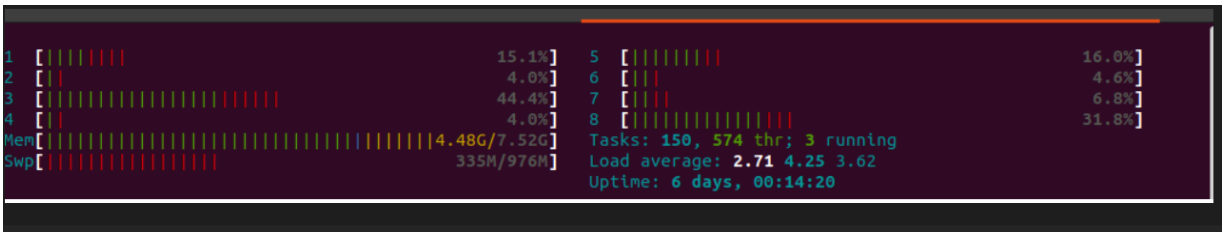
Wstęp:

Celem tego eksperymentu jest sprawdzenie sposobu działania biblioteki diff_match_patch. W tym przetestowanie jej wydajności oraz sposobu działania (jedno- czy wielowątkowe działanie). Poszczególne eksperymenty zostały przeprowadzone na zbiorze Scholar.csv, zawierającego dane książek. Wykorzystane z niego zostały tylko tytuły książek.

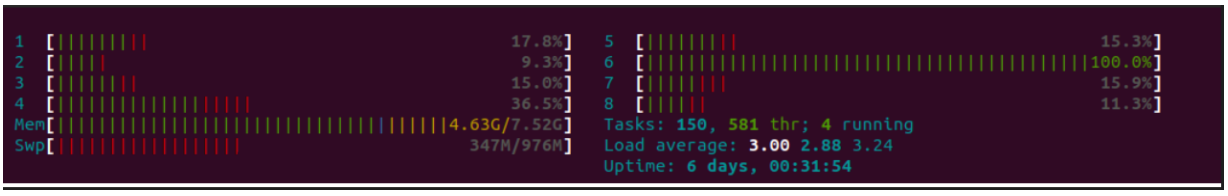
Metodyka:

Przygotowane funkcje zapisu oraz odczytu, były kolejno odpalane na jednej maszynie, z włączonymi monitorami zużycia zasobów. Osobno było testowane tworzenie patchy, tworzenie diffów, oraz metody odczytywania oryginalnych tekstów.

Wyniki:



rysunek 1: stan systemu w bezczynności IDLE



rysunek 2: stan w trakcie pracy

Z obserwacji wynika, że biblioteka pracuje z użyciem jednego wątku. Widać, że wzrasta do 100% obciążenie rdzenia szóstego i tylko tego.

Cel II:

Zbadanie czasu działania w zależności od wielkości instancji. Czy nakład czasu jest liniowy?

Metodyka:

Dla zbiorów o liczności <1000,10000>, przeprowadzane były testy z zastosowaniem wybranych metod (zapisu jako diff, jako patch oraz odczytanie z diff i patch oryginału), oraz zapisanie czasu wykonywania. Statystyki użytych zbiorów znajdują się w tabeli 1.

Metoda 1 polega na przygotowaniu samych patches. Patches to zbiór operacji potrzebnych do zamiany tekstu A w tekst B.

Metoda 2 tworzy najpierw zapis różnic (diff), następnie optymalizuje go pod kątem wydajności oraz przygotowuje z niego patche.

Metoda 3 działa podobnie jak metoda 2, tylko zamiast tworzyć patche, finalnie dostarcza nam zapis w postaci delty.

liczba rekordów	**średnia liczba znaków**	**min liczba znaków**	**max liczba znaków**
1000.0	60.781	3.0	105.0
2000.0	61.105	3.0	111.0
3000.0	61.002	3.0	111.0
4000.0	61.006	3.0	111.0
5000.0	60.981	3.0	111.0
6000.0	60.731	3.0	111.0
7000.0	60.794	3.0	111.0
8000.0	60.606	3.0	111.0
9000.0	60.540	3.0	111.0
10000.0	60.396	3.0	137.0

