

Rok akademicki 2012/2013

Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki



PRACA DYPLOMOWA MAGISTERSKA

inż. Bartłomiej Jańczak

Gęstościowe grupowanie danych i wyznaczanie
najbliższego sąsiedztwa z użyciem nierówności trójkąta

Opiekun pracy
prof. dr hab. inż. Marzena Kryszkiewicz

Ocena:

.....
Podpis Przewodniczącego
Komisji Egzaminu Dyplomowego



Kierunek: Informatyka
Specjalność: Inżynieria Systemów Informatycznych
Data urodzenia: 1988.03.06
Data rozpoczęcia studiów: 2007.10.01

Życiorys

Urodziłem się 6 marca 1988 r. w Pszczynie. Naukę rozpoczęłem w Śląskiej Prywatnej Szkole Podstawowej im. Janusza Korczaka w Pszczynie. Od czwartej klasy uczęszczałem do Szkoły Podstawowej nr 18 w Pszczynie. Kolejnym etapem mojej edukacji była nauka w Publicznym Gimnazjum nr 1 w Pszczynie. Swoje zainteresowania mogłem rozwijać w klasie matematyczno-informatycznej w Liceum Ogólnokształcącym nr 1 im. Bolesława Chrobrego w Pszczynie. W latach 2006-2007 angażowałem się w pracę Grupy Twórczej Quark Pałacu Młodzieży w Katowicach. Maturę zdąłem w 2007 r., co pozwoliło mi na rozpoczęcie studiów na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej na kierunku Informatyka. Po dwóch latach studiów wybrałem specjalność Inżynieria Systemów Informacyjnych. Po zdobyciu tytułu inżyniera, we wrześniu 2011 roku kontynuowałem naukę na studiach magisterskich, na tej samej specjalności. W trakcie studiów zdobywałem doświadczenie pracując jako programista hurtowni danych w firmach Accenture i Infovide-Matrix, jako programista aplikacji mobilnych w firmie Samsung SPRC, oraz jako programista aplikacji JEE w firmie e-point S.A.

.....
Podpis studenta

EGZAMIN DYPLOMOWY

Złożył egzamin dyplomowy w dniu 20__ r

z wynikiem

Ogólny wynik studiów:

Dodatkowe wnioski i uwagi Komisji:

.....

.....

STRESZCZENIE

Niniejsza praca rozpoczyna się od wprowadzenia w dziedzinę gęstościowego grupowania danych, w tym przedstawienia algorytmu DBSCAN i wyszukiwania k sąsiedztwa. Następnie opisano teoretyczne podstawy wykorzystania nierówności trójkąta w gęstościowym grupowaniu danych i wyszukiwaniu k sąsiedztwa. Przypomniano metryki odległości i miarę kosinusową oraz opisano metodę wyznaczania kosinusowego sąsiedztwa za pomocą sąsiedztwa opartego na odległości euklidesowej.

W dalszej części zaprezentowano algorytmy TI-DBSCAN i TI-k-Neighborhood-Index wykorzystujące nierówność trójkąta oraz zaproponowano ich modyfikacje DBSCAN-PROJECTION i k-Neighborhood-Index-Projection wykorzystujące rzutowanie. W opisie algorytmów korzystających z nierówności trójkąta skupiono się na możliwych metodach doboru punktów referencyjnych pozwalających na efektywne grupowanie danych z zastosowaniem nierówności trójkąta, natomiast w przypadku algorytmów stosujących rzutowanie skoncentrowano się na metodach doboru wymiarów rzutowania umożliwiających efektywne grupowanie z wykorzystaniem rzutowania. Zaproponowano także dwa warianty wyszukiwania k sąsiadów w drzewie metrycznym VP-Tree, które stanowią rozwinięcie algorytmu wyszukiwania najbliższego sąsiada w drzewie VP-Tree dostępnego w literaturze. Wszystkie te algorytmy i indeksy zostały zaimplementowane.

Następnie znajduje się najistotniejsza część pracy, czyli prezentacja i omówienie wyników badań eksperymentalnych przeprowadzonych z użyciem tychże algorytmów i indeksów na testowych zbiorach danych o różnej liczbie wymiarów i charakterystyce.

W podsumowaniu zawarto wnioski obejmujące porównanie ze sobą wydajności badanych algorytmów oraz indeksów z użyciem metryki odległości euklidesowej i kosinusowej miary podobieństwa.

Slowa kluczowe: gęstościowe grupowanie danych, wyznaczanie najbliższego sąsiedztwa, nierówność trójkąta, VP-Tree, DBSCAN, TI-DBSCAN

DENSITY-BASED CLUSTERING AND NEAREST NEIGHBORHOOD SEARCH BY MEANS OF THE TRIANGLE INEQUALITY

The thesis begins with a brief introduction to a density-based clustering domain, including the description of the DBSCAN algorithm and k neighborhood search. Next, density-based clustering algorithms and k neighborhood search by means of the triangle inequality were explained. Also, distance metrics and the cosine measure were recalled, as well as the determination of cosine neighborhood by means of the Euclidean distance was described.

Then, TI-DBSCAN and TI-k-Neighborhood-Index algorithms were presented. In addition, their modifications DBSCAN-PROJECTION and k-Neighborhood-Index-Projection were proposed. The description of the algorithms using the triangle inequality mainly focused on methods of selecting reference points to enable efficient clustering by means of the triangle inequality, while in the case of algorithms which use projection their description mainly focused on methods of selecting projection dimensions to enable efficient clustering by means of projection. There were proposed also two variants of k nearest neighbors search in a VP-Tree, which are enhancements of the algorithm for nearest neighbor search in the VP-Tree that is available in the literature. All these algorithms and indexes were implemented.

The main part of the dissertation are research results and conclusions drawn from the experiments carried out by means of the implemented algorithms and indexes on diverse datasets. In the concluding part of the thesis, the performance of the examined algorithms and indexes was compared both for the Euclidean distance metric and the cosine similarity measure.

