Spis treści

Rozdział 1

Wprowadzenie

1.1 Opis problemu

Obecnie prowadzonych jest wiele badań nad sieciami społecznymi używającymi etykiet (tzw. tagów, czy adnotacji) do porządkowania zasobów użytkownika. Część z nich za cel stawia sobie ulepszenie algorytmów wyszukiwania dokumentów, inne używają tych zasobów dla personalizacji wyników dla użytkowników, rekomendacji dokumentów, użytkowników, czy tagów w czasie procesu etykietowania.

Autorzy prac [15] i [6] analizują użyteczność etykietowania dokumentów przy wyszukiwaniu. Ich analizy bazują głównie na danych systemu delicious. Wnioski z tych prac są pozytywne. Zwracają one uwagę na korelacje miedzy popularnością stron w serwisie, a ich rankingiem według algorytmu PageRank. Dodatkowo wskazują na problemy z nowymi dokumentami dodanymi do sieci, z którymi mają problemy algorytmy bazujących na odnośnikach pomiędzy dokumentami, a które mogą być rozwiązane przez strony takie jak delicous. Również rozwiązanie problemu nowych danych jest zaproponowane w pracy [2]. W tym artykule, jako rozwiązania, autorzy proponują użycie informacji pobranych z systemu twitter, który służy do mikroblogowania. Pod uwagę brane są krótkie informacje umieszczane przez użytkowników zawierające odnośniki do dokumentów jak również inne informacje dostępne w ich profilach.

Autorzy pracy zaprezentowali formalny model folksonomi. Folksonomią nazywamy społeczne klasyfikowanie czy tagowanie różnych zasobów. Formalnie model folksonomii został przedstawiony w pracy [4] i zdefiniowany został jako:

Definicja 1. *Folksonomia jest to krotka* $F := (U, T, D, Y)$*, gdzie:*

- U*,*T *i* D *są to zbiory skończone, których elementy nazywają się odpowiednio użytkownicy, tagi i dokumenty*

- *Y jest relacją między tymi elementami: $Y \subseteq U \times T \times D$, nazywamy ją relacją przypisania tagu.*

Również w pracy [4] zaprezentowano algorytm AdaptedPageRank i jego bardziej spersonalizowana wersję algorytm FolkRank. Oba te algorytmy bazują na metodzie PageRank. Algorytm ten pozwala na rekomendacje tagów dla użytkowników jak również na ustalenie rankingu w wyszukiwanych elementach. Dokładny opis tych algorytmów znajduje się kolejnych rozdziałach.

W pracy [1] przedstawione zostały algorytmy SocialSimRank i SocialPageRank. Algorytm SocialPageRank jest statycznym rankingiem zasobów, opartym również o ideę algorytmy PageRank. Algorytm ten bierze pod uwagę ilość tagów wskazujących na dany dokument jak również różną wagę opisujących dokument etykiet. Drugim proponowanym w tej pracy algorytmem jest SocialSimRank, który estymuje podobieństwo między używanymi tagami, a następnie wyniki te są używane dla wyliczanie podobieństwa między zapytaniem użytkownika a tagami przypisanymi do danego zasobu. Ten algorytm również zostanie dokładniej opisany w kolejnych rozdziałach.

W pracach [9] i [7] autorzy biorą pod uwagę całą sieć społecznościową użytkownika i wykorzystują te dane dla przedstawienia spersonalizowanych wyników. W artykule [9] opisywany jest algorytm ContextMerge, który wykonuje wyszukiwanie biorąc pod uwagę zależności między użytkownikami. Dodatkowo pozwala na dynamiczne dodawanie nowych wyników do odpowiedzi dla uzyskania k. W pracy [7] autorzy zaproponowali system, który łączy wyniki wyszukiwarki z danymi użytkownikami aplikacji takich jak delicous. Jako bazowa wyszukiwarka może być użyta dowolna aplikacja, która zwraca dokumenty wraz z przypisanym do nich rankingiem. Wyniki te są następnie przeliczane za pomocą danych uzyskanych od użytkownika, czyli ze zbiorem dokumentów i tagów opisanych przez niego i innych użytkowników. Jako rezultat otrzymujemy nowy, bardziej spersonalizowany ranking dokumentów.

W pracy [8] celem autorów było spersonalizowanie wyników wyszukiwania w dokumentach z tagami. Do tego celu stworzone zostały dwa modele: użytkownik-tag, który ukazuje jak użytkownik użył tagi podobne do wybranego tagu. Drugim modelem jest model tag-zasób opisujący w jaki sposób dokumenty podobne do danego zasobu zostały opisane etykietami.

Probabilistyczne podejście do rankingu dokumentów przy użyciu tagów zostało zaprezentowane w pracy [16] i [5]. W [16] autorzy zaprezentowali model probabilistyczny dla generowania etykiet i zależnych im dokumentów i wyszukiwania ich tematów. W [5] autorzy prezentują metodę rankingu tagów w zależności od ich tematu i konstruują graf przejścia między tagami należącymi do różnych tematów. Metoda ta jest wykorzystana następnie dla rekomendacji tagów dla użytkownika w czasie opisywania dokumentów.

1.2 SocialPageRank

1.2.1 Opis

Ilość tagów przypisanych do danej strony może świadczyć o jej popularności i jakości. Jednak tradycyjne algorytmy wyliczające statyczny ranking strony nie są w stanie poprawnie wyliczyć popularności strony w oparciu o posiadane dane. Na przykład istnieją strony mające bardzo dużą liczbę opisanych tagów, jednak posiadają ranking PageRank równy zero. Dodatkowo nie można brać pod uwagę tylko ilość tagów, różne tagi mogą mieć różną wagę w trakcie wyliczania popularności danej strony.

Można zauważyć następującą relację między aktywnymi użytkownikami, popularnymi tagami i stronami o wysokiej jakości:

Obserwacja 1. *Popularne strony są dodawane przez wielu aktywnych użytkowników i opisywane popularnymi tagami. Aktywni użytkownicy lubią dodawać strony o wysokiej jakości i opisywać je popularnymi tagami. Popularne tagi są używane do opisywania popularnych stron przez aktywnych użytkowników.*

Bazując na powyższej obserwacji stworzony został algorytm SocialPageRank, służący do wyliczania rankingu strony w systemie opartym na socjalnym tagowaniu. Algorytm działa na zasadzie wzmacnia wzajemnych relacji między użytkownikami, tagami i opisanymi zasobami.

Załóżmy, że posiadamy N_T tagów, N_D stron internetowych i N_U użytkowników. Niech M_{DU} będzie $N_D \times N_U$ macierzą asocjacyjną między zasobami i użytkownikami, M_{UT} będzie macierzą $N_U \times N_T$ asocjacyjną między użytkownikami i tagami i niech M_{TD} będzie macierzą $N_T \times N_D$ asocjacyjną miedzy tagami a dokumentami. Macierze te zostały wypełnione w następujący sposób:

- W komórce macierzy$M_{DU}(d_n, u_k)$ znajduje się wartość będąca ilością tagów przypisanych do dokumentu d_n przez użytkownika u_k.
- Elementy $M_{UT}(u_k, t_n)$ to ilość dokumentów opisanych tagiem t_n przez użytkownika u_k.
- Elementy $M_{TD}(t_n, d_k)$: ile użytkowników dodawało dokument d_k i oznaczyło go tagiem t_n.

Niech D_0 będzie wektorem zainicjalizowanym losowymi wartościami z przedziału $[0, 1]$. Jest on pierwszym przybliżeniem rankingu dokumentów.

1.2.2 Algorytm

Dane wejściowe:

Macierze asocjacyjne M_{DU}, M_{UT} ,M_{TD} i zainicjalizowany wektor D_0.

repeat
$U_i = M_{DU}^T * D_i$
$T_i = M_{UT}^T * U_i$
$D'_i = M_{TD}^T * T_i$
$T'_i = M_{TD} * D'_i$
$U'_i = M_{UT} * T'_i$
$D_{i+1} = M_{DU} * U'_i$
$D_{i+1} = norm(D_{i+1})$
until wartości wektora D_n nie zbiegną

Wektory U_i, T_i i D_i opisują popularność użytkowników, tagów i dokumentów w i-tej iteracji. U'_i, T'_i i D'_i są wartościami pośrednimi. Z algorytmu powyżej wynika, że popularność użytkownika może zostać wyznaczona z popularności stron, które zostały przez niego opisane. Popularność tagów wyliczana jest z popularności użytkowników. Następnie w algorytmie 'popularność' jest przenoszona między tagami, do stron, od stron do użytkowników. Na koniec od wektor 'popularności' propaguje się od użytkowników do dokumentów. Wartości wektora D_n zbiegają do wektora D^*, który jest wynikiem działania algorytmu SocialPageRank.

Złożność

Złożoność czasowa każdej iteracji wynosi $O(N_u * N_d + N_t * N_d + N_t * N_u)$. Ponieważ macierze utworzone z posiadanych danych charakteryzują się tym, że są bardzo rzadkie, czas wykonywania jest mniejszy. Należy również pamiętać, że dane w sieci internetowej rosną bardzo szybko i czas trwania całych obliczeń będzie również szybko wzrastał wraz ze zwiększającym się zbiorem danych.

1.3 Adapted PageRank

1.3.1 Opis

Algorytm Adapted PageRank podobnie jak algorytm SocialPageRank został zainspirowany algorytmem PageRank. Algorytm ten od algorytmu SocialPageRank różni się sposobem tworzenia macierzy. Zamiast używania trzech macierzy, używana jest jedna, utworzona z grafu $G_f = (V, E)$.

Ponieważ algorytm PageRank nie może być bezpośrednio zastosowany do zebranych danych autorzy algorytmu zmienili strukturę danych na nieskierowany graf trzydzielny $G_f = (V, E)$.

Proces tworzenia grafu G_f:

1. Zbiór wierzchołków V powstaje z sumy rozłącznej zbioru użytkowników U, tagów T, i dokumentów D: $V = U \cup T \cup D$

2. Wszystkie wystąpienia łącznie tagów, użytkowników, dokumentów stają się krawędziami grafu G_f: $E = \{\{u,t\},\{t,d\},\{d,u\}|(u,t,d) \in Y\}$. Wagi tych krawędzi są przydzielane w następujący sposób: każda krawędź $\{u,t\}$ ma wagę $|\{d \in D : (u,t,d) \in Y\}|$, czyli jest to ilość dokumentów, którym użytkownik u nadał annotacje t. Analogicznie dla krawędzi $\{t,d\}$: $waga = |\{u \in U : \{(u,t,d) \in Y\}\}|$ i krawędzi $\{d,u\}$, gdzie $waga = |\{t \in T : \{(u,t,d) \in Y\}\}|$. Wagi te są analogiczne do wartości w macierzach w algorytmie Social PageRank.

Oryginalny algorytm PageRank bazuje na idei, że strona jest ważna jeśli dużo stron ma do niej odnośniki, i te strony są również ważne. Rozłożenie wag może zostać przedstawione jako punkt stały procesu przekazywania wag grafu sieci www. W tym algorytmie została zaimplementowana podobna analogia przeniesiona na folksonomie: strony, które zostały opisane ważnymi tagami, przez ważnych użytkowników stają się ważne. Analogiczna zasada odnosi się tagów i użytkowników. Dzięki temu otrzymujemy graf, w którym zależności między wierzchołkami wzajemnie się wzmacniają w czasie rozprzestrzenienia się wag.

Jak w algorytmie PageRank, tutaj również został użyty model losowego internauty (random surfer model), czyli definicje ważności strony opierającą się o idealnego losowego internautę, który najczęściej przechodzi przez strony internetowe używając odnośników. Jednak od czasu do czasu, internauta przeskoczy losowo do nowej strony, nie przechodząc po żadnym z odnośników. Używając tego założenia dla folksonomii otrzymujemy:

$$\vec{w} = \alpha\vec{w} + \beta A\vec{w} + \gamma\vec{p} \tag{1.1}$$

Gdzie A jest prawo stochastyczną macierz sąsiedztwa grafu G_f, $\vec{p}$ jest wektorem preferencji, odpowiadającym za losowego internautę. Wektor preferencji używany jest do obliczeń spersonalizowanych wyników. W przypadku algorytmu Adapted PageRank $\vec{p} = 1$. α, β i γ to stałe, gdzie: $\alpha, \beta, \gamma \in [0,1]$ i $\alpha + \beta + \gamma = 1$. Stała α wpływa na prędkość zbieżności algorytmu, β i γ regulują wpływ wektora preferencji na obliczenia.

1.3.2 Algorytm

Dane wejściowe:

A – prawo stochastyczna macierz sąsiedztwa grafu G_f
p – wektor preferencji
w – losowo zainicjalizowany wektor

repeat
$w_{n+1} = \alpha * w_n + \beta * A * w_n + \gamma * p$
until wartości wektora w_n nie zbiegną

wynikiem algorytmu jest wektor $\vec{w}$.

Dane zostały uzyskane dla paremetrów: $\alpha = 0.35, \beta = 0.65, \gamma = 0$ i pochodzą z pracy [4]. Wektor preferencj $\vec{p} = 1$.

1.3.3 Algorytm FolkRank

Algorytm FolkRank jest wersją algorytmu Adapted PageRank, który daje różne wyniki w zależności od tematu, czyli wektora preferencji $\vec{p}$. W poprzednim algorytmie wszystkie pola były ustawione na 1. Wektor ten może być ustalony na wybrany zbiór tagów, użytkowników, dokumentów, albo na pojedynczy element. Wybrany temat będzie propagowany na resztę dokumentów, tagów i użytkowników. Można dzięki temu, ustając wagę na zapytanie albo konto użytkownika systemu uzyskać wyniki bardziej zbliżone do zainteresowań danego użytkownika.

Algorytm FolkRank może pomóc w analizie zbioru danych, albo w systemach nie działających w czasie rzeczywistym, ale nie jest użyteczny w zaprezentowanym systemie. Nie jest możliwy do wykorzystania z powodu długiego czasu obliczania wag dokumentów.

1.4 Lucene

Lucene jest biblioteką napisaną w Javie. Framework ten jest w stanie indeksować dużą ilość dokumentów z różnych źródeł i przeprowadzać szybkie wyszukiwania w tych tekstach. W opisywanej aplikacji framework Lucene przechowuje również źródła stron.

Odnośniki do tych strony zostały pobrane z serwisu delicous. Następnie dla wszystkich odnośników pobierana jest treść strony. Każda z tych pobranych stron jest następnie oczyszczana ze znaczników HTML i przekazana do zapisania na dysk przy użyciu Lucene.

Do Lucene zapisywane są następujące informacje: identyfikator *id* dokumentu z bazy danych, oraz przetworzony tekst strony WWW. Przechowywanie identyfikatora dokumentu w danych Lucene pozwala na późniejsze powiązanie wyników wyszukiwania z odpowiednim rekordem w bazie danych.