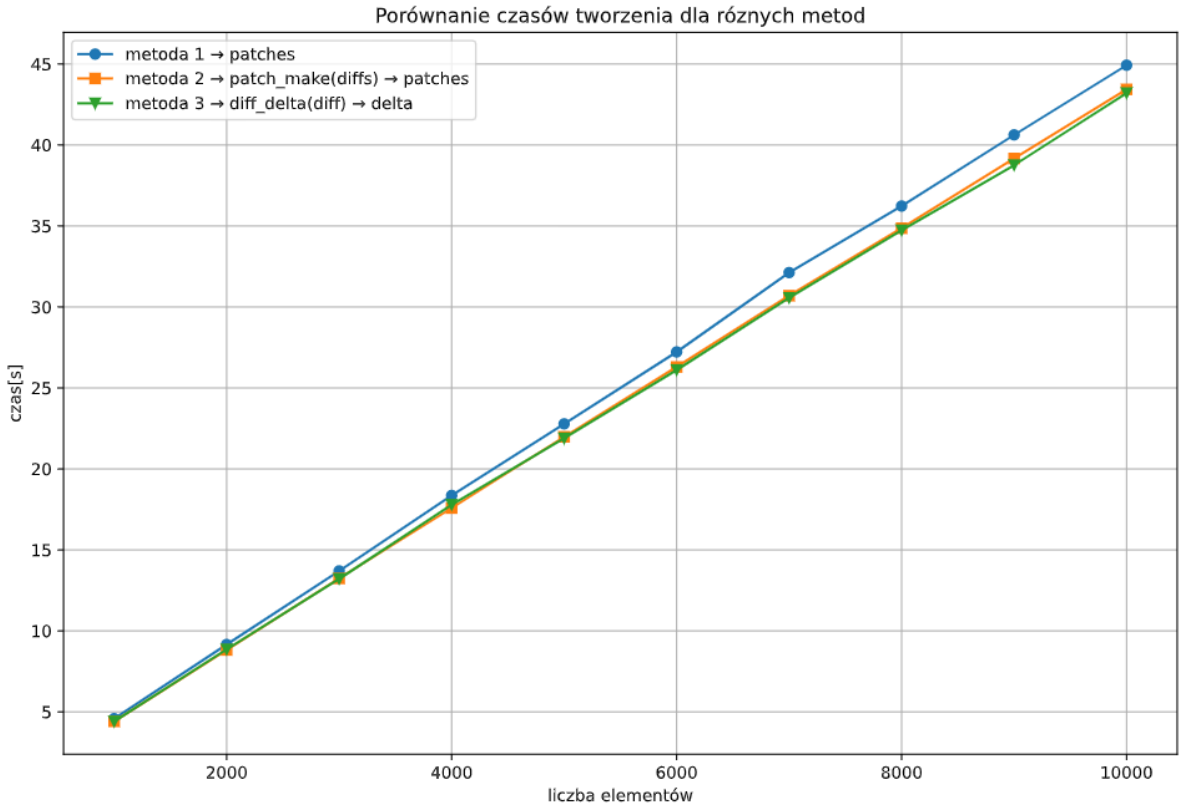
tabela 1: Statystyki użytych zbiorów

Wyniki:

Tabela 1 zawiera czasy kompresji dla opisanych metod. Wyniki zostały przedstawione na wykresie 1. Można zauważyć, że przyrost czasu wykonywania się, ma przebieg liniowy dla każdej z metod.

Liczba elementów	Metoda 1 → patches [s]	Metoda 2 →diffs→ patches [s]	Metoda 3 → delta [s]
1000	4.581198	4.413722	4.408278
2000	9.157612	8.820671	8.856514
3000	13.702620	13.238832	13.209607
4000	18.360261	17.582304	17.790571
5000	22.782317	21.982550	21.890714
6000	27.223517	26.293684	26.111189
7000	32.122719	30.694999	30.560659
8000	36.229824	34.862193	34.734727
9000	40.626112	39.167863	38.758537
10000	44.928458	43.434494	43.206983