Keywords: density-based clustering, nearest neighborhood search, triangle inequality, VP-Tree, DBSCAN, TI-DBSCAN

Spis treści

| | |
|--|----|
| 1. Wprowadzenie | 11 |
| 1.1. Przegląd literatury | 12 |
| 1.2. Motywacja i cel pracy | 13 |
| 1.3. Układ pracy | 14 |
| 2. Miary odległości i podobieństwa do określania sąsiedztwa..... | 17 |
| 2.1. Metryki odległości i sąsiedztwo..... | 17 |
| 2.2. Miara kosinusowa i sąsiedztwo..... | 19 |
| 2.3. Wyznaczanie kosinusowego sąsiedztwa za pomocą sąsiedztwa opartego na odległości euklidesowej..... | 21 |
| 3. Algorytm grupowania gęstościowego DBSCAN..... | 25 |
| 4. Szacowanie odległości | 33 |
| 4.1. Wykorzystanie nierówności trójkąta..... | 33 |
| 4.2. Wykorzystanie indeksu metrycznego VP-Tree..... | 38 |
| 5. Algorytmy gęstościowego grupowania danych i wyszukiwania k sąsiedztwa z użyciem nierówności trójkąta..... | 43 |
| 5.1. Zastosowanie nierówności trójkąta w ujęciu podstawowym | 43 |
| 5.1.1. Algorytm TI-DBSCAN | 43 |
| 5.1.2. Algorytm TI-DBSCAN-REF | 46 |
| 5.1.3. Algorytm TI-k-Neighborhood-Index | 48 |
| 5.1.4. Algorytm TI-k-Neighborhood-Index-Ref | 51 |
| 5.2. Zastosowanie rzutowania..... | 53 |
| 5.2.1. Algorytm DBSCAN-PROJECTION | 53 |
| 5.2.2. Algorytm k-Neighborhood-Index-Projection..... | 54 |
| 5.3. Zastosowanie indeksu metrycznego w algorytmie wyszukiwania k sąsiadów | 54 |
| 6. Badania eksperymentalne..... | 59 |
| 6.1. Dane testowe | 61 |
| 6.2. Badania algorytmu k-Neighborhood-Index | 62 |
| 6.2.1. Badania algorytmu TI-k-Neighborhood-Index..... | 62 |

| | |
|---|-----|
| 6.2.2. Badania algorytmu k-Neighborhood-Index-Projection | 72 |
| 6.2.3. Porównanie algorytmów k-Neighborhood-Index-Projection z TI-k-Neighborhood-Index | 75 |
| 6.2.4. Badania algorytmu TI-k-Neighborhood-Index-Ref – wybór dwóch punktów referencyjnych..... | 76 |
| 6.2.5. Porównanie implementacji odmian algorytmu k-Neighborhood-Index | 80 |
| 6.3. Badania algorytmu kNN-Index-Vp-Tree | 85 |
| 6.3.1. Implementacja algorytmu..... | 85 |
| 6.3.2. Implementacja struktury punktu..... | 89 |
| 6.4. Porównanie algorytmów TI-k-Neighborhood-Index z kNN-Index-Vp-Tree..... | 91 |
| 6.5. Badania algorytmu DBSCAN | 94 |
| 6.5.1. Badania algorytmu TI-DBSCAN | 94 |
| 6.5.2. Badania algorytmu DBSCAN-PROJECTION | 106 |
| 6.5.3. Porównanie algorytmów DBSCAN-PROJECTION i TI-DBSCAN..... | 108 |
| 6.5.4. Badania algorytmu TI-DBSCAN-REF – wybór dwóch punktów referencyjnych | 109 |
| 6.5.5. Porównanie implementacji odmian algorytmu DBSCAN | 113 |
| 6.6. Badania algorytmu k-Neighborhood-Index – miara kosinusowa | 117 |
| 6.6.1. Badania algorytmu TI- k-Neighborhood-Index | 117 |
| 6.6.2. Badania algorytmu k-Neighborhood-Index-Projection – miara kosinusowa | 126 |
| 6.6.3. Porównanie algorytmów k-Neighborhood-Index-Projection z TI-k-Neighborhood-Index – miara kosinusowa..... | 129 |
| 6.6.4. Badania algorytmu TI-k-Neighborhood-Index-Ref – wybór dwóch punktów referencyjnych – miara kosinusowa..... | 130 |
| 6.6.5. Porównanie implementacji odmian algorytmu k-Neighborhood-Index – miara kosinusowa..... | 135 |
| 6.7. Badania algorytmu kNN-Index-Vp-Tree – miara kosinusowa | 139 |
| 6.7.1. Implementacja algorytmu..... | 139 |
| 6.7.2. Implementacja struktury punktu..... | 142 |

| | |
|---|-----|
| 6.8. Porównanie algorytmów TI-k-Neighborhood-Index z kNN-Index-Vp-Tree – miara kosinusowa | 144 |
| 7. Podsumowanie | 147 |
| Bibliografia | 151 |
| A Informacje dla użytkowników oprogramowania..... | 153 |
| A.1. Struktura plików..... | 153 |
| A.2. Plik parametrów uruchomienia aplikacji | 155 |
| A.3. Plik parametrów uruchomienia algorytmu..... | 156 |
| A.4. Wyniki uruchomień programu | 163 |
| A.5. Zbiory danych | 166 |
| A.6. Uruchomienie aplikacji | 166 |

1. Wprowadzenie

Współczesne systemy komputerowe agregują i generują ogromną ilość danych zawierających cenną dla biznesu trudno odkrywalną wiedzę. Jej znajdowaniem zajmuje się dziedzina informatyki zwana odkrywaniem wiedzy. Mimo, że jest ona stosunkowo młoda, to stworzyła wiele technik eksploracji danych, które dzięki swojej skuteczności oraz wydajności znalazły szerokie praktyczne zastosowanie w rozwiązywaniu problemów związanych z analizą danych.