W czasie zapisywania Lucene przeprowadza dodatkowe operacje na tekstach stron. Czynności te pozwolą na lepsze i szybsze indeksowanie i wyszukiwanie. Są to miedzy innymi tokenizacja, usunięcie końcówek wyrazów, usunięcie tzw 'stop-words'.

Tokenizacja polega na rozbiciu tekstu na tokeny reprezentujące pojedyncze, indeksowane słowa. Ma to duży wpływ na późniejsze wyszukiwanie w tekscie. Po tokenizacji następuje proces mający na celu otrzymanie trzonu (ang. stem) danego słowa. Powodem tej czynności jest to, że w tekście występują różne gramatyczne wariacje słów. Dzięki temu, słowo 'rowery' zostanie zamienione na wyraz 'rower'. Jeśli będziemy wyszukiwać

dokumenty zawierające słowo 'rower', otrzymamy również te, zawierające różne formy tego słowa, np: 'rowery' czy 'rowerowy'.

Następnie z tekstu zostają usunięte stop-słowa (ang. stop-words). Są to wyrazy, nie wnoszące dużo do treści tekstu. W języku angielskim będą to na przykład słowa: 'the', 'a', 'and', 'about', 'what', ... Usunięcie tych słów przyśpiesza proces wyszukiwania i przyczynia się do poprawy jakości wyników. Kolejną czynnością jest normalizacja tekstu w trakcie której następuje usunięcie akcentów ze słów.

Ponieważ znaczna część posiadanych w bazie danych tekstów jest w języku angielskim, używana jest tylko lista stop-słów z języka angielskiego. Podobnie przy wyszukiwani trzonów słowa i normalizacji używana jest gramatyka języka angielskiego

Wyszukiwanie

Lucene łączy w sobie model binarny z modelem przestrzeni wektorowej (ang. Vector Space Model). Po otrzymaniu zapytania, na początku przeprowadzane jest wyszukiwanie w modelu binarnym. Wyszukiwanie przy użyciu modelu binarnego wskazuje dokumenty zawierające dane słowa. Możliwe jest użycie dzięki temu wyrażeń logicznych: 'AND', 'OR',... w zapytaniu. Otrzymujemy dzięki temu zdecydowanie mniejszy zbiór dokumentów. Następnie na tych dokumentach następuje wyszukiwanie przy pomocy modelu wektorowego.

W tym modelu, dokumenty reprezentowane są jako wektory ważone w wielowymiarowej przestrzeni. Każdy element wektora odpowiada termom z dokumentu, a jego wartość wyliczana jest przy użyciu funkcji tf-idf. Tf-idf informuje o częstości wystąpienia termów uwzględniając jednocześnie odpowiednie wyważenie znaczenia lokalnego termu i jego znaczenia w kontekście pełnej kolekcji dokumentów [13].

$$(tf - idf)_{i,j} = tf_{i,j} \times idf_i \tag{1.2}$$

$tf_{i,j}$ to częstotliwość występowania danego termu w dokumencie (ang. term frequency), wyraża się wzorem:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{1.3}$$

Gdzie $n_{i,j}$ jest liczbą wystąpień termu (t_i) w dokumencie d_j, a mianownik jest sumą liczby wystąpień wszystkich termów w dokumencie d_j.

idf_i to odwrotna częstość w dokumentach (ang. inverse document frequency), wyrażana wzorem:

$$idf_i = \log \frac{|D|}{|\{d \in D : t_i \in d\}|} \tag{1.4}$$

Gdzie:

- $|D|$ liczba wszystkich dokumentów,
- $|\{d : t_i \in d\}|$: liczba dokumentów zawierających przynajmniej jedno wystąpienie danego termu

Zapytanie jest przekazywane do frameworku, w którym jest one przekształcane na termy. Dla zapytania q i dla każdego dokumentu d wyliczana jest wartość funkcji $score(q, d)$:

$$score(query, doc) = coord(query, doc) \cdot queryNorm(query) \cdot \sum_{t \in q} (tf(t \in q) \cdot idf(t)^2) \tag{1.5}$$

Gdzie:

- tf, idf - Częstość termów i odwrotna częstość w dokumentach, które zostały opisane powyżej.
- $coord(query, doc)$ - wspolczynnik wyliczany na powstanie ilosci termów zapytania, które znajdują sie w wybranym dokumencie. Najczęscie, dokument w którym znajduje sie więcej termów zapytania otrzyma wiekszy wynik niz dokument z mniejsza ilością.
- $queryNorm(query)$ - czynnik normalizujacy wynik. Jego zadanie polega na umożliwieniu porównywania wyników różnych zapytań.

Czas wyszukiwania zapytania w dokumentach przechowywanych Lucene jest szybkie. Przy małej ilości danych (poniżej 1GB danych), wyszukiwanie następuje praktycznie w czasie rzeczywistym. Wynikiem są dokumenty posortowane według wyniku funkcji *score* [3].

Rozdział 2

Opis problemu

Praca ma na celu zbadanie problemu wyszukiwania dokumentów w wybranym serwisie społecznościowym. Badany tutaj serwis jest zbiorem odnośników do zasobów internetowych. Każdy z nich został dodanych przez użytkowników serwisu i opisanych odpowiednimi tagami, zwanymi również etykietami. Przykładem serwisów działających w opisany tutaj sposób jest delicous czy flickr. Wynikiem pracy będzie system pozwalający na wyszukiwanie w zebranym zbiorze dokumentów, sortujący wyniki w zależności od wybranych rankingów dokumentów i prezentujący otrzymane wyniki użytkownikowi.

W pracy tej zostanie zaprezentowane kilka algorytmów wyszukujących w zbiorze dokumentów. Jako pierwsze opisane zostaną dwa algorytmy: Adapted PageRank i SocialPageRank. Algorytmy te bazują na popularnym algorytmie PageRank. Kolejnym wykorzystanym algorytmem jest wersja algorytmu TF-IDF, który wyszukuje i porządkuje dokumenty ze względu na zapytanie i treść dokumentu. Jako dodatkowy element, zostaną użyte również dane mówiące o popularności danych wyników pozyskane z innych serwisów społecznościowych, takich jak twitter, facebook czy digg. Dane z tych stron zostały również wykorzystane w trakcie testów, do oceniania poprawności wyników.

W pracy opisane powyżej algorytmy zostaną przetestowane na zbiorze zebranych danych. Zbiór ten składa się z około 300 000 dokumentów, 200 000 użytkowników i 80 000 tagów pobranych z serwisu delicous.com. Na koniec zostanie zaprezentowany ranking łączący wszystkie zaimplementowane algorytmy.

FOLKSONOMIA

Użytkownicy i tagi identyfikowani są na podstawie ich unikalnych nazw własnych i identyfikatorów. Dokumenty mogą być różnymi danymi: stronami www, zdjęciami, plikami np: pliki pdf. Ta praca bazuje na danych pobranych z witryny delicous, które są stronami internetowymi. Dane, które nie spełniają tego warunku, nie są brane pod uwagę w tej pracy.

2.1 Opis metod rankingu dokumentów

Do wyliczenia rankingu dokumentów zostało użyte kilka metod. Pierwsze dwie metody to statyczne rankingi Adapted PageRank i SocialPageRank. Algorytmy te biorą pod uwagę popularność tagów, użytkowników, dokumentów i wzajemne relację między nimi.

Kolejnym użytym w pracy rankingiem jest wersja algorytmu TF-IDF zaimplementowana we frameworku Lucene.

Ostatnią metodą stanowiącą o popularności dokumentów są dane z serwisów takich jak facebook, twitter czy digg mówiące o ilości udostępnień danego zasobu przez użytkowników tych serwisów.

2.2 Inne rankingi

Opisywana tutaj aplikacja korzysta również z danych z innych serwisów internetowych do wyliczania popularności dokumentu. Wybrane zostały trzy popularne serwisy internetowe: digg, twitter i facebook.

Digg to serwis internetowy tworzony przez swoich użytkowników, zajmujący się gromadzeniem i ocenianiem linków do potencjalnie interesujących treści w internecie. Wiadomości, nowości czy linki do ciekawych stron lub blogów są dodawane przez zarejestrowanych użytkowników. Każdy taki wpis podlega ocenie innych użytkowników, którzy mają możliwość głosować na niego, w ten sposób 'awansując' go w punktacji setek innych wiadomości. Za pomocą systemu punktacji najlepsze wpisy ukazują się na stronie głównej serwisu [10].

Twitter to darmowy serwis społecznościowy udostępniający usługę mikroblogowania umożliwiającą użytkownikom wysyłanie oraz odczytywanie tak zwanych tweetów. Tweet to krótka, nieprzekraczająca 140 znaków wiadomość tekstowa wyświetlana na stronie użytkownika oraz dostarczana pozostałym użytkownikom, którzy obserwują dany profil. Użytkownicy mogą dodawać krótkie wiadomości do swojego profilu z poziomu strony głównej serwisu, wysyłając SMS-y lub korzystając z zewnętrznych aplikacji [14].

Facebook to serwis społecznościowy w ramach którego zarejestrowani użytkownicy mogą tworzyć sieci i grupy, dzielić się wiadomościami i zdjęciami oraz korzystać z różnych udostępnionych aplikacji. W grudniu 2011 roku liczba użytkowników na całym świecie wynosiła ponad 845 mln [11].

Każdy z tych serwisów posiada miarę pozwalającą stwierdzić o popularności zewnętrznej strony internetowej według jego użytkowników. Z serwisu digg pobrana została ocena danego dokumentu przyznana przez użytkowników. Twitter pozwala na sprawdzenie ilości wiadomości zawierających odnośnik do danej strony internetowej. W serwisie facebook

jako popularność danego dokumentu wzięta jest ilość użytkowników, którzy 'polubili' daną stronę internetową.

Serwisy te udostępniają API pozwalające na pobranie tych informacji i zapisanie ich w bazie danych. Z tych danych wyliczany jest następnie ranking poszczególnych dokumentów jako suma znormalizowanych 'popularności' danego dokumentu w kolejnych serwisach.

$$rank(doc_i) = norm(D_i) + norm(F_i) + norm(T_i) \tag{2.1}$$

Gdzie:

- doc_i - dokument i,
- D_i - ilość użytkowników udostepniających dokument i w serwisie digg,
- F_i - ilośc użytkowników udostępniających dokument i w serwisie facebook,
- T_i - ilość użytkowników udostępniających dokument i w serwisie twitter.

2.3 Łączenie rankingów

Wszystkie przedstawione powyżej rankingi dokumentów są łączone i wyliczany jest ostateczny ranking. Ranking ten jest brany jako ważona suma wszystkich innych rankingów:

$$\begin{aligned} final\_rank(doc_i) = w1 \cdot lucene\_score(doc_i) \\ + w2 \cdot social\_rank(doc_i) \\ + w3 \cdot adapted\_rank(doc_i) \\ + w4 \cdot other(doc_i) \end{aligned} \tag{2.2}$$

Gdzie w1, w2, w3, w4 są to wagi przydzielone wynikom otrzymanym odpowiednio przy pomocy Lucene, algorytmu SocialPageRank, Adapted PageRank i rankingowi według innych serwisów społecznościowych

Rozdział 3

Implementacja

3.1 Opis Aplikacji

Aplikację można podzielić na 3 główne części:

- interfejs użytkownika pozwalający na wyszukiwanie w zasobach i wyświetlający wyniki użytkownikowi.
- Główną część aplikacji, odpowiedzialna za preprocessing i wyliczanie algorytmów.
- Crawlery zbierające nowe dane i odświeżające już posiadane.

W pierwszej części odbywa się wykonywanie zapytań, wyliczenie ostatecznego rankingu dokumentów i wyświetlanie odpowiedzi użytkownikowi. Wykorzystany jest tutaj framework Lucene i wcześniejsze obliczenia wykonane w innych fragmentach aplikacji.

Druga część aplikacji zawiera głównie implementacje algorytmów Social PageRank, Adpated PageRank jak również odpowiada za wykonywanie wcześniejszych obliczeń (preprocessing), które przyśpieszają późniejsze wykonanie algorytmów.

Ostatnia część składa się z różnych wątków spełniających rolę crawlerów. Kilka z tych wątków pobiera i odświeża zapisane poprzednio dane z serwisu delicous. Inne odpowiadają za pobranie informacji o popularności danych zasobów w serwisach twitter, digg i facebook. Ostatni z crawlerów pobiera dla każdego dokumentu jego treść i zapisuje go przy użyciu frameworku Lucene na dysk. Dokładniejszy opis zbierania danych można znaleźć w rozdziale 5.1

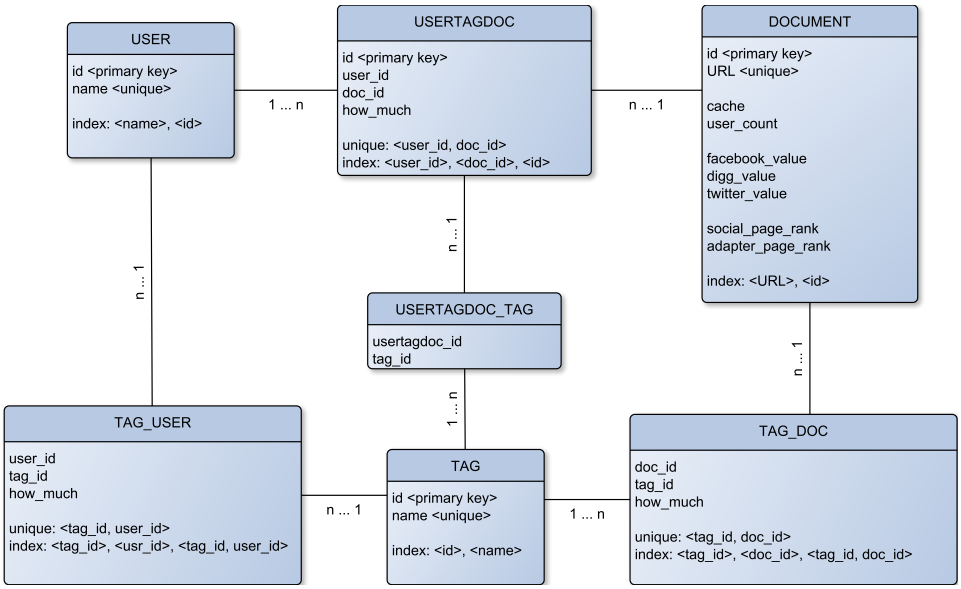
Wszystkie wymienione poprzednio fragmenty aplikacji korzystają z bazy danych, która bardziej szczegółowo zostanie opisana w kolejnych rozdziałach.

3.2 Baza danych

3.2.1 System zarządzania baza danych

Jako serwer bazy danych został użyty MySQL serwer z silnikiem bazy danych InnoDB. MySQL został wybrany głównie z powodu swojej szybkości działania jak również powodu prostoty używania. Dodatkowe funkcjonalności bazy danych, takie jak na przykład: funkcje, triggery, nie są używane w tej aplikacji.