tabela 2: Czasy kompresji

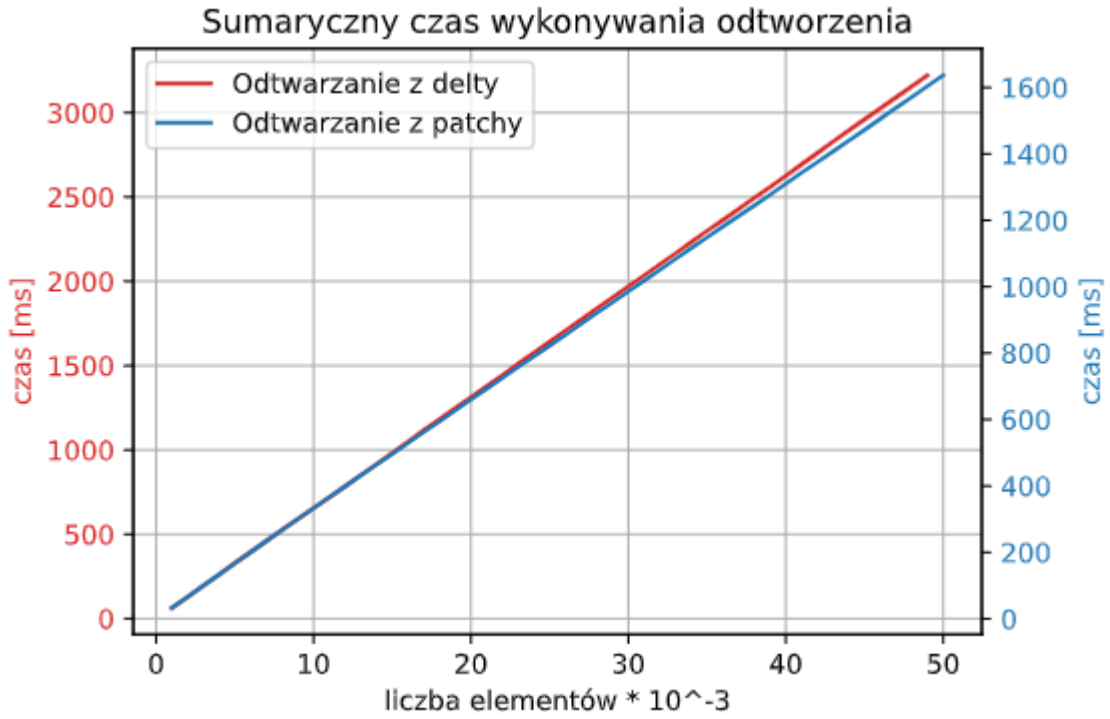


wykres 1: Wizualizacja czasów kompresji dla zbiorów o różnej liczebności, dane z tabeli 2

Tabela 2 zawiera czasy dekompresji oryginalnego zbioru, w zależności od użytego formatu zapisu: Delta lub patch. Podobnie jak dla obliczania różnic, czas odtwarzania również wzrasta liniowo. Można zauważyć, że z formatu danych patches, dane odtwarzają się prawie dwa razy szybciej względem czasu odtwarzania danych z formatu delta.

Liczba elementów	Czas odtwarzania z delty [ms]	Czas odtwarzania z patchy [ms]
5000	262.038469	166.988134
10000	591.025352	332.985640
15000	916.038513	495.990753
20000	1247.038364	659.988165
25000	1576.038599	821.963072
30000	1904.026747	985.963821
35000	2233.038425	1148.949862
40000	2560.024023	1309.947968
45000	2895.053387	1471.960545
50000	3222.082853	1636.951923

tabela 3: Czasy dekompresji



wykres 2: Wizualizacja czasów odtwarzania, dane z tabeli 3

Cel III:

Zbadanie, jaki próg dla współczynnika kompresji pojedynczej pary daje najlepsze wyniki dla zbioru. Drugim etapem jest zbadanie, czy można uzyskać lepszą kompresję poprzez zastosowanie kompresji delta do wszystkich rekordów. Jako odnośnik posłużą wyniki z poprzedniego etapu eksperymentu. Współczynnikiem kompresji nazywamy stosunek wielkości pliku wyjściowego do wejściowego. Im jest on mniejszy, tym lepiej.

$$WK = \frac{RozmiarPoSkompresowaniu}{RozmiarPrzedSkompresowaniem}$$

Metodyka:

Dany był stały rozmiar zbioru danych. Dla coraz większych progów badany był rozmiar wynikowy i jeśli współczynnik kompresji był większy niż zadany próg, wybierany był rozmiar oryginalnego rekordu. W celu symulowania specjalnego formatu danych, do wyniku końcowego (rozmiaru na dysku) w bajtach, dodawana była wielkość zbioru w bitach.

Wyniki:

Wykresy przedstawiają oryginalne rozmiary zbiorów oraz ich rozmiary dla różnych poziomów kompresji. Na osi y z prawej strony znajduje się wielkość zbioru na dysku (w bajtach). Niebieskimi kwadratami oznaczona jest liczba skompresowanych elementów. Mniejszy rozmiar = lepiej.