Zasadniczą przyczyną rychłego rozwoju odkrywania wiedzy jest spopularyzowanie wydajnych metod pozyskiwania i gromadzenia informacji. Zjawisko to nie byłoby możliwe bez postępu technologicznego w dziedzinie urządzeń agregujących dane i systemów bazodanowych oraz dzięki upowszechnieniu urządzeń umożliwiających automatyczną rejestrację sposobu ich wykorzystania. Z punktu widzenia konsumenta można tu wyszczególnić kody kreskowe, karty płatnicze oraz szeroko pojęte urządzenia mobilne. Kolejnym wątkiem uwagi źródłem danych jest sieć Internet, w której możliwa jest rejestracja wielu czynności korzystających z niej użytkowników, którzy ponadto dobrowolnie umieszczają w niej wiele informacji o sobie, przykładowo na portalach społecznościowych.

Wyżej wymienione zjawiska mają wpływ na osiąganie ogromnych rozmiarów przez współczesne zbiory danych. Tempo ich wzrostu jest szybsze niż przewidywano jeszcze kilka lat temu. Ich gromadzenie i przechowywanie na nośnikach pamięci masowej nie stanowi problemów dla współczesnych systemów, natomiast działanie na takiej ilości danych, pomimo stale wzrastającej mocy obliczeniowej komputerów, wciąż jest wyzwaniem dla dzisiejszej informatyki. Zbiory danych same w sobie nie stanowią wielkiej wartości, jednakże rozsądnie wykorzystane mogą stać się cennym źródłem szczególnej wiedzy. Nierzadko użyteczna wiedza ukryta jest między pewnymi składowymi danych, więc do jej odkrycia konieczne są właściwe algorytmy. Zagadniением tym poświęcona jest dziedzina informatyki zwana eksploracją danych, której ideą jest wykorzystanie komputera do znajdowania ukrytych dla człowieka wartościowych prawidłowości w danych zgromadzonych w dużych repozytoriach.

Odkrywanie wiedzy jest procesem złożonym, na który najczęściej składają się następujące etapy:

- analiza danych – poznanie charakteru danych i określenie celu eksploracji,
- selekcja danych – czyszczenie, weryfikacja poprawności i wybór danych, które zostaną poddane dalszej analizie,
- transformacja danych – przekształcenie danych do odpowiedniej postaci, określenie strategii wobec danych niepełnych,
- eksploracja danych – ekstrakcja wiedzy z danych,
- interpretacja wyników – logiczna i graficzna wizualizacja wyników, wybór najbardziej interesującej wiedzy, wnioskowanie.

Kluczową fazą procesu odkrywania wiedzy jest eksploracja danych. Do zasadniczych metod eksploracji danych należą:

- grupowanie,
- klasyfikacja,
- odkrywanie asocjacji,
- regresja.

Każda z metod ujawnia różnego rodzaju korelacje pomiędzy danymi, z czego wynika ich odmienne zastosowanie.

W tej pracy skoncentrowałem się na zagadnieniu grupowania danych, które określane jest jako wyznaczanie zbiorów obiektów podobnych przy zachowaniu właściwości maksymalizacji podobieństwa obiektów należących do tych samych grup i minimalizacji podobieństwa obiektów należących do innych grup.

1.1. Przegląd literatury

Grupowanie danych jest popularną metodą o wielu zastosowaniach, dlatego nie trudno o jej opis w literaturze. W przypadku algorytmów gęstościowego grupowania danych, na których skupiłem się w niniejszej pracy wyjątkowo przydatne okazały się artykuły naukowe.

Jednym z najpopularniejszych algorytmów stosujących metody gęstościowe jest DBSCAN [5] stanowiący często punkt odniesienia dla porównań z innymi algorytmami gęstościowych grupowań. Ciekawym algorymem podobnym do DBSCAN jest NBC [16], który

inaczej niż DBSCAN oparty jest na k sąsiedztwie, a nie na sąsiedztwie epsilonowym. Wyznaczanie k najbliższych sąsiadów również zalicza się do metod gęstościowego grupowania danych. Godnym uwagi indeksem pozwalającym na efektywne wyznaczanie k sąsiedztwa w wielowymiarowych danych jest indeks metryczny VP-Tree [15]. Ważnym dopełnieniem powyższych artykułów jest publikacja traktująca o wyznaczaniu sąsiedztwa opartego na mierze kosinusowej za pomocą miary odległości euklidesowej [11].

Nową koncepcją zwiększenia wydajności wyżej wymienionych algorytmów jest wykorzystanie nierówności trójkąta do redukcji liczby kosztownych operacji wyznaczania podobieństwa obiektów. Na przykładzie algorytmu k-środków [4] przedstawiane już były próby wykorzystania nierówności trójkąta w algorytmach grupowania danych. Natomiast po raz pierwszy została ona użyta do gęstościowego grupowania danych w [8] za pomocą algorytmu TI-DBSCAN i w [7] za pomocą algorytmu TI-k-Neighborhood-Index. Dokonano również badania wpływu liczby punktów referencyjnych i strategii ich wyboru na efektywność tych algorytmów [14]. W literaturze nie doszukałem się jak zastosowanie nierówności trójkąta wpływa na wydajność algorytmów, których celem jest wyznaczanie obiektów podobnych ze względu na miarę podobieństwa kosinusowego. Nie znalazłem również porównania wydajności wyszukiwania k sąsiedztwa z użyciem nierówności trójkąta w przypadku, gdy zbiór danych jest uporządkowany względem punktu referencyjnego [7] z wyszukiwaniem w przypadku, gdy zbiór danych jest dostępny za pomocą indeksu metrycznego VP-Tree [15].

1.2. Motywacja i cel pracy

Grupowanie danych to proces powszechnie stosowany w kategoryzacji produktów, segmentacji klientów, organizacji obiektów czy rozpoznawaniu i analizie obrazów. Procesy te wymieniane są pośród kluczowych elementów, na których bazuje szeroko rozumiana sztuczna inteligencja. We współczesnym świecie algorytmy grupowania danych znajdują coraz szersze zastosowanie. Naukowcy opracowują coraz sprawniejsze algorytmy lub modyfikują istniejące, które dotychczas wydawały się optymalne. Nierazko zdarza się, że usprawnienia po wielokroć zwiększają wydajność dotychczasowych rozwiązań, co z kolei umożliwia przetwarzanie zbiorów danych z większą liczbą obiektów bądź atrybutów. Niekiedy może to oznaczać sposobność użycia tych algorytmów w nieosiągalnych dotychczas obszarach.