3.2.2 Tabele



Rysunek 3.1: Schemat bazy danych

Baza danych składa się z trzech głównych tabel: USER, DOCUMENT i TAG. Zawierają one informację na temat, odpowiednio, użytkowników, dokumentów i adnotacji pobrane z serwisu delicous. W bazie danych znajdują się też tabele: USERTAGDOC i USERTAGDOC_TAG, które służą do zapisania relacji między użytkownikami a dokumentami (USERTAGDOC) i adnotacjami ich opisującymi (USERTAGDOC_TAG). W tych tabelach zapisane są informacje na temat tego czy dany użytkownik dodał dokument do serwisu i jakimi tagami została dana strona opisana.

3.2.3 Pomocnicze tabele i pola. Indeksy

Dodatkowo w bazie danych (rys 3.1)znajdują się dwie tabele: TAG_DOC i TAG_USR. W tabelach tych zapisywane są dane wyliczone z pozostałych

tabel. W tabeli TAG_DOC znajdują się informację na temat tego ilu użytkowników dodało dany dokument doc_i i opisało go tagiem tag_k. Odpowiednia w tabeli TAG_USR znajdują się informację na temat ilości różnych dokumentów dodanych przez użytkownika usr_n i opisanych tagiem tag_m. Dane ilości różnych tagów którymi użytkownik usr_l opisał dokument doc_j przechowywane są w już istniejącej tabeli USERTAGDOC.

Tabele TAG_DOC, TAG_USR i pole how_much w tabeli USERTAGDOC zostają wypełnione w czasie preprocessingu. Dane te posłużą później do utworzenia macierzy, używanych przez algorytmy Social PageRank i Adapted PageRank.

W tabelach na różnych polach zostały dodane indeksy. Przyśpieszają one działanie aplikacji, pozwalają na szybsze operacja przy często używanych polach. Główne indeksy zostały zaznaczone na rysunku 3.1

3.2.4 Listing

Poniżej znajduje się listing zapytania SQL tworzącego tabele w bazie danych.

Listing 3.1: Skrypt tworzący tabele w bazie danych

```
CREATE TABLE `DOCUMENT` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `url` varchar(255) NOT NULL,
  `digg_value` int(11) DEFAULT '0',
  `facebook_value` int(11) DEFAULT '0',
  `twitter_value` int(11) DEFAULT '0',
  `page_fetch` tinyint(1) DEFAULT '0',
  `tag_count` bigint(20) DEFAULT NULL,
  `user_count` bigint(20) DEFAULT NULL,
  `cache` text,
  `adapted_page_rank` double DEFAULT NULL,
  `social_page_rank` double DEFAULT NULL,
 PRIMARY KEY (`id`),
 UNIQUE KEY `url` (`url`)
)

CREATE TABLE `TAG` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `doc_count` bigint(20) DEFAULT NULL,
  `doc_dist_count` bigint(20) DEFAULT NULL,
  `tag` varchar(255) NOT NULL,
  `user_count` bigint(20) DEFAULT NULL,
```

```
  `adapted_page_rank` double DEFAULT NULL,
 PRIMARY KEY (`id`),
 UNIQUE KEY `tag` (`tag`)
)

CREATE TABLE `TAG_DOC` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `doc_id` bigint(20) DEFAULT NULL,
  `tag_id` bigint(20) DEFAULT NULL,
  `how_much` int(11) DEFAULT '1',
 PRIMARY KEY (`id`),
 UNIQUE KEY `tag_doc` (`doc_id`,`tag_id`),
 KEY `doc_id` (`doc_id`,`tag_id`),
 KEY `tag_doc_doc` (`doc_id`),
 KEY `tag_doc_tag` (`tag_id`)
)

CREATE TABLE `TAG_USR` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `user_id` bigint(20) DEFAULT NULL,
  `tag_id` bigint(20) DEFAULT NULL,
  `how_much` int(11) DEFAULT '1',
 PRIMARY KEY (`id`),
 UNIQUE KEY `tag_user` (`user_id`,`tag_id`),
 KEY `user_id` (`user_id`,`tag_id`),
 KEY `tag_usr_doc` (`tag_id`),
 KEY `tag_usr_usr` (`user_id`)
)

CREATE TABLE `USER` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `doc_count` bigint(20) DEFAULT NULL,
  `name` varchar(255) DEFAULT NULL,
  `new_data` tinyint(1) DEFAULT '1',
  `tag_count` bigint(20) DEFAULT NULL,
  `tag_dist_count` bigint(20) DEFAULT NULL,
  `adapted_page_rank` double DEFAULT NULL,
 PRIMARY KEY (`id`),
 UNIQUE KEY `name` (`name`)
)


CREATE TABLE `USERTAGDOC` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
```

```
  `doc_id` bigint(20) DEFAULT NULL,
  `user_id` bigint(20) DEFAULT NULL,
  `how_much` int(11) DEFAULT NULL,
 PRIMARY KEY (`id`),
 UNIQUE KEY `user_id` (`user_id`,`doc_id`),
 KEY `FKB30EFAE96714BC07` (`doc_id`),
 KEY `FKB30EFAE9520DD4E4` (`user_id`),
 CONSTRAINT `FKB30EFAE9520DD4E4` FOREIGN KEY (`user_id`) REFERENCES
`user` (`id`),
 CONSTRAINT `FKB30EFAE96714BC07` FOREIGN KEY (`doc_id`) REFERENCES
`document` (`id`)
)


CREATE TABLE `USERTAGDOC_TAG` (
  `USERTAGDOC_id` bigint(20) NOT NULL,
  `tags_id` bigint(20) NOT NULL,
 KEY `FK4C01D124CA702D03` (`tags_id`),
 KEY `FK4C01D1243D83CB28` (`USERTAGDOC_id`),
 CONSTRAINT `FK4C01D1243D83CB28` FOREIGN KEY (`USERTAGDOC_id`)
REFERENCES `usertagdoc` (`id`),
 CONSTRAINT `FK4C01D124CA702D03` FOREIGN KEY (`tags_id`) REFERENCES
`tag` (`id`)
)
```

3.3 Implementacja algorytmów SocialPageRank i AdaptedPageRank

Zarówno algorytm SocialPageRank i AdaptedPageRank wykorzystują w trakcie każdej iteracji sześć macierzy. Trzy z sześciu używanych macierzy zostały opisane poniżej. Kolejne trzy to ich transpozycje.

Używane macierze:

- M_{DU}: macierz $N_D \times N_D$ asocjacyjna między dokumentami a użytkownikami, która w każdej komórce $M_{DU}(d_m, u_k)$ zawiera ilość tagów, którymi dany użytkownik u_k opisał wybrany dokument d_m.
- M_{UT}: macierz $N_U \times N_T$ asocjacyjna między użytkownikami a tagami, zawierająca w komórce ilość dokumentów opisanych wybranym tagiem przez danego użytkownika/
- M_{TD}: macierz $N_T \times N_D$ asocjacyjna między tagami a dokumentami, zawierająca w komórach liczbę użytkowników, którzy dany dokument opisali wybranym tagiem.

Algorytmy te działały na danych składających sie z około 200,000 użytkowników, 300,000 dokumentów i 80,000 tagów. Macierze utworzone z tych danych są duże. Dodatkowo, w algorytmie Adapted PageRank używamy macierzy, która jest stworzona ze wszystkich 6-ciu opisanych powyżej:

$$G_f = \begin{pmatrix} 0 & M_{DU} & M_{TD}^T \\ M_{DU}^T & 0 & M_{UT} \\ M_{TD} & M_{UT}^T & 0 \end{pmatrix}$$

Problemem jest nie tylko sama wielkość macierzy, które nie mogą zmieścić sie jednocześnie w pamięci, ale również ich zawartość. Wyliczanie danych przy każdej iteracji jest bardzo czasochłonne. Dlatego, żeby nie wyliczać zawartości macierzy przy każdej iteracji, potrzebujemy zapamiętać ich zawartość. Dodatkowo, wyliczanie, choćby jednorazowe, danych jest bardzo kosztowne.

Żeby rozwiązać te problemy, na początku wykonywany jest preprocessing w bazie danych, wyliczający wymagane dane. Następnie, dane te są zapisywane na dysku w łatwą do użycia później strukturę danych, która zawiera tylko określoną ilość wierszy macierzy.

Pliki te później są używane jako fragmenty macierzy. Na tych fragmentach wykonywane są wymagane obliczenia. Kolejne wyniki obliczeń są następnie łączone i przekazywane do kolejnej iteracji, albo zwracane jako wynik algorytmu.

3.4 Preprocessing w bazie danych

W bazie danych wyliczone zostają informacje potrzebne do późniejszego utworzenia macierzy używanych w algorytmach Adapted PageRank i Social PageRank. Algorytmy te przy każdym przebiegu korzystają z tych samych macierzy, obliczanie ich przy każdej iteracji wymagałoby zbyt dużego nakładu czasu. Dane te są wyliczane w dwóch częściach. Na początku wyliczane są w bazie danych i zapisywane w pomocniczych tabelach. Następnie zapisywane są one do do struktury i serializowane w oddzielnych plikach. Pliki te są używane później do tworzenia macierzy.

Informacje te zostaną zapisane w tabeli USERTAGDOC w polu how_much i w tabelach TAG_USR i TAG_DOC. Wyliczenie tych informacji pozwoli później na szybszy dostęp do nich.

Czas wykonania preprocessingu w bazie danych jest różny. Jak widać w tabeli poniżej (3.1) najwięcej czasu trwało tworzenie tabeli TAG_USR i powstało w niej najwięcej nowych rekordów. Wykonywanie tych obliczeń przy każdej iteracji algorytmu spowodowałoby znacznie zwiększenie czasu działania aplikacji.

Poniżej znajdują się listingi zapytań SQL obliczających pola tabeli USERTAGDOC 3.4 i wypełniające tabele TAG_USR (3.2) i TAG_DOC (3.3).

Listing 3.2: Skrypt dodający dane do tabeli tag_doc

```
insert into tag_doc (doc_id, tag_id, how_much)
select d.new_id, tag.new_id, count(utd.user_id)
from
tag,
usertagdoc_tag as utd_t,
usertagdoc as utd, document as d
where
d.id = utd.doc_id
and utd_t.usertagdoc_id = utd.id
and utd_t.tags_id = tag.id
group by utd.doc_id, tag.id;
```

Listing 3.3: Skrypt dodający dane do tabeli tag_usr

```
insert into tag_usr (tag_id, user_id, how_much)
select utd.user_id, tag.new_id, count(utd.doc_id)
from
tag,
usertagdoc_tag as utd_t,
usertagdoc as utd
where
utd_t.usertagdoc_id = utd.id and
utd_t.tags_id = tag.id
```

Listing 3.4: Skrypt updatujący pole how_much w tabeli usertagdoc

```
update usertagdoc utd
set how_much = (select count(distinct tags.tags_id)
from usertagdoc_tag tags
where utd.id = tags.usertagdoc_id);
```

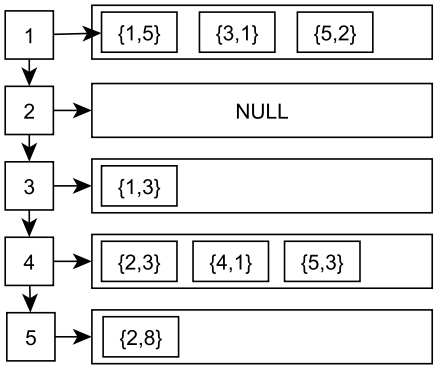
Tabela	czas wykonania polecenia	ilość powstałych rekordów
TAG_USR	FIXME	FIXME
TAG_DOC	ok. 3h	FIXME
USERTAGDOC	FIXME	FIXME

Tablica 3.1: Czas wykonania wypełnienia odpowiednich tabel w bazie danych !FIXME!

3.4.1 Preprocessing: zapis do plików

Po wyliczeniu w bazie danych pomocniczych tabel, wyniki są pobierane przez program, a następnie zapisywane do pliku. Ponieważ pamięć maszyny ogranicza wielkość macierzy na której jesteśmy w stanie operować, tworzone pliki zawierają tylko wyznaczoną ilość wierszy. Ilość wierszy macierzy zapisanych w pliku jest konfigurowalna i zależna od przydzielonej pamięci aplikacji.

W czasie działania algorytmów wymagane są również traspozycje wybranych macierzy. Ponieważ ograniczenia pamięciowe uniemożliwiają obrócenie macierzy w pamięci, w plikach zapisane zostaną również traspozycje macierzy.



Rysunek 3.2: Struktura zapisana w pliku

Wynikiem działania tej części preprocesingu jest struktura będąca tablicą z hashowaniem. Zawiera ona w komórce i listę wszystkich niezerowych elementów znajdujących się w i-tym wierszu macierzy wraz z ich wartościami. Figura 3.2 pokazuje fragment macierzy, który zostanie zapisany do pliku. Poniżej znajduje sie macierz $M_{5,5}$, która powstanie ze struktury przedstawionej na 3.2

$$M_{5,5} = \begin{pmatrix} 5 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 1 & 3 \\ 0 & 8 & 0 & 0 & 0 \end{pmatrix}$$

3.4.2 Inne metody przyśpieszenia wykonywanych preprocesingów

Jednym z głównym problemów jest to, że w czasie wykonywania wymienionych powyżej procedur, tabele USER, TAG i DOCUMENT są

zablokowane na zmiany. W działającej aplikacji byłoby to poważnym problemem.

Można poradzić sobie z tym problemem poprzez posiadanie kopi bazy danych i uzupełnianie obydwu jednocześnie. Dzięki czemu operacje wymagające dużej ilości czasu, mogłyby być wykonane na innej bazie. Po dokonaniu wszystkich obliczeń wymagane jest zsynchronizowanie obydwu baz danych.

Inną możliwością jest użycie obliczen równoległywch, na przykład: podzielenie obliczenia tabel TAG_DOC, TAG_USER i USERTAGDOC na różne maszyny. Dodatkową możliwością jest podzielenie samych tabel na różne procesy. Na przykład wyliczanie tabeli TAG_DOC można podzielić na niezależne procesy ze względu na pole ID obiektów z tabeli TAG. Analogicznie można przeprowadzić taki podział dla innych tabel.

Kolejną możliwością są zmiany danych w tabelach TAG_DOC, TAG_USER i USERTAGDOC w czasie kiedy dodawane są nowe rekordy do tabel USER, DOCUMENT i TAG. Wyeliminowało by to potrzebę wykonywania opisanego powyżej preprocesingu, ale stanowiło by problem przy tworzeniu macierzy. Tworzenie macierzy (dokładnie: plików z których następnie tworzona jest macierz) jest czasochłonne i wymaga zatrzymania zmian dokonywanych na tabelach TAG_DOC, TAG_USER i USERTAGDOC. Problem ten dałoby się rozwiązać przez np: posiadanie kopii wymienionych wcześniej tabel. Synchronizacja kopii tabeli i głównej tabeli nie zajmowała by aż tak wiele czasu. Pomysł ten spowodował by jeszcze jeden potencjalnie groźny problem. Ilość operacji, które trzeba by wykonać w czasie gdy dodawane są nowe rekordy do tabel, znacznie by się zwiększyła. Mogłoby to spowodować większą awaryjność np: problemy z transakcjami.