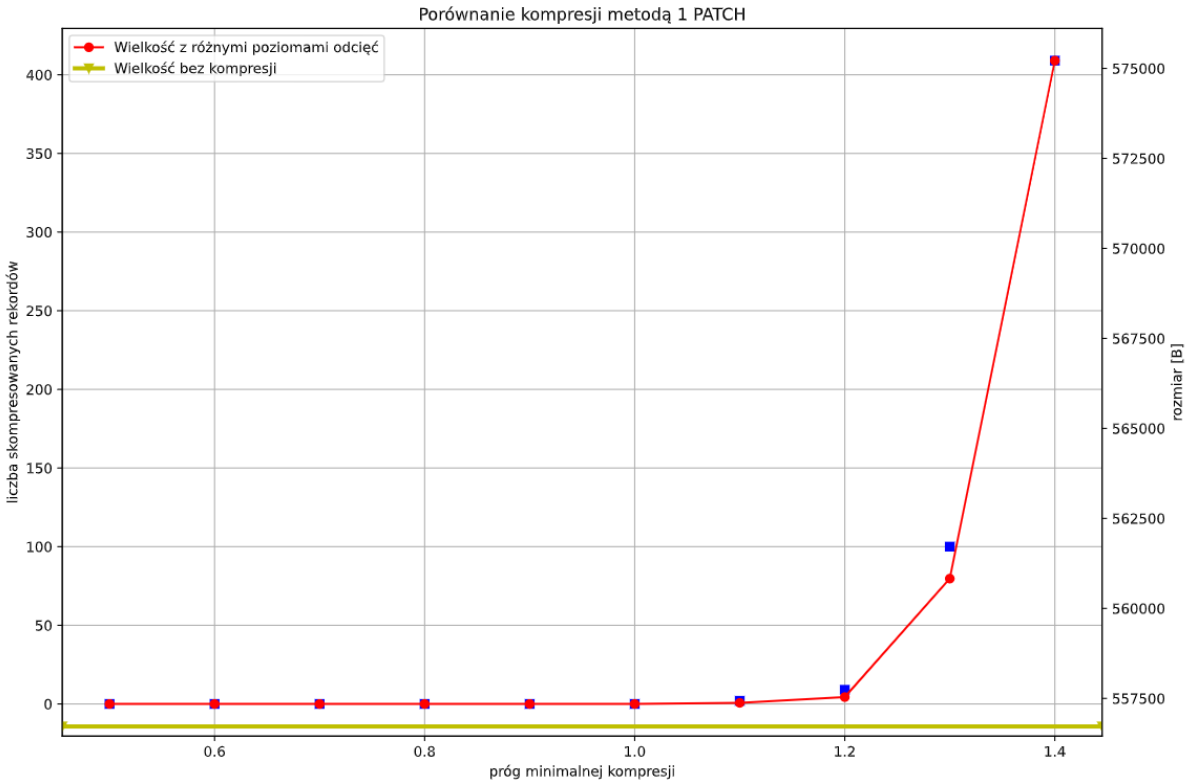
Dla metody patch nie udało się dokonać kompresji, niezależnie od przyjętej wartości progu minimalnej kompresji (wykres 3).

Natomiast przy metodzie delta (wykres 4), dla poziomów z przedziału <0.9;1>, nastąpiła nieznaczna poprawa i przy progu = 1, udało się dokonać kompresji 201 z 5000 rekordów, co zmniejszyło rozmiar zbioru o ok 0.2 - 0.3%.

Również przy próbie kompresji bez stosowania progu, nie udało się uzyskać poprawy wielkości zbioru.

Próg kompresji	Rozmiar [B]	Liczba skompresowanych rekordów
0.5	557346	0
0.6	557346	0
0.7	557346	0
0.8	557346	0
0.9	557346	0
1.0	557346	0
1.1	557376	2
1.2	557536	9
1.3	560827	100
1.4	575216	409
Rozmiar oryginału	556721	0