Jednym z najnowszych pomysłów na zwiększenie wydajności algorytmów grupowania danych jest zastosowanie nierówności trójkąta. Do dalszych badań tej metody motywują rezultaty otrzymane w [8], [7] i [14].

Celem pracy jest analiza algorytmów TI-DBSCAN i TI-k-Neighborhood-Index, w szczególności wpływ strategii wyboru punktów referencyjnych i ich liczby oraz zastosowania rzutowania na wymiar zamiast nierówności trójkąta na wydajność powyższych algorytmów. Ponadto celem pracy jest analiza wyszukiwania k sąsiedztwa przy użyciu indeksu metrycznego VP-Tree, a zwłaszcza wpływu wyboru implementacji wyszukiwania w indeksie VP-Tree, oryginalnej [15] lub korzystającej z zaproponowanego usprawnienia, na wydajność tej metody oraz porównanie jej z wydajnością wyszukiwania k sąsiedztwa w algorytmie TI-k-Neighborhood-Index. Istotnym celem pracy jest weryfikacja wydajności badanych algorytmów w zależności od charakterystyki zbioru danych, w tym danych tekstowych, które pozwolą na analizę wydajności badanych algorytmów w sytuacji gdy działają na danych wielowymiarowych. W szczególności celem pracy jest również zbadanie wpływu zastosowania miary kosinusowej jako miary podobieństwa na wydajność badanych algorytmów wyszukiwania k sąsiedztwa.

1.3. Układ pracy

W rozdziale drugim dokonałem opisu metryk odległości, wprowadziłem pojęcie miary kosinusowej oraz wyjaśniłem metodę wyznaczania kosinusowego sąsiedztwa za pomocą sąsiedztwa opartego na odległości euklidesowej. W rozdziale 3 dokonałem wprowadzenia w tematykę algorytmów grupowania danych, a następnie zaprezentowałem algorytm DBSCAN oraz wyszukiwanie k najbliższych sąsiadów. Opisy charakterystycznych dla wprowadzonych algorytmów cech oraz taksonomii uzupełniłem o pseudokody. Rozdział 4 przedstawia wykorzystane w pracy metody szacowania odległości. Opisy zostały wzbogacone o przykłady oraz ilustracje pozwalające na łatwiejsze zrozumienie danego podejścia. W rozdziale 5 opisałem badane algorytmy, będące modyfikacjami algorytmów opisanych w rozdziale 3, korzystające z wcześniej opisanych metod szacowania odległości.

W rozdziale 7 skupiłem się na praktycznej stronie pracy – przedstawieniu badań eksperymentalnych i uzyskanych wyników. W pierwszej kolejności opisałem wykorzystane w testach zbiory danych. Następnie zaprezentowałem wyniki badań wszystkich

zaimplementowanych algorytmów, a zwłaszcza skupiłem się na algorytmach TI-DBSCAN, TI-k-Neighborhood-Index oraz indeksie metrycznym VP-Tree. Pracę kończy podsumowanie, w którym zawarłem ostateczne konkluzje oraz dodatek A, w którym zamieściłem informacje dla użytkowników stworzonej w ramach pracy aplikacji.

Praca była finansowana przez Narodowe Centrum Badań i Rozwoju w ramach Programu Strategicznego "Interdyscyplinarny system interaktywnej informacji naukowej i naukowo technicznej", Umowa SP/I/1/77065/10 .

2. Miary odległości i podobieństwa do określania sąsiedztwa

Podobieństwo jest pojęciem fundamentalnym w niemal każdej dziedzinie naukowej. Przykładowo, w matematyce, geometryczne metody oceny podobieństwa wykorzystywane są do określania przystawania jak również w dziedzinach pokrewnych takich jak trygonometria. W biologii molekularnejjącym problemem jest mierzenie podobieństwa par białek. Zbiory rozmyte wykształcili własne miary podobieństwa znajdujące zastosowanie na polach zarządzania, medycyny czy meteorologii. W niniejszym rozdziale skupiłem się na wybranych miarach podobieństwa wektorów, tj. metrykach odległości oraz mierze kosinusowej. W dalszej części pracy zakładam, że obiekty (punkty) reprezentowane są przez wektory przestrzeni n wymiarowej. Każdy wektor u rozumiany jest jako sekwencja n komponentów $[u_1, \dots, u_n]$, gdzie komponent u_i jest wartością i -tego wymiaru u , $i = 1..n$. Wektor o wszystkich wymiarach równych 0 będzie nazywany *wektorem zerowym*, w przeciwnym przypadku będzie nazywany *wektorem niezerowym*.

2.1. Metryki odległości i sąsiedztwo

Metryką odległości w zbiorze wektorów D jest miara podobieństwa $distance: D \times D \rightarrow [0, +\infty)$, która spełnia następujące warunki dla dowolnych wektorów p, q oraz r w D :

1. $distance(p, q) = 0 \Leftrightarrow p = q;$
2. $distance(p, q) = distance(q, p);$
3. $distance(p, r) \leq distance(p, q) + distance(q, r);$

Warto zauważyć, że istnieje wiele miar odległości. W zależności od zastosowania, w danym przypadku, jedne miary mogą być stosowniejsze niż inne. Najpopularniejszą miarą odległości jest *odległość euklidesowa*. Odległość euklidesowa między wektorami p i q oznaczana jest jako $Euclidean(p, q)$ i definiowana jako:

$$Euclidean(p, q) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

Gdy stosowana jest odległość euklidesowa, to otoczenie wektora przyjmuje sferyczny kształt.