3.5 Tworzenie macierzy na potrzeby algorytmów

W algorytmach Social PageRank i Adapted PageRank potrzebne są 3 rodzaje macierzy i ich transpozycje. Macierze te są tworzone z danych zebranych z serwisu delicous:

- M_{UT}: macierz ta zawiera w komórce $m_{n,m}$ informacje o ilości dokumentów dodanych przez użytkownika u_n i opisanych tagiem t_m. Dane, z których zostaje utworzona ta macierz znajdują się w tabeli TAG_USR. Tabela ta została wyliczona w czasie preprocessingu.

- M_{TD}: macierz ta zawiera w komórce $m_{n,m}$ informacje o ilości użytkowników którzy opisali tagiem t_n dokument d_m. Źródłem danych dla tej macierzy jest tabela TAG_DOC.

- M_{DU}: analogicznie, ta macierz zawiera dane na temat ilości tagów. Informacje pobierane są z tabeli USERTAGDOC.

Wykorzystywane są one w kolejnych iteracjach algorytmu Social PageRank. Przy algorytmie Adapted PageRank są one używane pośrednio. Struktura na której operuje algorytm Adapted pagerank jest macierzą złożoną z macierzy M_{UD}, M_{TD}, M_{UT} i ich transpozycji. Macierz używana w algorytmie wygląda następująco:

$$G_f = \begin{pmatrix} 0 & M_{DU} & M_{TD}^T \\ M_{DU}^T & 0 & M_{UT} \\ M_{TD} & M_{UT}^T & 0 \end{pmatrix}$$

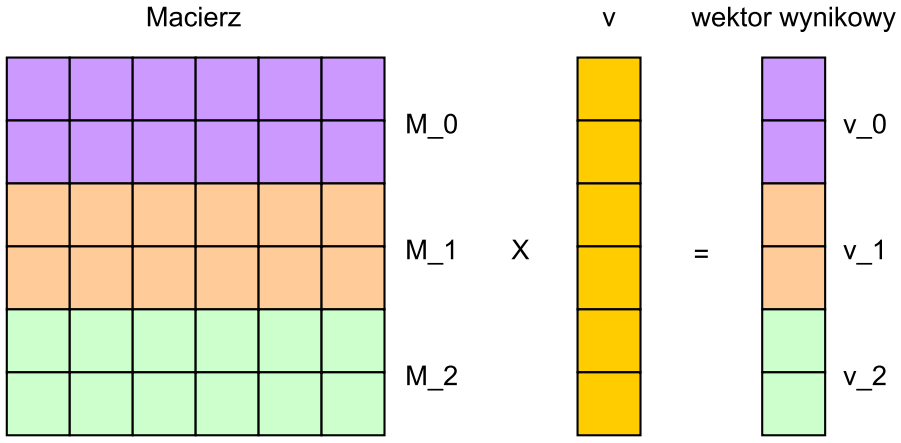
Ważną cechą macierzy na których przeprowadzane są operacje jest to, że są to macierze rzadkie. Liczba niezerowych komórek w macierzy wynosi: [TODO, dane się zmieniły, uzupełnić po testach]

3.6 Operacje przeprowadzane na macierzach

W każdej iteracji algorytmów, główną operacją przeprowadzaną jest mnożenie wymienionych wcześniej macierzy przez wektor. Operacja ta jest przeprowadzana do czasu uzyskania zbieżności wartości wektora wynikowego.

Z powodu wielkości macierzy w aplikacji nie możemy wczytać bezpośrednio całych macierzy do pamięci i na nich operować. Dodatkowo używana biblioteka stawia ograniczenie na iloczyn kolumn i wierszy takie że: $ilosc\_kolumn * ilosc\_wierszy <= 2^{31} - 1$. Gdzie wartość $2^{31} - 1$ jest to maksymalna liczba jaką można przypisać zmiennej typu int w języku Java.

Implementacja



Rysunek 3.3: Mnożenie macierzy przez wektor v

Mnożenie macierzy odbywa się częściami. Bierzemy fragment macierzy: M_i i mnożymy go przez wektor v. Wynikiem jest wektor p_i, który stanowi i-ty fragment wynikowego wektora. Powstałe fragmenty wektora łączymy razem w odpowiedniej kolejności otrzymując w ten sposób w wektor wynikowy (3.3)

Listing 3.5: Mnożenie macierzy przez wektor v

```
def matrix_multiply(v, matrix_source):
        v_return = empty_vector()  #wektor zwracany
        for matrix_part in matrix_source.get_part_matrixes():
                #mnozenie macierzy i wektora
                v_part = multiply(matrix_part, v)
                # dokladamy wyliczony fragment wektora na koniec
                v_return.append(v_part)
        return v_return
```

Mnożenie odbywa się poprzez funkcje z biblioteki Colt. Biblioteka ta uwzględnia to, że macierz na której odbywają się operacje jest macierzą rzadką. Oszczędzana jest pamięć przez zapisywanie tylko niezerowych elementów. W czasie mnożenia przez wektor pomijane są wszystkie zerowe komórki, przez co sama operacja mnożenia jest krótka. Najwięcej czasu zajmuje samo tworzenie częściowych macierzy i ładowanie plików zawierających kolejne fragmenty macierzy.

3.6.1 Inne metody przyśpieszenia obliczeń na macierzach

Opisana powyżej metoda nie mogłaby zostać wykorzystana w działającej. Przy tylko 1 milionie dokumentów czas wykonania algorytmu Social PageRank wynosi około 36 godzin. Prawdziwy system zawierałby zdecydowanie więcej danych. Poniżej opisanych jest kilka pomysłów na ulepszenie i przyśpieszenie działania aplikacji

Zmiana języka w którym została zaimplementowana aplikacja

Maszyna wirtualna Javy nie jest bardzo wydajna jeśli chodzi o szybkość obliczeń, dodatkowo dochodzą ograniczenia związane np: z pamięcią. Możliwe jest również przepisanie tylko kluczowych dla obliczeń fragmentów, a następnie ich uruchamianie z aplikacji.

Proste zrównoleglenie obliczeń

Wykorzystując opisaną wcześniej metodę dzielenia macierzy na kawałki, możliwe jest podzielenie macierzy na różne procesory/maszyny. Każdy proces po zakończeniu obliczania swojej części zapisałby wynikowy wektor do np: bazy danych, w której następowałaby synchronizacja procesów.

Obliczanie przy użyciu GPU/CUDA

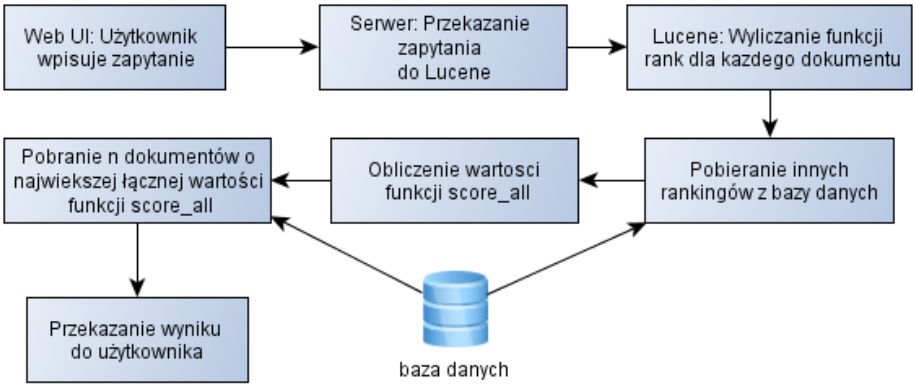
Możliwe jest wykorzystanie GPU do wykonania obliczeń na macierzach. Obecne jednostki graficzne zawierają dodatkowe technologie wspomagające operacje na macierzach rzadkich.

Wykorzystanie innych aplikacji

Obliczanie kolejnych iteracji mogłoby również zrealizowane za pomocą zewnętrznych aplikacji np: MATLAB.

3.7 Wyliczanie rankingu dokumentów

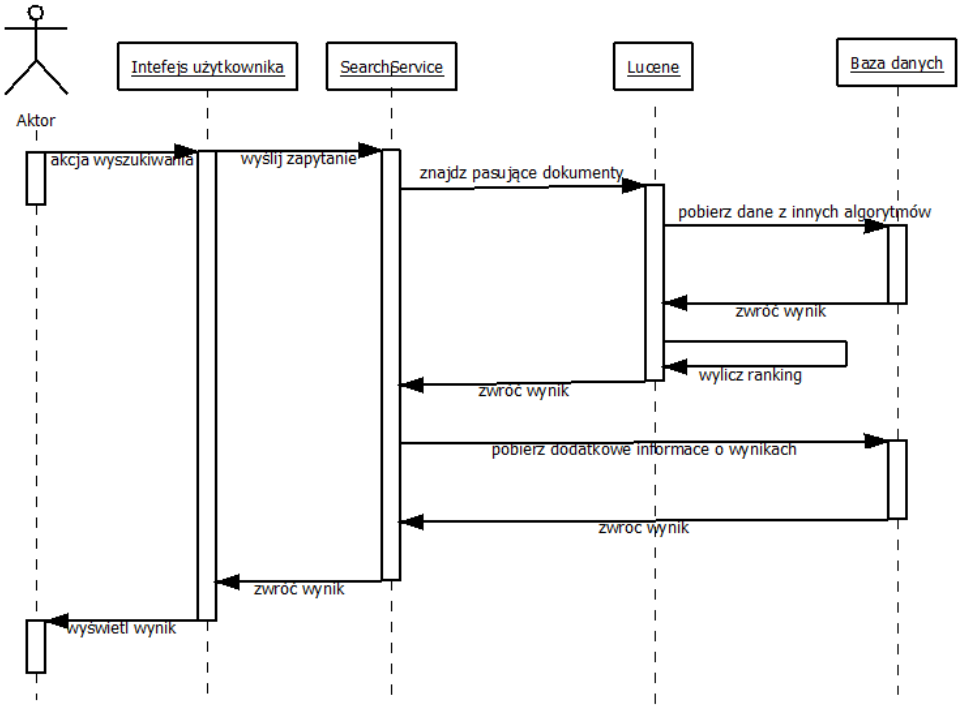
Wyszukiwanie dokumentów jest akcją, która zostaje zapoczątkowana przez użytkownika, poprzez wpisanie odpowiedniego zapytania. Zapytanie to jest w postaci słowa przesyłane jest na serwer, razem z parametrami (wagami) wpływającymi na końcowy ranking dokumentu.



Rysunek 3.4: Proces wyszukiwania w postaci schematu

Pierwszym krokiem otrzymania wyszukiwanych dokumentów jest przekazanie zapytania do frameworku Lucene. W tym systemie zapytanie jest czyszczone i zamieniane na odpowiednie tokeny. Następnie tworzony jest zbiór dokumentów zawierających poszukiwane słowa.

Każdy dokument z tego zbioru ma przydzieloną wartość rankingu Lucene, wyznaczoną przy użyciu funkcji TF-IDF. Jest to pierwszy z czynników wpływających na ostateczny wynik klasyfikacji dokumentu. Dla każdego otrzymanego dokumentu, z bazy danych pobierane są informacje o ich rankingach uzyskanych za pomocą algorytmów AdaptedPageRank, SocialPageRank i danych otrzymanych z innych serwisów społecznościowych (Twitter, Facebook, Digg).



Rysunek 3.5: Diagram przebiegu wyszukiwania

Każdy z cząstkowych rankingów wpływających na ostateczny wynik ma przydzieloną wagę. Waga ta może zostać zmieniona z poziomu interfejsu użytkownika. Ranking dokumentu wyliczany jest według wzoru 2.2.

Rozdział 4

Aplikacja

4.1 Interfejs użytkownika

Interfejs użytkownika został zaprojektowany przy pomocy frameworku Google Web Toolkit (GWT). Jest on narzędziem do tworzenia aplikacji AJAX w oparciu o język Java. Pozwala on na generowanie kodu bez konieczności ręcznego programowania i łączenia elementów języka Java oraz JavaScript. Po napisaniu kodu aplikacji następuje kompilacja części klienckiej do JavaScriptu, CSS oraz HTML. Część serwerowa zostaje skompilowana przez standardowy kompilator Javy. Framework w standardzie zapewnia poprawne wyświetlanie komponentów w przeglądarkach: Firefox, Opera, IE, Safari, Google Chrome. Oprócz tego daje wsparcie dla internacjonalizacji, zarządzania historią w przeglądarce, wykonywania testów jednostkowych JUnit [12] .

Powstała strona internetowa pozwala na wpisanie wyszukiwanego słowa w pole tekstowe. Po wpisaniu słowa i przyciśnięciu klawisza ENTER, nastąpi przesłanie zapytania na serwer. W przypadku słowa występującego w wielu dokumentach wyszukiwanie może potrwać kilka sekund. Po otrzymaniu odpowiedzi, wyniki zostaną wyświetlone na tej stronie, poniżej pola tekstowego.

Interfejs umożliwia również manipulacje wynikami, poprzez zmienienie ich wpływu na końcowy ranking dokumentu. Możliwe jest wybranie wag przydzielonych wynikom otrzymanych z algorytmów SocialPageRank, AdaptedPageRank, frameworku Lucene jak również dla danych otrzymanych z innych serwisów społecznościowych. Wartości wpisane w kolejne pola, są następnie przekazane jako parametry funkcji wyliczającej ranking dokumentu. Ostatnią informacją są poszczególne rankingi otrzymane z różnych źródeł i ostateczny ranking strony.

Uzyskane wyniki zaprezentowane są w przejrzysty sposób. Z lewej strony znajduje się adres adresy wyszukanej strony. Obok adresy, znajduje się wynik otrzymany z frameworku Lucene. Jest to fragment tekstu, w którym

zaznaczona została wyszukiwana fraza. W wyniki zaprezentowane są również popularne tagi oraz ilość użytkowników, którzy daną stronę tymi tagami opisali.

[TODO] IMG INTERFAJSU UZYTKOWNIKA!!![TODO]

Rozdział 5

Dane testowe

5.1 Pobieranie danych

Dane zostały pobrane przy pomocy utworzonych crawlerów. Crawlery te zostały stworzone przy użyciu języka Java i zaimplementowane jako wątki. W aplikacji działają trzy crawlery odpowiedzialne za pobieranie danych z serwisu delicious. Wątki te są odpowiedzalne za jedno z poniższych zadań:

- zbieranie najnowszych danych ze strony delicious.com,
- odświeżanie informacji na temat użytkowników z bazy danych,
- odświerzanie informacji na temat dokumentów zapisanych w bazie danych.

Wszystkie te informacje zostały pobrane z trzech kanałów RSS udostępniany przez stronę delicious.com. Każdy typ crawlera odpowiada za inny kanał RSS.