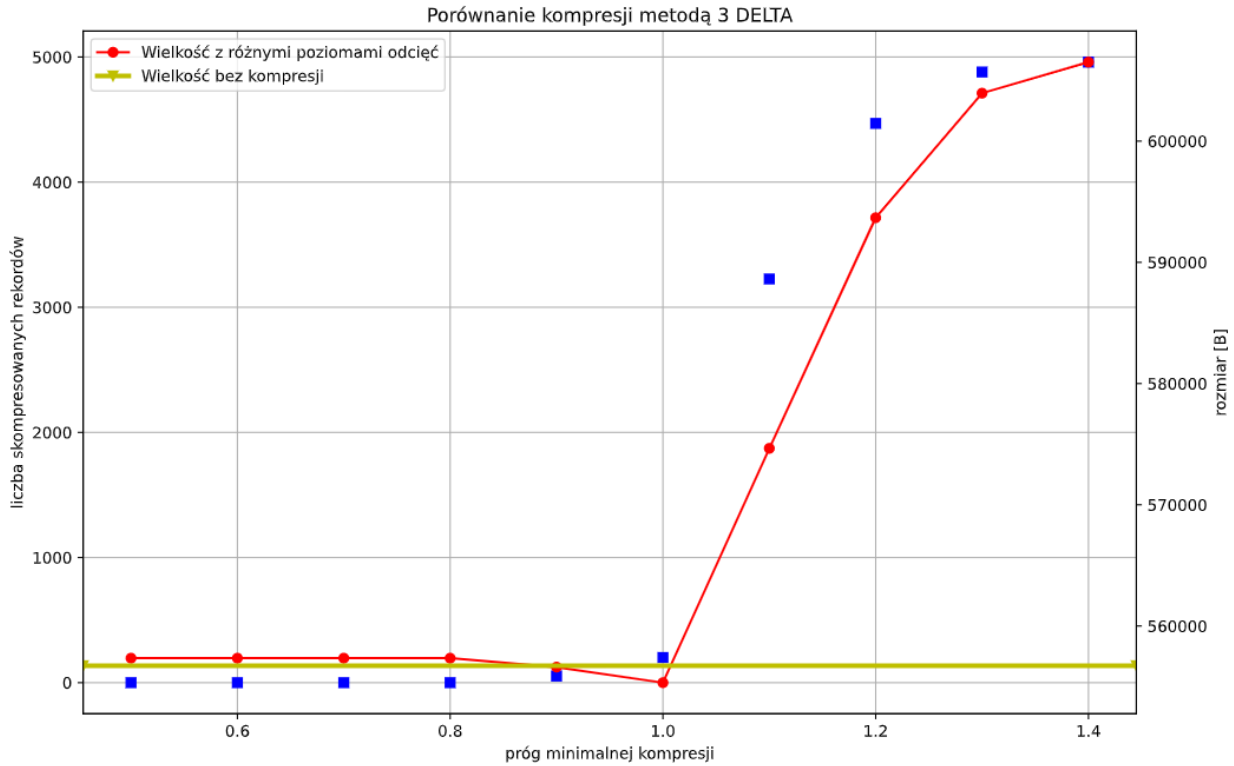
tabela 4: Dane dotyczące kompresji z użyciem patches



wykres 3: Rozmiar dla patchy, na podstawie danych z tabeli 4

Próg kompresji	Rozmiar [B]	Liczba skompresowanych rekordów
0.5	557346	0
0.6	557346	0
0.7	557346	0
0.8	557346	0
0.9	556609	53
1.0	555324	201
1.1	574656	3226
1.2	593682	4469
1.3	603955	4880
1.4	606520	4959
Rozmiar oryginału	556721	0

tabela 5: Dane dotyczące kompresji z użyciem delty



wykres 4: Rozmiar dla delt, dane z tabeli 5

Wnioski:

Czas działania biblioteki skaluje się liniowo wraz ze wzrostem rozmiaru danych. Oznacza to, że algorytm może być przydatny i skalowalny. Z kolei jednowątkowość daje nadzieje na możliwość poprawy działania poprzez zrównoleglenie obliczeń, np. podzielenie zbioru na kilka części i liczenie dla każdej z nich podobieństw równolegle, na końcu obliczając wartości dla danych na łączeniach.

Zastosowane dane były różnej długości i nie były do siebie zbyt podobne. Uzyskana dla nich kompresja jest całkowicie nieprzydatna, ze względu na słabą jakość. Konieczny nakład czasu i mocy obliczeniowej przewyższa osiągane korzyści.

5. Testowanie wpływu różnych metod sortowania na jakość kompresji

Kolejny eksperyment polegał na przetestowaniu różnic w wynikach kompresji dla wstępnie posortowanych danych w zależności od różnych metod sortowania.
Wykorzystane metody sortowania:

- Alfabetyczne,
- Losowe,
- Po liczbie różnych znaków,
- Rozmiar stringa,
- Brute force z użyciem różnych miar (dla małych instancji),
- Heurystyczne z użyciem różnych miar,
- Zachłanne z użyciem różnych miar.

Problem optymalnego sortowania zbioru wejściowego, aby wyniki kompresji były najlepsze jest problemem NP-trudnym i w większości wykorzystanych metod wymaga dużej mocy obliczeniowej i czasu wykonania. Do testów wykorzystaliśmy 500 i 100 elementowe podzbiory pliku „Scholar.csv”, podzbiory te zawierały losowe stringi z tego zbioru, które miały między sobą podobieństwo bliskie zeru. Wielkość tych zbiorów pozwoli nam zbadać wpływ metod na wyniki kompresji zbiorów oraz sprawdzić, czy wykorzystanie sortowań bądź heurystyk będzie opłacalne przed kompresją danych.