Innym przykładem popularnej miary odległości jest *odległość Manhattan*. Odległość Manhattan między wektorami p i q oznaczana jest jako $Manhattan(p, q)$ i definiowana w następujący sposób:

$$Manhattan(p, q) = \sum_{i=1}^n |p_i - q_i|$$

Gdy stosowana jest odległość Manhattan, to otoczenie wektora przyjmuje prostokątny kształt.

Zarówno odległość Manhattan jak i odległość euklidesowa są szczególnymi przypadkami *odległości Minkowskiego*. Odległość Minkowskiego rzędu m między wektorami p i q oznaczana jest jako $Minkowski(p, q)$ i definiowana jako:

$$Minkowski(p, q) = \sqrt[m]{\sum_{i=1}^n |p_i - q_i|^m}$$

Dla uproszczenia, bez straty ogólności, w swoich rozważaniach będę posługiwał się odlegością euklidesową jako metryką odległości.

Niech p będzie wektorem zbioru D , a odległość między wektorami p i q będzie wyrażana jako $distance(p, q)$. ε sąsiedztwo wektora p należącego do zbioru D jest oznaczane jako $\varepsilon NB(p)$ i zdefiniowane jako zbiór wszystkich wektorów w D różnych od p , których odległość od p nie przekracza ε , czyli:

$$\varepsilon NB(p) = \{q \in D \mid q \neq p \wedge distance(p, q) \leq \varepsilon\}.$$

Zbiór wszystkich wektorów w D , które są różne od p i bliższe p niż q będzie oznaczany przez $Closer(p, q)$; czyli:

$$Closer(p, q) = \{s \in D \mid s \neq p \wedge distance(s, p) < distance(q, p)\}.$$

k sąsiedztwo wektora p , oznaczane przez $kNB(p)$, jest definiowane jako zbiór wszystkich wektorów q w D , takich, że liczba wektorów różnych od p i bliższych p niż q jest mniejsza niż k ; czyli:

$$kNB(p) = \{q \in D | q \neq p \wedge |Closer(p, q)| < k\}.$$

Oczywiście jeżeli D zawiera co najmniej k elementów, to $kNB(p)$ także zawiera co najmniej k elementów. W przeciwnym przypadku $kNB(p)$ zawiera mniej niż k elementów.

Jeśli $kNB(p)$ zawiera co najmniej k elementów, to każdy k -elementowy podzbiór zbioru $kNB(p)$ jest nazywany zbiorem k sąsiadów wektora p w zbiorze wektorów D i oznaczany jako $kNN(p)$. W przeciwnym przypadku zbiór k sąsiadów jest definiowany jako $kNB(p)$.

W dalszych podrozdziałach, na podstawie [11], przedstawiono miarę kosinusową oraz wyznaczanie kosinusowego sąsiedztwa za pomocą sąsiedztwa opartego na odległości euklidesowej.

2.2. Miara kosinusowa i sąsiedztwo

W wielu aplikacjach, w szczególności odkrywających wiedzę w danych tekstowych, chemii, czy inżynierii biomedycznej, *miara podobieństwa kosinusowego* stosowana jest w celu znajdowania obiektów podobnych.

Miara podobieństwa kosinusowego między wektorami u i v oznaczana jest jako $\cosSim(u, v)$ i definiowana w następujący sposób:

$$\cosSim(u, v) = \frac{u \cdot v}{|u| \cdot |v|}$$

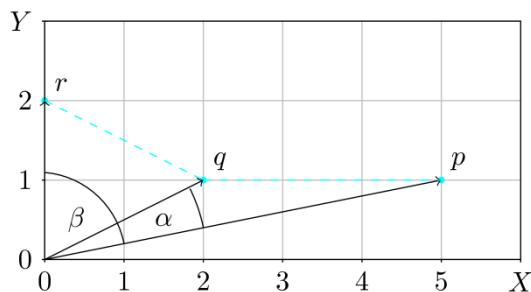
Znormalizowaną formą wektora u jest wektor $NF(u)$ powstały przez podzielenie każdego komponentu wektora u przez długość u , czyli $NF(u) = \frac{u}{|u|}$. Znormalizowany wektor przemnożony przez stałą α nazywamy α -znormalizowanym wektorem.

Przykład 1. Na rys. 1 przedstawiono trzy wektory p , q i r . Należy zwrócić uwagę, że odległość między p i q jest większa niż odległość euklidesowa (zaznaczona na rys. 1 kolorem cyjanowym) między r i q . Z drugiej strony, w sensie podobieństwa kosinusowego, p jest

bardziej podobne do q niż r , ponieważ kosinus kąta między p i q ($\cos Sim(p, q) = \cos \alpha$) jest większy niż kosinus kąta między r i q ($\cos Sim(r, q) = \cos \beta$).

W tab. 1 zamieszczono wartości podobieństwa kosinusowego między wektorami p , q i r .

Można zauważyć, że dla podobieństwa kosinusowego warunki $\cos Sim(p, q) \leq \cos Sim(p, r) + \cos Sim(r, q)$ oraz $-\cos Sim(p, r) \leq -\cos Sim(p, q) + (-\cos Sim(q, r))$ i $(1 - \cos Sim(p, r)) \leq (1 - \cos Sim(p, q)) + (1 - \cos Sim(q, r))$ nie są spełnione.



Rys. 1. Odległość euklidesowa i miara podobieństwa kosinusowego [11]

Tab. 1. Podobieństwo kosinusowe wektorów z rys. 1. [11]

| (u, v) | $\cos Sim(u, v)$ |
|----------|------------------|
| (p, q) | 0,965 |
| (p, r) | 0,196 |
| (q, r) | 0,447 |

Z powyższego przykładu płyną następujące wnioski:

- a) Nie ma gwarancji, że nierówność trójkąta będzie spełniona dla $\cos Sim$.
- b) Nie ma gwarancji, że nierówność trójkąta będzie spełniona dla $-\cos Sim$.
- c) Nie ma gwarancji, że nierówność trójkąta będzie spełniona dla $(1 - \cos Sim)$.