Głównym źródłem nowych danych jest główny kanał RSS strony delicious. Z tego kanału RSS pobierane zostały informacje o ostatnio dodanych dokumentach. Dodatkowo udostępniane zostały informacje o tagach, którymi zostały opisane dokumenty i użytkownikach, którzy je dodali. Za pozyskiwanie tych informacji odpowiedzialny jest pierwszy typ crawlera.

Wszystkie te dane zostają zapisane w bazie danych. Z informacji zapisanych w bazie: nazwie użytkownika i url dokumentu jesteśmy w stanie uzyskać dostęp do kanału rss użytkowników i dokumentów.

W serwisie delicious dla każdego użytkownika i dokumentu tworzony jest jego własny kanał RSS. W tym kanale, w przypadku użytkowników, uzyskano informacje na temat dodanych przez niego dokumentach. Kanał RSS dokumentu posiada informacje na temat użytkowników, którzy dodali ostatnio dany dokument. Z każdego kanału można było uzyskać maksymalnie 100 ostatnio dodanych zasobów.

Za kanały użytkownika i dokumentu odpowiadają dwa ostatnie typy crawlerów. Wątki te przeglądają bazę danych i dla każdego użytkownika i dla każdego dokumentu sprawdzane są ich kanały. W przypadku kanału dokumentów otrzymamy najczęściej nowych użytkowników i tagi którymi opisali dany dokumentów. Analogicznie z kanału użytkownika otrzymujemy nowe dokumenty. Wszystkie te otrzymane informację są zapisywane w bazie danych.

Wszystkie trzy typy crawlerów działają jednocześnie. Każdy z nich działa w kilku instancjach. Pozwala to na szybsze sprawdzanie i przeglądanie informacji. Crawlery odpowiedzialne za pobieranie nowych informacji przeglądają co pewien odstęp czasu główny kanał RSS systemu delicious zapisując informacje do bazy danych. Każdy z crawlerów odpowiedzialnych za odświeżanie informacji w bazie danych, przegląda przydzielony mu fragment bazy, sprawdza odpowiednie kanały RSS i dodaje uzyskane dzięki temu informacje do bazy.

Dane były przeglądane w trakcie zapisywania i po zakończeniu procesu zbierania. W trakcie pobierania danych, sprawdzane były dodawane tagi. W trakcie tego procesu usunięte zostały wszystkie znaki specjalne.

Następnie po zebraniu wystarczającej ilości danych, sprawdzone zostały statystyki wystąpień dokumentów, tagów i aktywność użytkowników. Dodatkowo sprawdzana została jeszcze raz jakość tagów, pominięte znaki specjalne i ich długość.

5.1.1 Czyszczenie tagów przed zapisem

Tagi pobierane z serwisu delicious nie zawsze są w postaci wymaganej przez aplikację. Spowodowane to jest błędami użytkowników, czy też specyficznym stylem zapisywania tagów.

Niektóre tagi mają różne znaczenie w zależności od kontekstu, na przykład tag 'design' ma inne znaczenie w kontekście strony o programowaniu, a inne w kontekście strony o sztuce. Część użytkowników żeby poradzić sobie z tym problemem dodają do tagów informacje mówiące o ich domenie. Często domena ma wygląd 'programming@design' czy 'art#design'.

Z powodu tego specyficznego zapisu każda etykieta, przed zapisaniem do bazy danych jest dzielona na dwa lub więcej słów w miejscach występowania popularnych znaków specjalnych.

Dodane kontekstu do tagu mogłoby być przydatne w aplikacji, ale z powodu tego że każdy użytkownik ma swój specyficzny sposób opisywania dokumentów np: design@art i art#design, trudno jest je zunifikować. Dodatkowym problemem jest to, że nie jest to sposób opisu używany przez wszystkich użytkowników.

Adnotacje przypisywane przez użytkowników często kończą się lub zaczynają od znaków specjalnych. Jest to spowodowane np: błędami

(dodatkowe przecinki) albo specyficznym stylem opisywania danych przez użytkownika. Wszystkie znaki specjalne z końca i początku dokumentu są usuwane przed dodaniem do bazy danych.

Przykłady danych przed i po ich oczyszczeniu:

- '@java' : 'java'
- '@@java' : 'java'
- '#java6@' : 'java6'
- 'design!$%@art' : 'design', 'art'
- 'art!#,': 'art'

Przykładowe dokumenty wraz z użytkownikami i tagami którymi zostały opisane można zobaczyć w tabeli 5.1.

5.1.2 Czyszczenie bazy danych przed testami

Po zebraniu dużej ilości danych dane zostały jeszcze dodatkowo przejrzane. Posiadany zbiór danych testowych składał się z 1.000.000 dokumentów, 600.000 użytkowników i 300.000 tagów. Dane te zawierały dużo dokumentów nieopisanych tagami. Często dokumenty zapisane w bazie danych dodane były tylko parę razy. Podobnie z użytkownikami. Dane te powiększały bazę danych, zwiększając zdecydowanie czas obliczeń. Usunięcie tych dokumentów spowodować powinno również zwiększenie jakości wyników używanych algorytmów.

W pierwszym kroku przejrzane zostały jeszcze raz tagi. Usunięte zostały wszystkie zawierające znaki specjalne pominięte w czasie dodawania dokumentów do bazy danych. Następnym krokiem było sprawdzenie długości etykiet. Wszystkie o długości mniejszej niż 3 i większej od 15 zostały usunięte. Ostatnim krokiem przy przeglądaniu tagów, było usunięcie wszystkich używanych mniej niż 10 razy.

Z powodu usunięcia dość dużej ilości tagów, powstała duża grupa użytkowników i dokumentów nie będąca w żadnej relacji. Wszystkie takie obiekty zostały usunięte.

Dane użytkowników i dokumentów przeglądane były tylko pod względem wystąpień w bazie danych. Przy użytkownikach sprawdzana była ilość dodanych przez nich dokumentów. W przypadku dokumentów, analogicznie sprawdzana była liczba użytkowników, którzy dany dokument dodali. Usunięci zostali wszyscy użytkownicy, którzy dodali mniej niż 10 dokumentów. Z dokumentów usunięte zostały wszystkie o ilości dodań mniejszej niż 5.

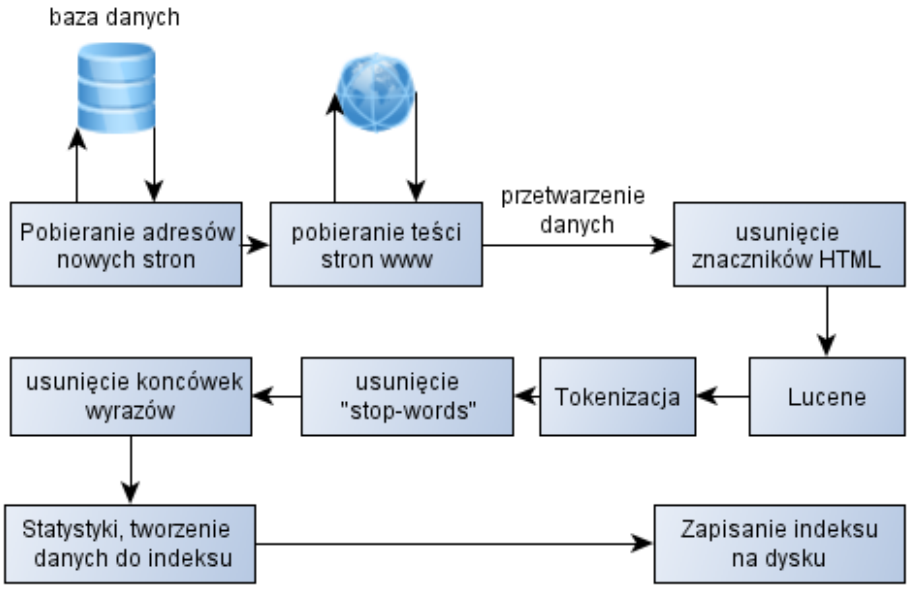
Ostatnim krokiem było przejrzenie jeszcze raz tabeli zawierającej tagi. Usunięte zostały wszystkie, które po czyszczeniu tabel użytkowników i dokumentów pozostały bez żadnej relacji.

W taki sposób otrzymana została baza, zawierająca około 300.000 dokumentów, 200.000 użytkowników i 30.000 tagów.

5.1.3 Pobieranie stron www

Framework Lucene wymaga dla wyszukiwania pobranych wcześniej stron www. Treści tych stron zostały pobrane po zebraniu wszystkich danych w bazie danych i ich oczyszczeniu. Za pobieranie odpowiedzialny był osobny program.

Program składał się z kilku wątków przeglądających przydzielony im tabeli DOCUMENT. Informacje o adresie strony zapisane są w polu URL. Dla każdego zapisanego dokumentu pobierana jest treść strony na która wskazuje. Strona www jest później czyszczona ze znaczników HTML i przekazana do frameworku lucene do zindeksowania. Jeśli wszystkie czynności zakończą się powodzeniem, w bazie danych zostaje odnotowana informacja o posiadaniu na dysku danego dokumentu.



Rysunek 5.1: Lucene: pobieranie danych i indeksowanie

We frameworku lucene zapisywana jest nie tylko treść strony www, ale również identyfikator dokumentu w bazie danych. W czasie wyszukiwania otrzymujemy jednocześnie informacje o rezultacie funkcji wyszukującej razem z odnośnikiem do dokumentu w bazie danych.

5.1.4 Dane testowe

W bazie danych znajdują się również dokumenty przeznaczone tylko do testowania skuteczności algorytmów. Dane te zostały pobrane ręcznie z różnych serwisów internetowych. Dzięki odmiennej tematyce tych stron internetowych, możemy założyć że dane te stanowią dwie odrębne grupy testowe:

- PMC (http://www.ncbi.nlm.nih.gov/pmc/): zawierająca bazę artykułów z dziedziny medycyny. Pobrano 300 dokumentów.
- Arxiv: artykułów z dziedziny fizyki, astronomii i pokrewnych. Wybrano 146 artykulów.

Dane z bazy PMC można podzielić jeszcze na kilka poddziałów medycyny. Powstałe grupy to badania na temat chorób tarczycy, wirusy i AIDS. Ostatnie dwie grupy: wirusy i AIDS mogą się pokrywać.

Każdemu dokumentowi zostały przydzielone odpowiednie etykiety. Utworzone zostały jako suma przydzielonych tagów przez autora tekstu i słów wchodzących w skład tytułu dokumentu. Łączna ilość tagów przydzielona tym dokumentom wynosi 2856.

Większość dokumentów pobranych z bazy PMC posiada informacje na temat ilości prac cytujących dany artykuł. Zostały one użyte jako ilość osób, które dodały dany dokument do bazy danych. Na potrzeby testów utworzono 198 użytkowników, których przyporządkowano do konkretnych dokumentów zgodnie z rozkładem normalnym. Dzięki temu wybrani użytkownicy charakteryzowali się zróżnicowaną aktywnością.

Tagi, którymi użytkownik opisał dokument, są również losowane z przypisanego dla danego dokumentu zbioru. Dzięki temu każdy użytkownik, który dodał dokument mógł go opisać różnymi słowami kluczowymi.

Tytuł strony (URL)	Użytkownik: przypisane tagi
BBC News (http://www.bbc.co.uk/news/)	skygreen: daily, media, news, education, english, newspapers,...
	hannibalsfather: news, media, world, online, politics, education, ...
	stphnclysmth: imported, portal, news, media, world, online, ...
Fontspring (http://www.fontspring.com/)	betorj30: css3, design, development, font, fontes, fonts, free, generator, html, online, typography, webdesign, websites, ...
	craigruks: imported, gallery, type, typeface, inspiration, css3, resources, typography, shop, design, desktop, shopping, webdesign
	maitreya11: fonts, typography, webdesign, font, design, type, webfonts, free, css3, resources, shop, webdev, ...
web.py (http://webpy.org/)	cainmark: python, framework, programming, webdev, development, opensource, tutorial, toread, tool, todo, technology, html, archive, apache, code, internet, coding,...
	itsnotvalid: code, application, database, design, development, opensource, tools, webdevelopment, library, ...

Tablica 5.1: Wybrane dokumenty i przypisane im tagi

Rozdział 6

Wyniki

6.1 Proste testy

Dla zaprezentowania działania algorytmów SocialPageRank i Adapted PageRank przeprowadzono obliczenia na prostych danych. Wszystkie obliczenia wykonano używając danych podanych w tabeli 6.1.

	Użytkownicy	
	użytkownik 1	użytkownik 2
http://www.ted.com/	inspiration	
http://www.colourlovers.com/	design	inspiration
http://www.behance.net/	portfolio, design	portfolio, inspiration

Rysunek 6.1: Dane do prostych testów

6.2 Wyniki algorytmu SocialPageRank dla prostych danych

Dla danych z tabeli 6.1 macierz $M_{d,u}$ mówiąca o zależności dokumentów z użytkownikami ma postać:

$$M_{d,u} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Macierz użytkowników i tagów, $M_{u,t}$:

$$M_{u,t} = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 0 & 1 \end{pmatrix}$$

Macierz tagów i dokumentów, $M_{t,d}$:

$$M_{t,d} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}$$

wyniki:

Dla powyższych danych wyniki algorytmu zbiegają po czterech iteracjach z dokładnością $|P_3 - P_4| < 10^{-10}$. Wyniki zostały przedstawione w tabelce 6.1

	Social PageRank
www.ted.com	0.2381373691295440
www.colourlovers.com	0.4343479235414989
www.behance.net	0.8686958470829979

Tablica 6.1: Wyniki działania algorytmu Social PageRank

Można zauważyć, że największy ranking ma strona behance.net. Strona ta została dodana przez dwóch użytkowników. Strona colourlovers.com została dodana przez taką samą liczbę użytkowników, jednak uzyskała ranking o połowę mniejszy. Spowodowane to zostało tym, że behance.net opisana została większą ilością bardziej popularnych tagów: dwa razy tagiem 'portfolio', użytym tylko dla tej strony, raz tagiem 'design' i tagiem 'inspiration', który jest najpopularniejszym tagiem, użytym aż 3 razy.

6.3 Wyniki algorytmu Adapted PageRank dla prostych danych

Algorytm Adapted PageRank przetestowano na tych samych danych co algorytm SocialPageRank (tabelka 6.1). Macierz asocjacyjna powstała z tych danych ma wymiary 8×8 i wygląd:

$$G_f = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 2 & 2 & 1 & 1 & 2 \\ 1 & 1 & 2 & 0 & 0 & 1 & 2 & 1 \\ 0 & 1 & 2 & 0 & 0 & 2 & 0 & 1 \\ 1 & 1 & 1 & 1 & 2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Zbierzność wektora została uzyskana po 22 iteracjach. Jest to zdecydowanie dłuższy czas w porównaniu z czterema wymaganymi iteracjami przy poprzenim algorytmie

	Adapted PageRank
doc: http://www.ted.com/	0.280676409730572
doc: http://www.colourlovers.com/	0.369220441236174
doc: http://www.behance.net/	0.384551423972747
usr: użytkownik A	0.402473669662513
usr: użytkownik B	0.354578057974890
tag: inspiration	0.383291185401107
tag: design	0.321455285546253
tag: portfolio	0.314739469615726

Tablica 6.2: Wyniki działania algorytmu Adapted PageRank

Analizując wyniki z tabeli 6.1 można zauważyć, że kolejność stron według obydwu algorytmów jest taka sama. Najwyższy ranking wśród dokumentów ma również strona behence.net, a najniższy strona ted.com.