Opis wykorzystanych metod:

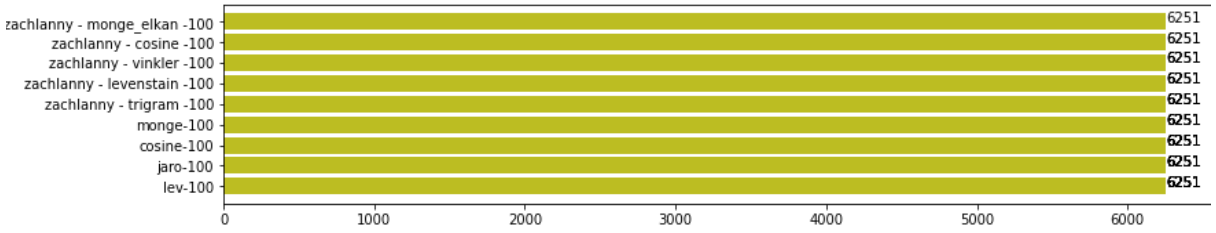
1. Alfabetyczne - przygotowany zbiór wejściowy został ułożony w kolejności alfabetycznej
2. Losowe - zbiór ułożony losowo
3. Po ilości różnych znaków- przygotowany zbiór wejściowy został posortowany od stringów posiadających jak najmniejszą liczbę różnych znaków do największej liczby różnych znaków
4. Rozmiar stringa - sortowanie zbiorów od najkrótszych stringów

5. Brute force z użyciem miar (dla małych instancji) - rozwiązanie to ze względu na dużą złożoność nie nadawało się do przetestowania dla zbiorów 100 i 500-elementowych
6. Heurystyczne z użyciem różnych miar
    - trigram
    - levenstein
    - jaro-winkler
    - cosine
    - monge-elkan

    Dla każdej z miar wykorzystaliśmy heurystykę solve_tsp_simulated_annealing. Heurystyka ta polega na obliczaniu macierzy odległości na podstawie miar prawdopodobieństwa.

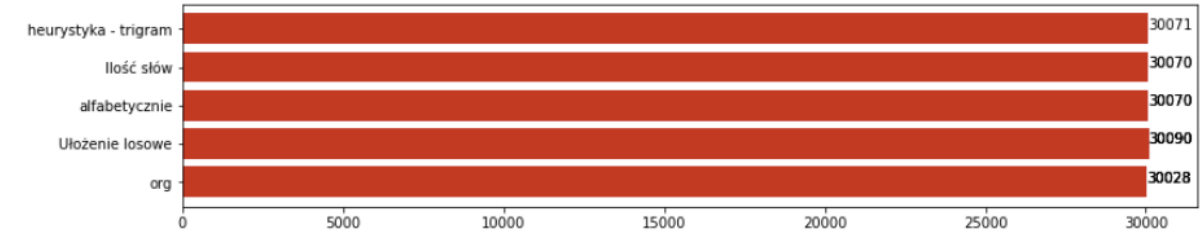
7. Zachłanne z użyciem różnych miar
    - dla wszystkich miar z poprzedniego punktu zastosowaliśmy metodę zachłanną. Jako początkowy element wybierany był pierwszy element ze zbioru, a następne na podstawie najlepszych wartości miar, jakie zostały otrzymane dla danej iteracji algorytmu.

Wykres nr 5 przedstawia długość ciągu dla zbiorów 100-elementowych, skompresowanych różnymi metodami. Można zauważyć, że kompresja nie zmieniła się względem zastosowanych metod, wynika to z małego podobieństwa wykorzystanych elementów i małego wpływu sortowań.



wykres 5: wyniki kompresji dla zbioru 100 elementowego

Wykres nr 6 przedstawia długość ciągu dla zbiorów 500-elementowych skompresowanych różnymi metodami. Można zauważyć, że w tym wypadku metody te dały gorsze wyniki niż uszeregowanie oryginalne.



wykres 6: wyniki kompresji dla zbioru 500 elementowego

Wnioski:

Wstępne sortowanie zbiorów ma znikomy wpływ na kompresję. Wykorzystanie prostych sortowań w niektórych zbiorach może mieć sens ze względu na niską złożoność obliczeniową natomiast stosowanie heurystyk mimo długiego czasu wykonywania i dużej złożoności nie daje zadowalających efektów.

6. Testowanie wpływu różnych metod podobieństwa na jakość kompresji

Eksperyment ten polegał na sprawdzeniu, czy podobne rekordy dają poprawę kompresji, względem zbioru gdzie to podobieństwo jest mniejsze. Do wyznaczenia miary podobieństwa wykorzystane zostały następujące miary:

- Levenstein,
- Winkler-Jaro,
- Tri-gram,
- Cosine,
- Monge Elkan.