Ponieważ miara podobieństwa kosinusowego między niezerowymi wektorami u i v opiera się wyłącznie na kącie zawartym między nimi i nie zależy od ich długości, stąd obliczanie $\cos Sim(u, v)$ może być wyznaczone w oparciu o znormalizowane wektory u i v , tj. $NF(u)$ i $NF(v)$. Z powyższego spostrzeżenia wynikają poniższe własności:

- a) $\cos Sim(NF(u), NF(v)) = NF(u) \cdot NF(v);$
- b) $\cos Sim(u, v) = \cos Sim(NF(u), NF(v));$

$$c) \ cosSim(u, v) = NF(u) \cdot NF(v).$$

Niech u będzie wektorem zbioru D , a podobieństwo między wektorami u i v będzie wyrażana jako $cosSim(u, v)$. Kosinusowe ε sąsiedztwo wektora u należącego do zbioru D jest oznaczane jako $\varepsilon NB_{cos}(u)$ i zdefiniowane jako zbiór wszystkich wektorów w D różnych od u , których podobieństwo kosinusowe do u wynosi co najmniej ε , czyli:

$$\varepsilon NB_{cos}(u) = \{v \in D | v \neq u \wedge cosSim(u, v) \geq \varepsilon\}.$$

Zbiór wszystkich wektorów w D , które są różne od u i bardziej kosinusowo podobne do u niż v będzie oznaczany przez $MoreSimilar_{cos}(u, v)$; czyli:

$$MoreSimilar_{cos}(u, v) = \{s \in D | s \neq u \wedge cosSim(s, u) > cosSim(v, u)\}.$$

Kosinusowe k sąsiedztwo wektora u , oznaczane przez $kNB(u)$, jest definiowane jako zbiór wszystkich wektorów v w D , takich, że liczba wektorów różnych od u i są bardziej kosinusowo podobnych do u niż v jest mniejsza niż k ; czyli:

$$kNB(u) = \{v \in D | v \neq u \wedge |MoreSimilar_{cos}(u, v)| < k\}.$$

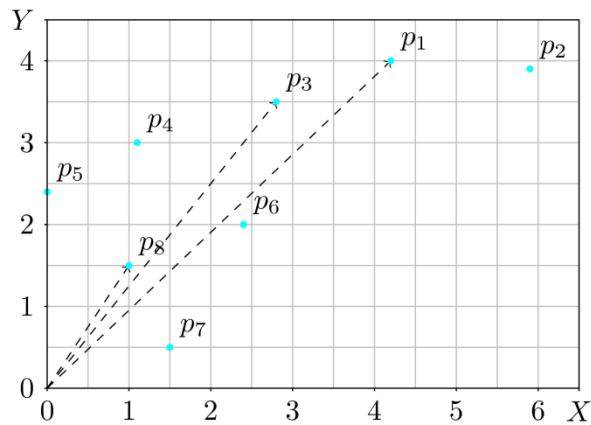
Jeśli $kNB(u)$ zawiera co najmniej k elementów, to każdy k -elementowy podzbiór zbioru $kNB(u)$ jest nazwany zbiorem kosinusowych k sąsiadów wektora u w zbiorze wektorów D . W przeciwnym przypadku zbiór kosinusowych k sąsiadów jest definiowany jako $kNB(p)$.

2.3. Wyznaczanie kosinusowego sąsiedztwa za pomocą sąsiedztwa opartego na odległości euklidesowej

W artykule [11] dowiedziono, że kosinusowe sąsiedztwo w zbiorze wektorów D może zostać wyznaczone za pomocą odpowiedniego sąsiedztwa opartego na odległości euklidesowej w zbiorze wektorów D' składającym się z α -znormalizowanych wektorów z D . Korzystając z uzyskanych wyników teoretycznych, zaproponowano tam także następujące podejście do wyznaczania sąsiedztwa opartego na podobieństwie kosinusowym.

W pierwszej kolejności początkowy zbiór wektorów $D = (p_{(1)}, \dots, p_{(m)})$ transformowany jest do $D' = (u_{(1)}, \dots, u_{(m)})$ - zbioru α -znormalizowanych wektorów z D . Następnie kosinusowe ε - sąsiedztwo (lub kosinusowe k sąsiedztwo) w zbiorze D jest wyznaczane jako ε' -sąsiedztwo (lub alternatywnie k sąsiedztwo) w zbiorze D' , gdzie $\varepsilon' = |\alpha| \sqrt{2 - 2\varepsilon}$. Warto zwrócić uwagę, że w przeciwnieństwie do kosinusowego ε -sąsiedztwa, ε' -sąsiedztwo spełnia własność nierówności trójkąta.

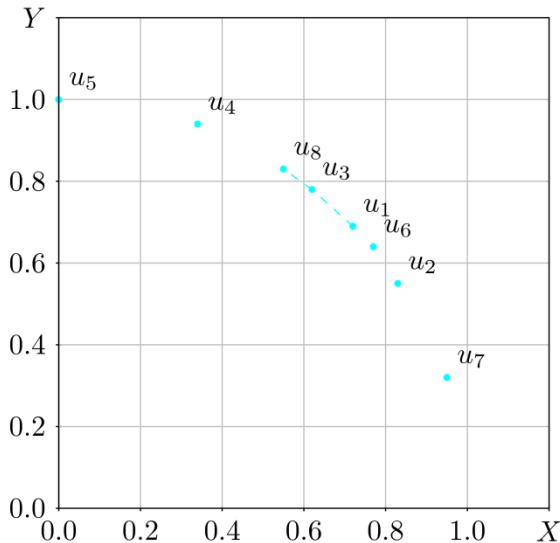
Przykład 2. W tab. 2 zdefiniowano przykładowy zbiór $D = (p_1, \dots, p_8)$. Zbiór ten przedstawiono również na rys. 2. Na rys. 3 zamieszczono zbiór $D' = (u_1, \dots, u_8)$ α -znormalizowanych wektorów z D , gdzie $\alpha = 1$. Warto zauważyć, że znormalizowane wektory zbioru D' mają długość równą $|\alpha|$, a punkty opisywane wektorami zbioru D' układają się na okręgu o środku w punkcie $(0,0)$ i promieniu $|\alpha|$.