Porównując wyniki algorytmów Adapted PageRank i SocialPageRank można zauważyć różnice miedzy poszczególnymi wartościami. Dla przykładu, różnica miedzy pierwszym a drugim dokumentem jest bardzo mała w porównaniu z prawie dwa razy większą wartością uzyskaną przy pomocy algorytmu Social PageRank. Ponieważ strony różnią się tym, że zostały opisane różna ilością tagów można po tym wywnioskować ze nadanie większej ilości tagów nie ma dużego wpływu na ranking strony. Za to zmniejszenie liczby użytkowników którzy tą stronę dodali bardzo zmniejsza wynik.

6.4 Wyniki działania algorytmów

W tablicach 6.6 i 6.5 przedstawiono rezultaty działania algorytmów SocialPageRank, AdaptedPageRank. Wyniki te zostały posortowane w zależności od wyniku algorytmu SocialPageRank 6.5 i AdaptedPageRank 6.6. Dodatkowo w tablicy 6.7 znajdują się wyniki posortowane w zależności od sumy popularności dokumentów w sieciach twitter, facebook i digg. Można zauważyć, że żaden z pierwszych 25 dokumentów się nie pokrywa.

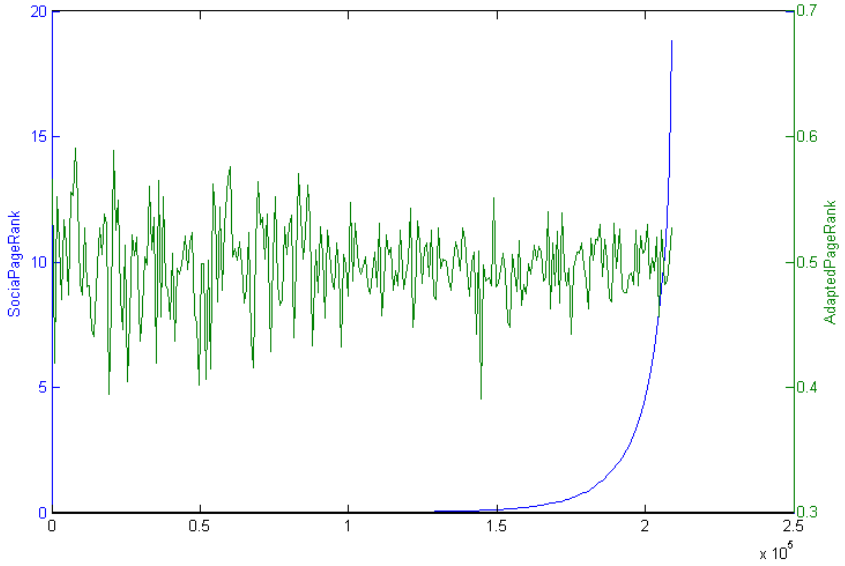
W tablicy 6.3 znajdują się wartości maksymalne, minimalne i średnie dla różnych algorytmów. Można zauważyć, że podobnie jak dla prostych przykładów, algorytm AdaptedPageRank ma zdecydowanie mniejsze różnice miedzy poszczególnymi wynikami.

Dokładniej zelażnością między poszczególnymi wynikami można się przyjrzeć na rysunkach 6.2, 6.3, 6.4, 6.5. Na wykresach tych przedstawiono wyniki algorytmów posortowane po jednej z danych. Dodatkowo na rysunku 6.4 i 6.5 znajdują się wyniki odpowiednio algorytmu AdaptedPageRank i SocialPageRank przedstawione razem z danymi uzyskanymi z innych sieci społecznościowych. Na 6.5 można zaobserwować zależność między wynikami

	maksymalna	średnia	minimalna
SocialPageRank	59.9655565	0.654392541	1.4996654e-012
AdaptedPageRank	0.7196510	0.497870340	0.2520965
inne	578287638	17132.4861 (21544.9729)	0

Tablica 6.3: Maksymalne, średnie, minimalne wartości algorytmów.

algorytmu SocialPageRank a popularnością danych dokumentów w innych sieciach społecznościowych.



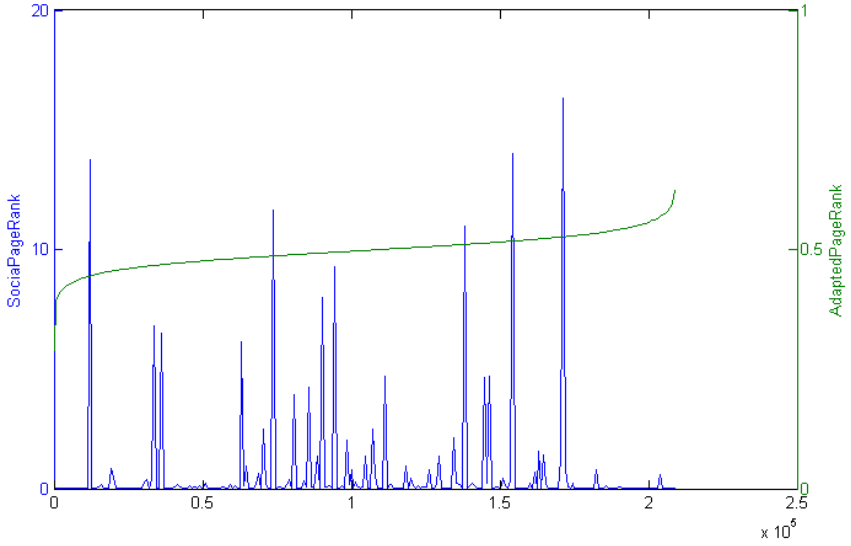
Rysunek 6.2: Zależności między wynikami algorytmów SocialPageRank i AdaptedPageRank. Posortowane po wartościach algorytmu SocialPageRank

6.5 Problemy oceny wyników

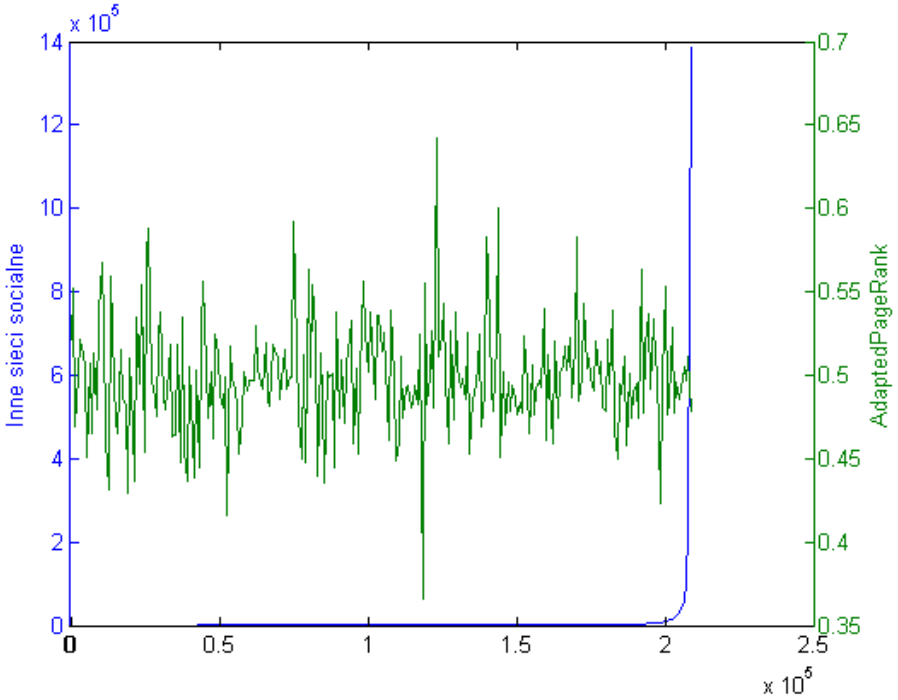
6.6 Testy

6.6.1 Opisy przeprowadzonych eksperymentów

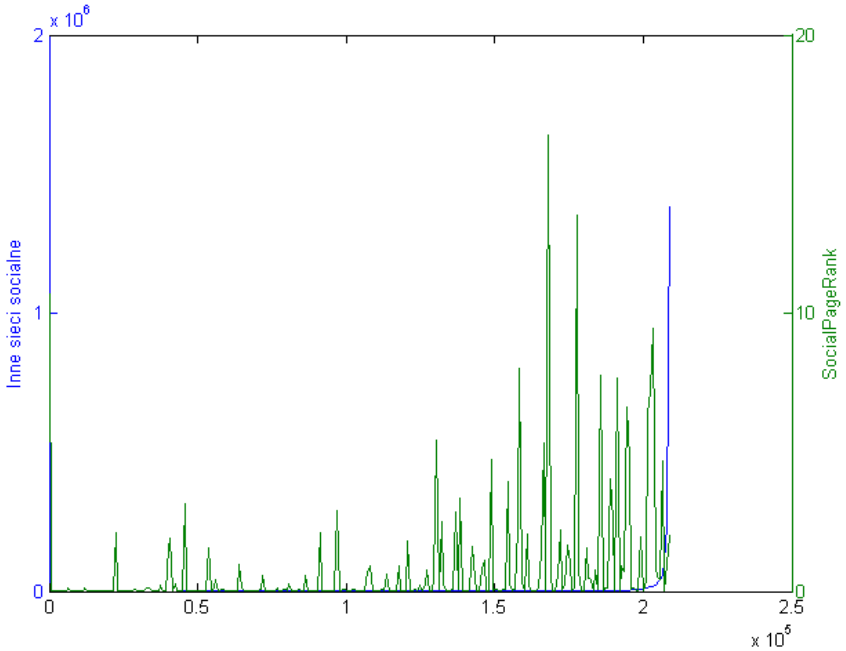
Dane testowe składają się ze zbioru prac naukowych pobranych ze stron Arxiv i PMC. Każdy dokument posiada przypisane mu tagi i ilość cytowań w innych pracach naukowych.



Rysunek 6.3: Zależności między wynikami algorytmów SocialPageRank i AdaptedPageRank. Posortowane po wartościach algorytmu AdaptedPageRank



Rysunek 6.4: Zależności między wynikami algorytmów AdaptedPageRank i danymi z innych sieci socialnych. Posortowane po wartościach z innych sieci



Rysunek 6.5: Zależności między wynikami algorytmów SocialPageRank i danymi z innych sieci socialnych. Posortowane po wartościach z innych sieci

6.6.2 Jakość wyszukiwania

Każdemu dokumentowi została przypisana jego 'jakość'. Jakość dokumentu została określona jako znormalizowana ilości cytowań danego dokumentu w innych pracach naukowych. Należy tutaj zaznaczyć, że przy dodawaniu danych testowych do bazy danych, liczba użytkowników, którzy dodali dany dokument, jest równa ilości cytowań. Może to mieć wpływ na wyniki testów. Jednakże przy obydwu algorytmach, wpływ maja nie tylko użytkownicy, ale również tagi i ich jakość. Dodatkowo, przy tworzeniu macierzy, liczba ta nie jest używana bezpośrednio.

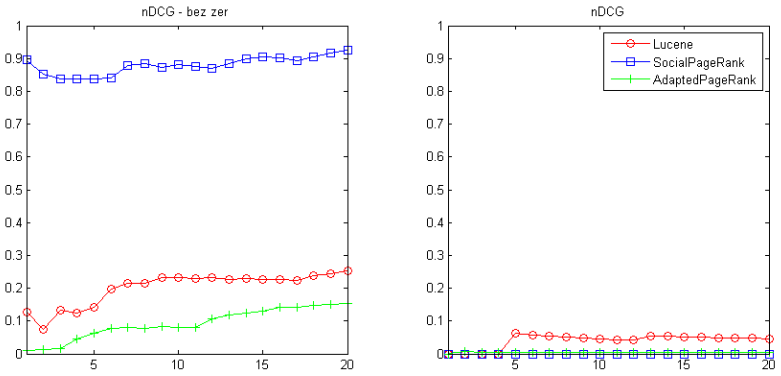
Miarą testów jest znormalizowany zysk całościowy (ang. Normalized Discounted Cumulative Gain). Wyliczany jest on według wzoru:

$$nDCG_p = \frac{DCG_p}{IDCG_p} \tag{6.1}$$

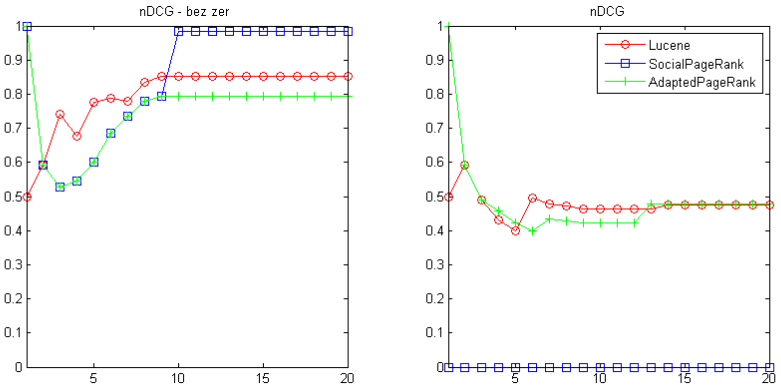
Gdzie DCG to zysk całościowy (ang. Discounted Cumulative Gain), a IDCG to idealny zysk całościowy obliczony jako maksymalny możliwy wynik dla danego zapytania p .

Zysk całościowy wyliczany jest za pomocą wzoru:

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2(i+1)} \tag{6.2}$$



Rysunek 6.6: Wyniki testów pod względem jakości, wyszukiwanie słowa: 'genome'



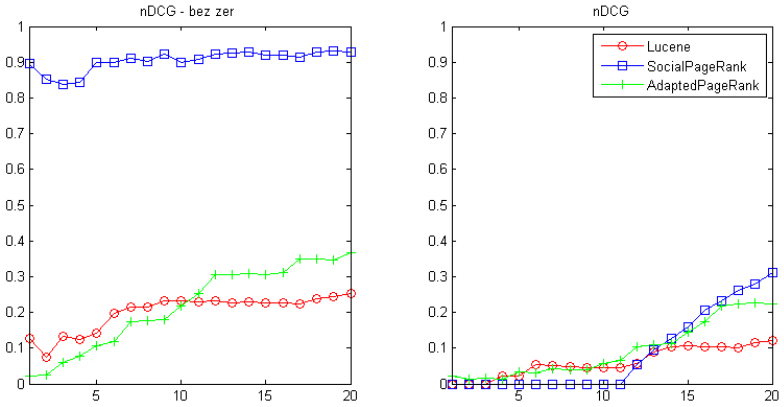
Rysunek 6.7: Wyniki testów pod względem jakości, wyszukiwanie słowa: 'hashimoto'

Gdzie rel_i jest jakością dokumentu otrzymanego w wyniku na pozycji i.