Do testów posłużyliśmy się zbiorem 370102 słów ze słownika języka angielskiego[1]. Wpierw obliczane było podobieństwo każdego ze słów do słowa „flower” z wykorzystaniem powyższych miar. Kompresji były poddawane rekordy o podobieństwie z różnych przedziałów. Ze zbioru było brane pierwsze 1000 rekordów, które mieściły się w przedziale. Oznacza to, że wyrazy były posortowane w sposób alfabetyczny, ponieważ wejściowy zbiór miał właśnie taki porządek. Wybór krańców przedziału wymagał, aby w każdym z nich znajdowało się minimum 1000 rekordów. Uznaliśmy, że jest to już reprezentatywna próbka. Rozmiar skompresowanego zbioru był wyznaczany jako delta zmian sąsiadujących rekordów. Jeżeli delta zmian w stosunku do poprzedniego rekordu była mniejsza od obecnego, wówczas dodawany był jej rozmiar. Jeżeli okazywało się, że delta była większa, niż rozmiar rekordu, wtenczas dodawany był rozmiar rekordu. Jedyna wartość, przy której nie była obliczana różnica, był to rekord pierwszy. Dodatkowo doliczany był jeszcze jeden bit do każdego rekordu, który informowałby czy dana wartość jest oryginalna, czy jest deltą zmian od poprzedniego rekordu.

Deltę zmian postanowiliśmy obliczać ze wzoru:

$$\Delta = L * 3$$

Gdzie L jest odległością Levenshteina pomiędzy badanymi słowami. Ta reprezentacja nie jest obarczona narzutem wybranej technologii. Współczynnik kompresji obliczany jest ze wzoru:

$$WK = \frac{RozmiarPoSkompresowaniu}{RozmiarPrzedSkompresowaniem}$$

Zbiory 1000-elementowe zawierające słowa, których podobieństwa względem słowa „flower” znajdują się w przedziale <minimalne podobieństwo; maksymalne podobieństwo>, dały następujące wyniki dla badanych miar:

A. Levinstein

minimalne podobieństwo	maksymalne podobieństwo	współczynnik kompresji (mniej = lepiej)
0.1	0.2	0.8152
0.2	0.3	0.8587
0.3	0.4	0.8531
0.4	0.49	0.8828

tabela 6: wyniki dla miary Levinsein

B. Winkler-Jaro

minimalne podobieństwo	maksymalne podobieństwo	współczynnik kompresji (mniej = lepiej)
0.0	0.4	0.8250
0.4	0.6	0.8054
0.6	0.7	0.7918
0.7	0.9	0.8295

tabela 7: wyniki dla miary Winkler-Jaro

C. Tri-gram

minimalne podobieństwo	maksymalne podobieństwo	współczynnik kompresji (mniej = lepiej)
0.03	0.04	0.8313
0.04	0.11	0.8353
0.11	0.15	0.8539
0.15	0.18	0.8318

tabela 8: wyniki dla miary Tri-gram

D. Cosine

minimalne podobieństwo	maksymalne podobieństwo	współczynnik kompresji (mniej = lepiej)
0.02	0.50	0.7858
0.50	0.60	0.8922
0.60	0.70	0.9007
0.7	1.0	0.8632

tabela 9: wyniki dla miary Cosine

E. Monge Elkan

minimalne podobieństwo	maksymalne podobieństwo	współczynnik kompresji (mniej = lepiej)
0.02	0.50	0.8003
0.50	0.60	0.7906
0.60	0.70	0.8221
0.7	1.0	0.8185

tabela 10: wyniki dla miary Monge Elkan

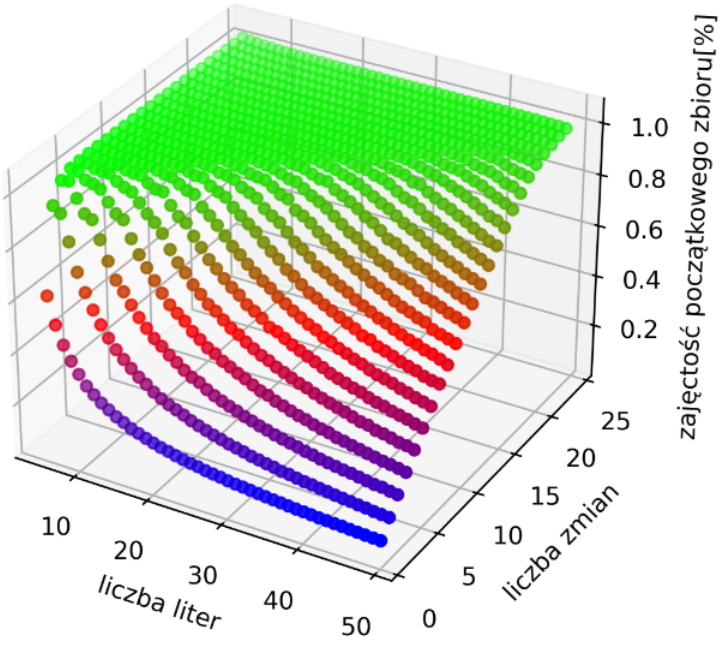
Z pomiarów wynika, że nie ma związku pomiędzy stopniem kompresji a podobieństwem słów. Wniosek ten wydaje się być mało prawdopodobny, więc postanowiliśmy powtórzyć eksperyment dla zwiększonego progu podobieństwa. Wiąże się to z tym, że próbki nie mają równej liczebności. Wykorzystaliśmy miarę podobieństwa Levinsteina. Tym razem można zauważyć, że wraz ze wzrostem podobieństwa zwiększa się poziom kompresji, jednakże liczba podobnych rekordów znacząco się zmniejsza (tabela 10).