Rys. 2. Przykładowy zbiór wektorów D [11]

Tab. 2. Przykładowy zbiór wektorów \mathbf{D} [11]

| <i>Wektor $p_{(i)}$</i> | $p_{(i)x}$ | $p_{(i)y}$ |
|------------------------------------|------------|------------|
| $p_{(1)}$ | 4.20 | 4.00 |
| $p_{(2)}$ | 5.90 | 3.90 |
| $p_{(3)}$ | 2.80 | 3.50 |
| $p_{(4)}$ | 1.10 | 3.00 |
| $p_{(5)}$ | 0.00 | 2.40 |
| $p_{(6)}$ | 2.40 | 2.00 |
| $p_{(7)}$ | 1.50 | 0.50 |
| $p_{(8)}$ | 1.00 | 1.50 |



Rys. 3. Zbiór wektorów α -znormalizowanych \mathbf{D}' [11]

Założymy, że chcemy znaleźć wektory najbardziej podobne do wektora p_3 ze względu na miarę kosinusową. Z rys. 2 widać, że są nimi wektory p_1 i p_8 . Analogicznie przedstawia się relacja między α -znormalizowanymi wektorami p_1 , p_3 i p_8 , tj. u_1 , u_3 i u_8 . Najbardziej podobnymi wektorami do u_3 , względem kosinusowego podobieństwa, są wektory u_1 i u_8 . Co więcej, wektory u_1 i u_8 są również najbardziej podobne do u_3 ze względu na miarę euklidesową, co zaznaczono na rys. 3 błękitną przerywaną linią.

3. Algorytm grupowania gęstościowego DBSCAN

Istnieje wiele rozwiązań problemu grupowania danych, czyli wyznaczania zbiorów obiektów podobnych przy zachowaniu właściwości maksymalizacji podobieństwa obiektów należących do tych samych grup i minimalizacji podobieństwa obiektów z różnych grup. Wiele algorytmów grupowania umożliwia stosowanie różnych miar podobieństwa obiektów. Do popularnych miar służących określaniu stopnia podobieństwa obiektów zalicza się odległość euklidesową i miarę kosinusową. Mnogość zastosowań grupowania, częstokroć o odmiennych wymaganiach co do rezultatu oraz specyficznych danych wejściowych (np. o różnej liczności, rozkładzie bądź liczbie atrybutów), prowadzi do dużej liczby wyspecjalizowanych algorytmów. W każdym z nich można doszukać się wad oraz zalet, jednakże nie znaleziono dotychczas uniwersalnego algorytmu. Często trudno porównywać algorytmy grupowania danych, ponieważ różnią się one nie tylko sposobem grupowania, ale także definicją pojęcia grupy.

Najpopularniejsza klasyfikacja algorytmów grupowania dzieli je na algorytmy oparte na podziale i algorytmy hierarchiczne. W przypadku pierwszej klasy kluczowym elementem jest znalezienie najlepszego podziału zbioru na z góry zadaną liczbę możliwie najbardziej jednorodnych grup. Początkowy podział odpowiednio ze zdeterminowaną strategią optymalizowany jest w kolejnych iteracjach zgodnie z przyjętą funkcją celu. Przykładami metod podziału są algorytmy k-średnich i k-medoidów. Wynikiem drugiej klasy algorytmów grupowania jest dendrogram¹ - drzewo, które iteracyjnie dzieli zbiór danych na coraz to mniejsze podzbiory dopóki każdy podzbiór składa się z jednego obiektu. W takiej hierarchii każdy węzeł drzewa reprezentuje klaster zbioru danych. Relacja między węzłami w dendrogramie a ich przodkami odpowiada relacji między podgrupami a grupami. Dendrogramy mogą być tworzone od liści w góre do korzenia (*podejście aglomeracyjne*) lub od korzenia w dół do liści (*podejście podziału*) poprzez scalanie lub podział klastrów z każdym krokiem algorytmu. Obie wymienione klasy algorytmów grupowania posiadają pewne wady. W przeciwieństwie do algorytmów opartych na podziale algorytmy hierarchiczne nie wymagają

¹ Dendrogram to diagram stosowany do prezentacji związków między elementami lub grupami elementów w kształcie przypominający drzewo.

arbitralnie zadanej liczby klastrów, jednakże wymagają zdefiniowania *warunku zakończenia* wskazującego kiedy proces podziału lub scalania powinien się zakończyć.

Wyniki wyżej wymienionych metod rzadko odpowiadają oczekiwaniom. Taki stan rzeczy można tłumaczyć nienaturalnym dla człowieka mechanizmem grupowania. Gdyby zlecić człowiekowi zadanie pogrupowania punktów dwuwymiarowej przestrzeni, to okazałoby się, że nie dzieliłby on zbioru hierarchicznie na kolejne podzbiory czy też nie próbowałby podzielić go na z góry określoną liczbę podzbiorów. Ludzie z łatwością rozpoznają grupy o dowolnych kształtach oraz szum. Głównym powodem, dla którego rozpoznajemy grupy jest fakt, iż wewnętrz każej z nich można wyszczególnić pełną gęstość punktów znacznie wyższą niż gęstość punktów poza grupą. Zatem do grupy zaliczamy punkty leżące w obszarze o gęstości wyraźnie większej niż punkty leżące w obszarze otaczającym ją. Do tak zdefiniowanej metody grupowania najbliższej jest algorytmem gęstościowym.

DBSCAN, czyli Density Based Spatial Clustering of Applications with Noise [5] jest jednym z najpopularniejszych algorytmów gęstościowego grupowania danych. Zaproponowany w 1996 roku wciąż jest sztandardowym algorytmem gęstościowym będącym punktem odniesienia w wielu pracach naukowych dotyczących tematyki grupowania oraz prezentujących nowe rozwiązania lub algorytmy.