Testy zostały przeprowadzone na zbiorze wszystkich zebranych dokumentów. Do wyników brane były tylko dokumenty ze zbioru prac naukowych. W przypadku uzyskania w wyniku dokumentu z poza tego zbioru, jakość tego wyniku równa jest zero. Dla porównania przedstawiono również wyniki testów, w których pominięto wyniki z poza zbioru.

Można zauważyć na wykresach, że otrzymujemy lepsze wyniki przy użyciu algorytmu SocialPageRank w zbiorze danych z których usunięto elementy zerowe. Dokumenty otrzymane przy pomocy innych metod są gorszej jakości.

W zbiorze w którym występowały dokumenty o jakości zero, Lucene i AdaptedPageRank dawały zdecydowanie lepsze wyniki. Mimo, że koszt całościowy jest niski, to w przypadku algorytmu SocialPageRank jest



Rysunek 6.8: Wyniki testów pod względem jakości, wyszukiwanie słowa: 'immunodeficiency'

on w większości zerowy. Nawet jeśli jakość tych wyników jest niska, AdaptedPageRank i Lucene dają poprawne wyniki w pierwszych 10 elementach.

Zaobserwować również wysokie wyniki otrzymane przy pomocy Lucene. W tym przypadku, może to byś spowodowane wysoką jakością publikacji naukowych w porównaniu do treści stron internetowych.

6.6.3 Precyzja wyszukiwania

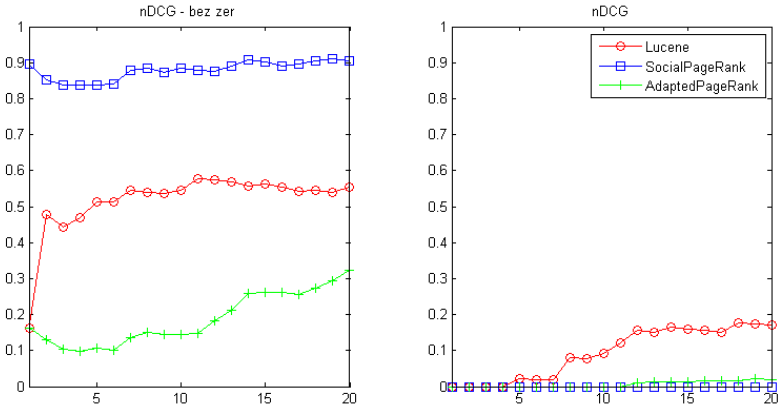
Kolejną miarą wyników algorytmów jest precyzja. Miara ta będzie sprawdzana dla pierwszych n wyników:

$$Prec(n) = \frac{number\_rel_n}{n} \tag{6.3}$$

Precyzja dla n jest to ilość poprawnie wybranych elementów ($number\_rel$) dla pierwszych n elementów.

Dla posiadanych danych, precyzja odpowiedzi została przetestowana dla pierwszych 20 wyników. Przykładowe wyniki znajdują się na wykresach 6.11, 6.12, 6.13, 6.14 i 6.15.

Można zaobserwować, że poza wykresem 6.15 i 6.12 (od elementu 15) najlepsze wyniki uzyskujemy używając algorytmu Adapted PageRank. Przy użyciu Lucene otrzymujemy gorsze wyniki, ale ciągle dość wysokie. Algorytm SocialPageRank w pierwszych 20 wynikach podaje bardzo złe wyniki, często zerowe.



Rysunek 6.9: Wyniki testów pod względem jakości, wyszukiwanie słowa: 'pathogen'

6.6.4 Średni wzajemny ranking

Przeprowadzone zostały również testy biorące pod uwagę cały zbiór dokumentów, czyli dokumenty zebrane z serwisu delicious.com i dokumenty złożone z publikacji naukowych. Problemem w testowaniu algorytmów wyszukiwania na takim dużym zbiorze jest sam proces ocenienia otrzymanych odpowiedzi. Dodatkowo należy sprawdzić czy w zbiorze pominiętych dokumentów nie znajdują się wyniki, które mogą nas interesować. Dlatego też do przeprowadzenia tych testów użyto miary 'średniego wzajemnego rankingu', która bierze pod uwagę tylko wybrany zbiór zapytań i jedną szukaną odpowiedz.

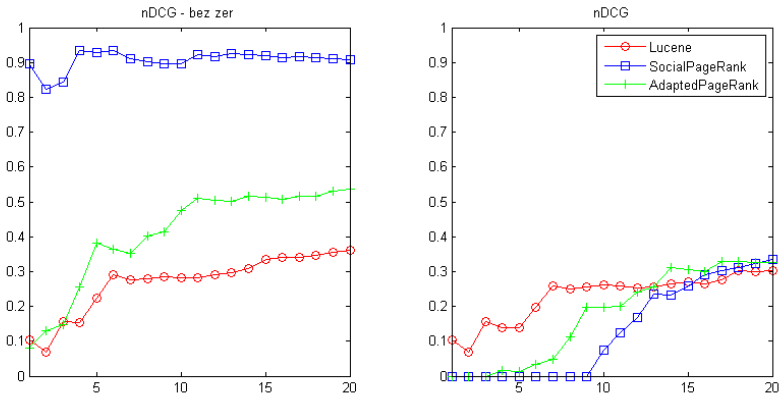
Średni wzajemny ranking (MMR, Mean reciprocal rank) jest miarą pozwalającą na ocenianie jakości wyników zwracanych jako lista elementów posortowanych według trafności. Kryterium to bierze pod uwagę zbiór zapytań i jedna poprawna odpowiedz. Mierzona jest pozycja, na której znajduję się interesująca nas odpowiedz. Dla przykładu, jeśli w zbiorze zapytań znajduję się fraza 'bbc' oczekujemy, że główna strona internetowa serwisu informacyjnego BBC będzie na wysokiej pozycji w rankingu. Im dalej w wynikach znajduje się oczekiwana odpowiedz, tym gorszy będzie wynik MMR.

Średni wzajemny ranking (MMR) wyrażany jest wzorem:

$$\mathrm{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\mathrm{rank}_i} \tag{6.4}$$

Gdzie:

- Q - zbiór zapytań,



Rysunek 6.10: Wyniki testów pod względem jakości, wyszukiwanie słowa: 'retrovirus'

- $rank_i$ - pozycja poprawnego wyniku.

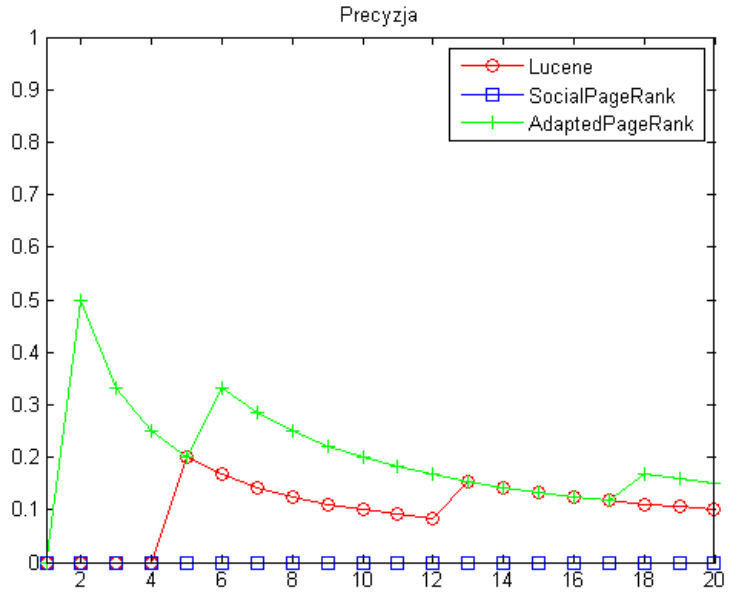
Do testów wybrane zostało wybrane dziesięć stron internetowych i zapytania opisujące je w jednoznaczny sposób. Ponieważ nie zawsze główne strony znajdują się w bazie danych, jako wynik wystarczający uznawana jest również strona będącą w tej samej domenie. Dla przykładu, dla zapytania 'BBC' akceptowane są strony bbc.co.uk i bbc.co.uk/tv jako poprawne odpowiedzi.

W tabeli 6.4 znajduje się zbiór zapytań i numer pozycji w wynikach, na której otrzymaliśmy poszukiwaną stronę. Wynik zero oznacza brak oczekiwanego wyniku w pierwszych dwudziestu odpowiedziach.

Zapytanie	Lucene	SocialPageRank	AdaptedPageRank	Inne serwisy
Debian	7	4	0	0
BBC	2	0	2	5
Intel	1	8	0	0
Deviantart	1	2	0	6
Pastebin	1	0	8	3
Wikipedia	4	1	1	14
Guardian	2	4	1	6
Youtube	0	0	2	11
Sourceforge	3	3	3	2

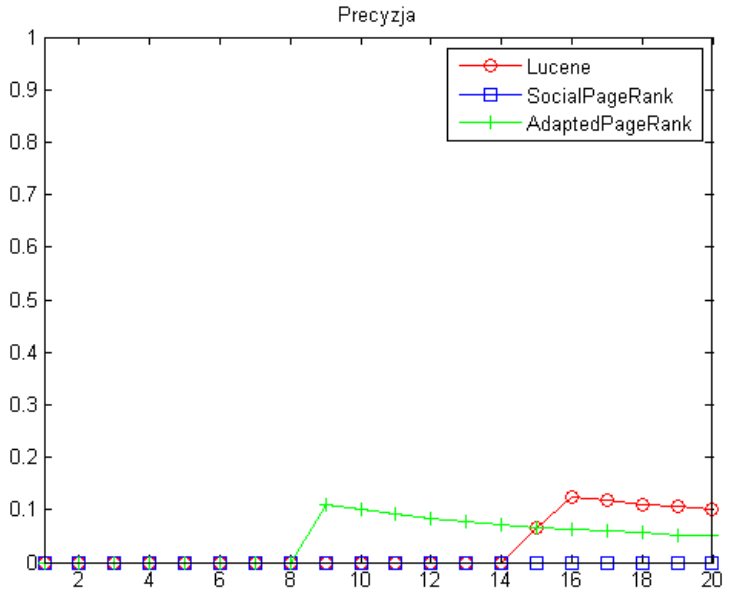
Tablica 6.4: Pozycja odpowiedzi w wynikach w zależności od algorytmu i zapytania

Wyniki zostały również przedstawione w postaci figury 6.16. Tak samo jak w pojedynczych winkach jak uśrednionych można zaobserwować,

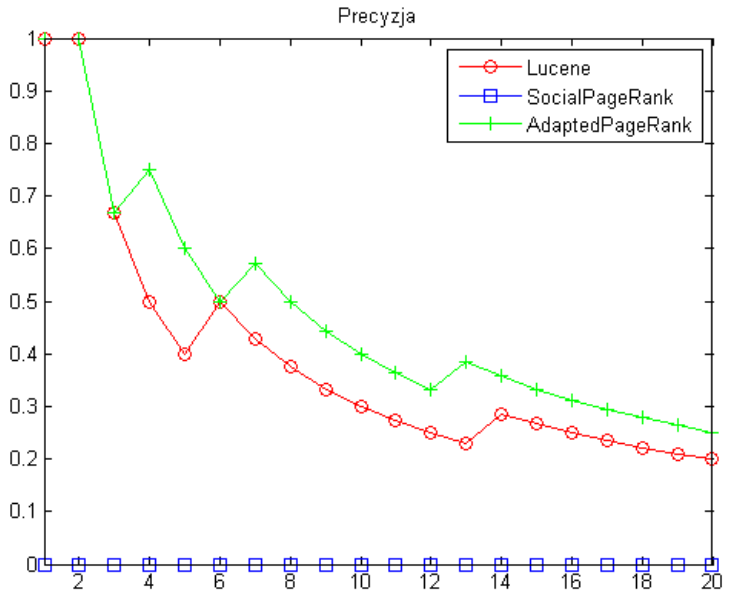


Rysunek 6.11: Wyniki precyzji wyników, wyszukiwanie słowa: 'Genome',

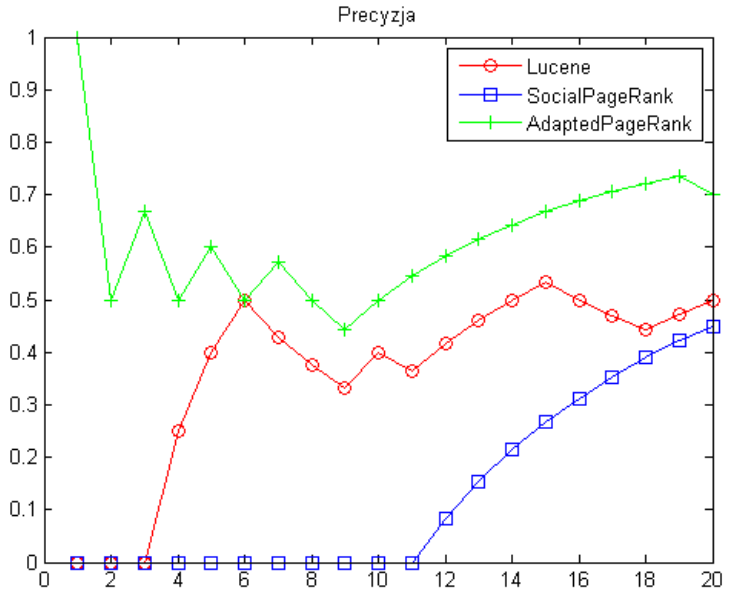
że najlepsze wyniki otrzymujemy przy użyciu frameworku Lucene, czyli algorytmu TF-IDF.