minimalne podobieństwo	liczebność zbioru	współczynnik kompresji (mniej = lepiej)
0.49	1885	0.8110
0.50	577	0.8031
0.59	293	0.8100
0.63	170	0.7962
0.70	48	0.7784

tabela 11: wyniki dla testu wpływu większych podobieństw z wykorzystaniem mary Levinseina na współczynnik kompresji

Ostatnim eksperymentem było zbadanie opłacalności w zależności od długości ciągu znaku oraz ilości zmian w stosunku do poprzedniego rekordu. W tym podejściu słowa były generowane sztucznie. Polegało to na tym, że najpierw generowane było słowo w sposób losowy składające się z dużych i małych liter. Następnie na słowie były dokonywane losowe zmiany, polegające na zmianie przypadkowych liter na inne również losowe litery. Długość utworzonych słów była z zakresu od 5 do 50 znaków, natomiast zmiany były dokonywane od 1 do 25 razy.

Z wykresu nr 7 wynika, że kompresja jest najbardziej opłacalna gdy rekordy są większe, a zmiany pomiędzy nimi niewielkie.



wykres 7: współczynnik kompresji w zależności od ilości zmian oraz długości słów

Wnioski:

Współczynnik podobieństwa w zbiorze ma pozytywny wpływ na jego kompresję, lecz zaczyna on wpływać na wyniki kompresji dopiero w pewnych warunkach. Pozytywny wpływ w kompresji można zauważyć dla zbiorów z większą wartością współczynnika podobieństwa zbioru, a dla wartości współczynnika mniejszych niż pewien próg, różnice w wynikach w kompresji zbiorów nie są statystycznie istotne.

Największym problemem, który napotkaliśmy w tym eksperymencie, było poprawne oszacowanie rozmiaru delty zmian. Początkowo sprawdzaliśmy rozmiar funkcją getsizeof z biblioteki sys, okazało się, że nie nadaje się ona do złożonych typów, jakimi są delty zmian utworzone za pomocą biblioteki diff-match-patch. Następnie do określenia rzeczywistego rozmiaru została wykorzystana funkcja opisana w wątku [2]. Niestety okazało się, że nie daje ona oczekiwanego rezultatu. Narzuty związane z wykorzystaniem języka Python okazały się zbyt duże, przez co doliczane były dodatkowe bajty do rozmiarów. Miało to na tyle duży wpływ, że dopiero wyrazy dłuższe niż 10 liter w sytuacji zmiany jednego znaku (w stosunku do poprzedniego rekordu) sprawiały, że opłacalnym stała się kompresja. Postanowiliśmy więc, że lepszym pomysłem będzie reprezentować deltę jako różnicę pojedynczych zmian, czyli odległością Levenshteina pomnożoną przez 3, co dało nam liczbę bajtów niezbędnych do zapisania różnicy. Uznaliśmy, że taka reprezentacja wielkości będzie niezależna od wykorzystanego języka programowania, dzięki czemu wyniki eksperymentów będą uniwersalne.

7. Podsumowanie

Z przeprowadzonych badań wynika, że stosowanie kompresji typu delta, może być stosowane tylko przy zachowaniu odpowiednich warunków dotyczących analizowanych danych. Przy próbie kompresowania względnie losowych danych, niezależnie od sposobu sortowania, nie udało się uzyskać kompresji lub była ona względnie niewielka. Kompresja zaczyna być opłacalna dopiero gdy sąsiadujące rekordy cechują się bardzo dużym podobieństwem, co w praktyce oznacza małą liczbę edycji. Najlepsze wyniki osiągnęliśmy dla długich rekordów z niewielką liczbą zmian.

Należy zauważyć, że nawet dla krótszych wyrazów, przy niewielkiej liczbie zmian, wraz ze wzrostem podobieństwa wyrazów ciągu, następuje poprawa kompresji. Uzyskane wyniki sugerują pole i kierunek dalszych badań.

Źródła

[1] link do zboiru słów https://github.com/dwyl/english-words

[2] link do wątku
https://stackoverflow.com/questions/13530762/how-to-know-bytes-size-of-python-object-like-arrays-and-dictionaries-the-simp