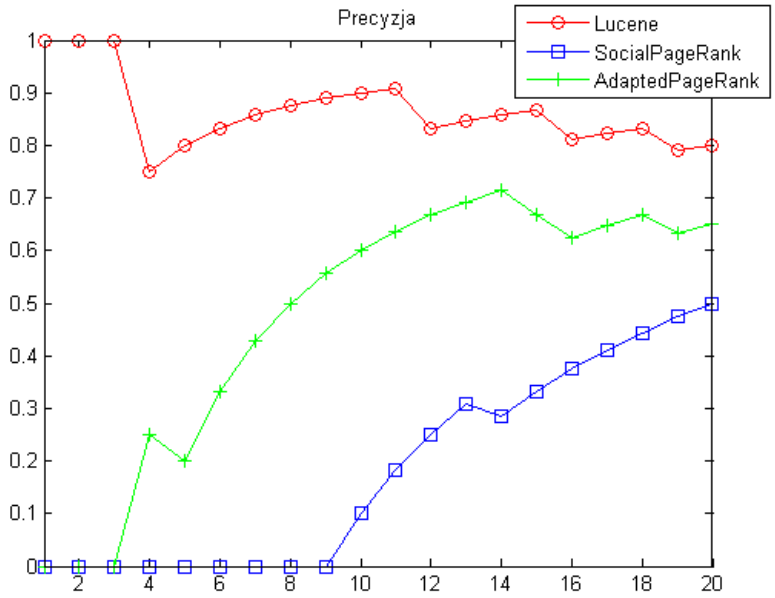
Rysunek 6.12: Wyniki precyzji wyników, wyszukiwanie słowa: 'Geophisics',



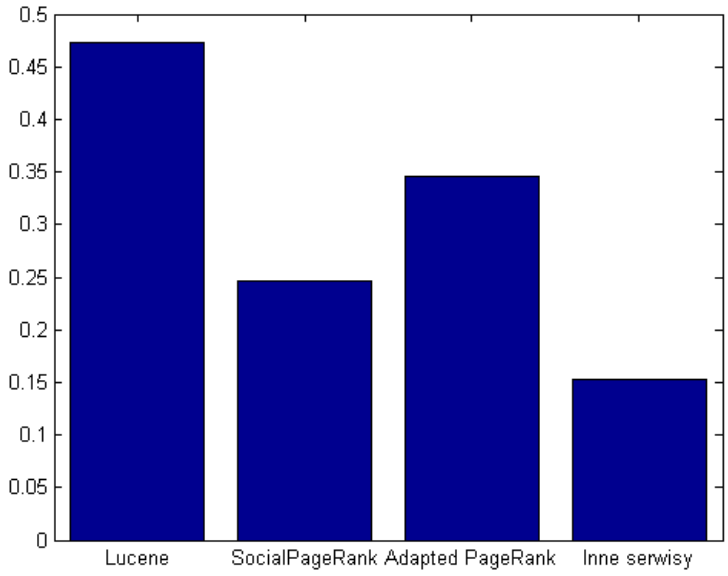
Rysunek 6.13: Wyniki precyzji wyników, wyszukiwanie słowa: 'Hashimoto',



Rysunek 6.14: Wyniki precyzji wyników, wyszukiwanie słowa: 'Immunodeficiency',



Rysunek 6.15: Wyniki precyzji wyników, wyszukiwanie słowa: 'Retrovirus',



Rysunek 6.16: Średni wzajemny ranking

URL	Social PageRank	Adapted PageRank	inne
http://simplepie.org/	59.9655565290432	0.499035904668234	387
http://www.tutorialized.com/	48.5601883893407	0.4534007447034	2263
http://www.winehq.org/	44.4587236045799	0.520931523735242	4423
http://spoon.net/	39.1906058833332	0.493572342717463	3582
http://www.voidtools.com/	38.7604379886134	0.494614126702101	4904
http://sxsw.com/	37.2726961819357	0.487624639504345	27723
http://code.google.com/closure	37.2548272381968	0.483262379806695	332
http://www.smashingmagazine.com/2010/05/06/modern-css-layouts-....	35.4053295474874	0.512348711683268	637
http://psd2cssonline.com/	35.350819038433	0.464286546334638	664
http://superuser.com/	34.6226519522827	0.515859553270239	2706
http://partedmagic.com/	34.1270908460213	0.495873910050547	331
http://noscript.net/	31.4436112768568	0.53215157984271	5926
http://windirstat.info/	30.8137477170295	0.519347815131998	2133
http://sixrevisions.com/resources/websites_for_web_development/	30.3211379131049	0.512499653235814	3061
http://www.rawtherapee.com/	30.1353246293996	0.539472001581894	878
http://www.twospy.com/galleriffic/	29.4617221389003	0.458659482118933	383
http://www.smashingmagazine.com/2009/07/15/clever-png-optimization-...	29.1316173184832	0.493594194689542	292
http://www.radicalcartography.net/	29.063512610649	0.507540786838185	3975
http://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro	28.7877450055635	0.463743529594843	658
http://www.openfiler.com/	28.7461193203883	0.50020384218663	340
http://webpy.org/	28.6142226104057	0.532521857305088	323
http://quixapp.com/	28.3061763646058	0.507577478140865	628
http://iconfactory.com/freeware	28.1447238812035	0.511970489720602	468
http://awesome-fontstacks.com/	27.8307412330065	0.485250048705417	3004
http://www.google.com/talk/service/badge/New	27.7486823470783	0.533924549035468	120
http://www.microsoft.com/windows/virtual-pc/download.aspx	27.627771443592	0.531548185540212	4648
http://freelanceswitch.com/productivity/the-monster-collection-of-...	27.499217396276	0.479470014272181	704

Tablica 6.5: Wyniki posortowane ze względu na algorytm Social PageRank

URL	Social PageRank	Adapted PageRank	inne
http://www.youtube.com/watch?v=CS3ErDN50Qk	0.00017529173902660 3	0.719651043519777	61741
http://www.bnet.com/	4.62506990825593e-008	0.718636745108625	1316
http://www.youtube.com/watch?v=–9OsfTfb6E	0.00017529173902660 3	0.715614078173755	9923
http://www.youtube.com/watch?v=QSZ-0Mr30mU	0.00017529173902660 3	0.712967347913995	52468
http://www.youtube.com/watch?v=v0mb0_SUx-A	0.00035058347805320 6	0.710175094948763	68377
http://www.guardian.co.uk/world/2011/nov/13/first-world-war-brit...	4.95146109907588e-010	0.709787547524645	471
http://www.tnr.com/print/article/world/magazine/95915/hemingwa...	2.92862659591136e-008	0.700991622853237	200
http://www.bbc.co.uk/news/technology-15239109	2.92862659591136e-008	0.699693745492752	748
http://www.theglobeandmail.com/news/arts/books/truth-lies-and...	4.95146109907588e-010	0.698880767885222	79
http://www.massmediajobs.com/component/jobs/detail_job/30473...	2.92862659591136e-008	0.695874348100943	0
http://www.thesun.co.uk/sol/homepage/hold_ye_front_page/	4.95146109907588e-010	0.692231199372216	755
http://archiveofourown.org/works/78631	1.66340928821329e-005	0.689718168156393	0
http://employerdrivenhealthplans.com/?post_type=page&p=670	1.29520003384321e-006	0.687329875547237	0
http://darmano.typepad.com/logic_emotion/2011/10/influence.html	2.92862659591136e-008	0.686017872336727	258
http://advocateforcats.com/wp-content/uploads/2011/05/Advocat...	2.54058468176938e-006	0.684812719480609	0
http://blackboardbattlefield.com/2011/10/28/follow-friday-my-top...	4.95146109907588e-010	0.684579259911935	59
http://aschroder.com/	0.00032080209298206 1	0.682416977311465	2
http://gigaom.com/2011/11/02/with-card-case-square-launches-...	2.92862659591136e-008	0.680638448035459	467
http://advocateforcats.com/	2.54058468176938e-006	0.678915529472558	1
http://www.fanfiction.net/s/6545975/1/Dalton_Academy_A_Strange_New_World	4.92327250229103e-008	0.678056935852713	0
http://www.wired.com/gadgetlab/2011/11/apple-siri-down-outage/	2.92862659591136e-008	0.677336847765719	365
http://techcrunch.com/2011/10/20/google-reader-getting-overhauled-rem...	2.92862659591136e-008	0.67629943599706	2321

Tablica 6.6: Wyniki posortowane ze względu wyniki algorytmu Adapted PageRank

URL	Social PageRank	Adapted PageRank	inne
http://www.google.com/	0.401923361785337	0.423413021952925	105036899
http://www.whitehouse.gov/live?utm_source=wh.gov&utm_medium=shorturl&utm_campaign=shorturl	0.0214091482305497	0.505473285895009	86007664
http://www.facebook.com/#!/	0.335268158167203	0.501679745730669	61392273
http://www.facebook.com/	0.230330110142633	0.459285976449425	48756929
http://www.facebook.com/cocacola	0.13566999182353	0.485330922581667	37321394
http://www.youtube.com/watch?v=KQ6zr6kCPj8&ob=av2e	0.000395612647186402	0.466532962230913	29909940
http://www.youtube.com/watch?v=KQ6zr6kCPj8	0.0564150464732929	0.491406592113344	29899161
http://www.youtube.com/watch?v=uelHwf8o7_U&ob=av3e	0.00631144465629103	0.49688753793285	29270964
http://www.facebook.com/Starbucks	0.013362038395477	0.452895058368686	27268773
http://www.facebook.com/officialavatarmovie	0.00140130124762319	0.49004410348477	25434788
http://twitter.com/#!/fashionvault	0.00641336348164869	0.491954503817067	24301374
http://twitter.com/#!/annetkentroom/status/135146055103815680	0.00360623979656725	0.545398894442845	24271044
http://twitter.com/#!/nytopinion/opinionstaff/members	0.00599226414436835	0.555170611434515	24268516
http://twitter.com/#!/tng_s8	0.0124161620674808	0.549757391723913	24226321
http://twitter.com/#!/ladygaga	0.00124784122278975	0.471785959107418	24219342
http://twitter.com/#!/	2.01484307899899	0.525005266835375	24212409
http://twitter.com/#!/mentions	0.00258548098191947	0.506158745452038	24212112
http://twitter.com/#!/AlArabiya_Eng	0.000619717059772999	0.413060072058518	24211878
http://twitter.com/#!/saurik	0.00160646423354955	0.53723746921186	24199201
http://twitter.com/#!/AdamSchefter	0.00139124476293768	0.463074712423196	24197288
http://twitter.com/#!/RealTimeWWII	0.0132752648543194	0.506326607842251	24196016
http://www.youtube.com/watch?v=EPo5wWmKEaI	0.00784340198127326	0.484918411853964	23604659
http://gitimmersion.com/	13.9374242718583	0.483097956675647	22133807
http://www.orkut.com.br/Main#Home	0.161080122091895	0.519917489725066	21871919
http://www.orkut.com.br/Main#Home?rl=t	0.000426107317107683	0.494849754071563	21840258
http://www.youtube.com/watch?v=rYEDA3JcQqw	0.187982681938117	0.469456050873167	21099082

Tablica 6.7: Wyniki posortowane ze względu na dane z innych sieci socialnych

Rozdział 7

Wnioski

W pracy zostały przeanalizowane dwa algorytmy Adapted PageRank i SocialPageRank i przetestowane razem z algorytmem TF-IDF, i z prostym wskaźnikiem popularności strony pobranym z różnych portali społecznościowych. Analizując te wyniki testów trudno wyciągnąć wniosek, który z opisywanych algorytmów jest lepszy.

Adapted PageRank dawał dobre wyniki w wyszukiwaniu prac naukowych w dużym zbiorze dokumentów, gdzie nie radził sobie algorytm SocialPageRank. Z drugiej strony, w ograniczonym zbiorze, tylko prac naukowych, to właśnie algorytm SocialPageRank dawał zdecydowanie lepsze wyniki.

Jeśli weźmie się pod uwagę testy średniego rankingu (MMR) najlepsze wyniki otrzymywaliśmy przy użyciu algorytmu TF-IDF. Ale tutaj trzeba zaznaczyć, że to może być spowodowane niepełnymi danymi. Przeglądając dokładnie wyniki z wysokim rankingiem otrzymane z algorytmów SocialPageRank i Adapted PageRank, można było zauważyć, że są to dokumenty zgodne z tematem poszukiwanego zapytania, ale nie będące poszukiwaną stroną.

Używając systemu opisanego i zaimplementowanego w czasie powstania tej pracy, użytkownik może sterować wagami przypisanymi algorytmom. Dzięki temu jest w stanie sterować zbiorem wynikiem, i samemu zdecydować czy chce uzyskać wynik o dużej jakości tekstu (TF-IDF), czy uznany przez użytkowników za popularny według algorytmów SocialPageRank, Adapted PageRank, czy innych systemów społecznościowych.

Bibliografia

[1] Shenghua Bao, Guirong Xue, Xiaoyuan Wu, Yong Yu, Ben Fei, and Zhong Su. Optimizing web search using social annotations. In *Proc. WWW '07*, pages 501–510, Banff, Canada, 2007.

[2] Anlei Dong, Ruiqiang Zhang, Pranam Kolari, Jing Bai, Fernando Diaz, Yi Chang, Zhaohui Zheng, and Hongyuan Zha. Time is of the essence: improving recency ranking using twitter data. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 331–340, New York, NY, USA, 2010. ACM.

[3] The Apache Software Foundation. Apache lucene - scoring.

[4] Andreas Hotho, Robert Jäschke, Christoph Schmitz, and Gerd Stumme. Information retrieval in folksonomies: Search and ranking. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications*, volume 4011 of *Lecture Notes in Computer Science*, pages 411–426, Berlin/Heidelberg, June 2006. Springer.

[5] Yan'an Jin, Ruixuan Li, Kunmei Wen, Xiwu Gu, and Fei Xiao. Topic-based ranking in folksonomy via probabilistic model. *Artificial Intelligence Review*, pages 1–13, February 2011.

[6] Makoto Kato, Hiroaki Ohshima, Satoshi Oyama, and Katsumi Tanaka. Can social tagging improve web image search? In James Bailey, David Maier, Klaus-Dieter Schewe, Bernhard Thalheim, and Xiaoyang Wang, editors, *Web Information Systems Engineering - WISE 2008*, volume 5175 of *Lecture Notes in Computer Science*, chapter 19, pages 235–249. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2008.

[7] Michael Noll and Christoph Meinel. Web search personalization via social bookmarking and tagging. pages 367–380. 2008.

[8] Majdi Rawashdeh, Heung-Nam Kim, and Abdulmotaleb El-Saddik. Folksonomy-boosted social media search and ranking. In Francesco G. B. De Natale, Alberto Del Bimbo, Alan Hanjalic, B. S. Manjunath, and Shin'ichi Satoh, editors, *ICMR*, page 27. ACM, 2011.

[9] Ralf Schenkel, Tom Crecelius, Mouna Kacimi, Sebastian Michel, Thomas Neumann, Josiane X. Parreira, and Gerhard Weikum. Efficient top-k querying over social-tagging networks. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 523–530, New York, NY, USA, 2008. ACM.

[10] The Free Encyclopedia Wikipedia. Digg.

[11] The Free Encyclopedia Wikipedia. Facebook.

[12] The Free Encyclopedia Wikipedia. Google web toolkit.

[13] The Free Encyclopedia Wikipedia. Tfi-df.

[14] The Free Encyclopedia Wikipedia. Twitter.

[15] Y. Yanbe, A. Jatowt, S. Nakamura, and K. Tanaka. Can social bookmarking enhance search in the web? In *JCDL '07: Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 107–116, New York, NY, USA, 2007. ACM.

[16] Ding Zhou, Jiang Bian, Shuyi Zheng, Hongyuan Zha, and C. Lee Giles. Exploring social annotations for information retrieval. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 715–724, New York, NY, USA, 2008. ACM.