Oprócz grup, czyli zbioru punktów o dużej gęstości punktów, DBSCAN rozpoznaje również szum, do którego należą punkty leżące w obszarze o małej gęstości. Algorytm ten wymaga podania jedynie dwóch parametrów wejściowych, które opisują najmniejszą grupę będącą obiektem zainteresowania. Pierwszym z nich jest promień ϵ sąsiedztwa *Eps*, a drugim minimalna liczność ϵ sąsiedztwa oznaczana przez *MinPts*. Para parametrów *Eps* i *MinPts* stanowi intuicyjną definicję najmniejszej akceptowalnej gęstości grupy, tym samym definiując minimalną liczbę wykrywanych grup.

Podstawowym pojęciem używanym w kontekście algorytmu DBSCAN jest *otoczenie epsilonowe* oznaczane przez $N_{Eps}(p)$ i definiowane jako zbiór takich punktów zbioru D , które są różne od p i nie bardziej odległe od p niż Eps , czyli:

$$N_{Eps}(p) = \{q \in D \mid \text{distance}(p, q) \leq Eps\}.$$

W DBSCAN wyróżnia się dwa rodzaje punktów wchodzące w skład klastra: punkty wewnętrz grupy zwane *punktami rdzeniowymi* oraz punkty leżące na obrzeżach klastra - *punkty brzegowe*.

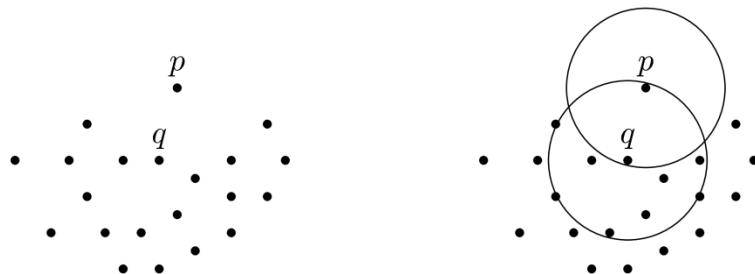
Punktem rdzeniowym nazywamy taki punkt p , którego otoczenie epsilonowe zawiera wymaganą liczbę punktów, czyli:

$$|N_{Eps}(p)| \geq MinPts.$$

Mówimy, że punkt q jest *bezpośrednio gęstościowo osiągalny* z punktu p względem Eps i $MinPts$, jeżeli q należy do otoczenia epsilonowego p oraz q jest punktem rdzeniowym:

$$q \in N_{Eps}(p) \wedge |N_{Eps}(q)| \geq MinPts.$$

Relacja *bezpośredniej gęstościowej osiągalności* jest symetryczna tylko dla punktów rdzeniowych. Na rys. 4 przedstawiono klaster, w którym zaznaczono pewien punkt brzegowy p oraz punkt rdzeniowy q . Okręgami zaznaczono otoczenie epsilonowe o promieniu Eps , a $MinPts$ wynosi 5. Rysunek prezentuje asymetryczny przypadek, w którym p jest bezpośrednio gęstościowo osiągalny z q , natomiast q nie jest bezpośrednio gęstościowo osiągalny z p .

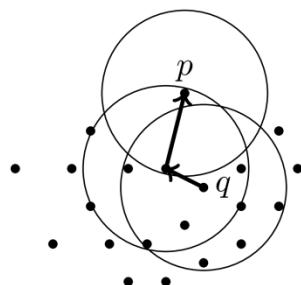


Rys. 4. Przykład ilustrujący punkty rdzeniowe i brzegowe [5]

Mówimy, że punkt p jest *gęstościowo osiągalny* z punktu q względem Eps i $MinPts$, jeżeli istnieje sekwencja punktów p_1, \dots, p_n $p_1 = q, p_n = p$ takich, że p_{i+1} jest *bezpośrednio gęstościowo osiągalny* z p_i , gdzie $i = 1, \dots, n - 1$. Relacja ta, to kanoniczne rozszerzenie relacji

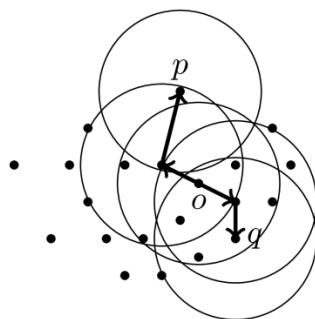
bezpośredniej gęstościowej osiągalności. Jest ona transzytywna, lecz nie jest symetryczna. Z tego powodu w [5] wprowadzono symetryczną relację gęstościowego połączenia.

Mówimy, że punkt p jest *gęstościowo połączony* z punktem q względem Eps i $MinPts$, jeżeli istnieje punkt o taki, że p i q są gęstościowo osiągalne z o względem Eps i $MinPts$. Na rys. 5 przedstawiono grupę, w której zaznaczono pewien punkt brzegowy p , punkt rdzeniowy q , okręgami zaznaczono otoczenia epsilonowe Eps pewnych punktów, a $MinPts$ wynosi 5. Analiza rysunku pozwala zauważyć, że punkt p jest gęstościowo osiągalny z q , natomiast q nie jest gęstościowo osiągalny z p .



Rys. 5. Przykład ilustrujący relację gęstościowej osiągalności [5]

Na rys. 6 przedstawiono grupę, w której zaznaczono punkty brzegowe p i q oraz punkt o . Okręgami wyznaczono otoczenia epsilonowe Eps punktów, a $MinPts$ wynosi 5. Studium rysunku pozwala spostrzec, że punkty p i q są gęstościowo osiągalne z o , czyli punkty p i q są gęstościowo połączone.



Rys. 6. Przykład ilustrujący relację gęstościowej łączności [5]

Wszystkie terminy niezbędne do przedstawienia gęstościowego pojęcia grupy zostały już wprowadzone. Niech D będzie zbiorem punktów. Grupą G względem Eps i $MinPts$ nazywamy każdy niepusty podzbiór D spełniający następujące warunki:

1. $\forall p, q: \text{ jeśli } p \in G \text{ i } q \text{ jest gęstościowo osiągalne z } p \text{ względem } Eps \text{ i } MinPts, \text{ wtedy } q \in G.$
2. $\forall p, q \in G: p \text{ jest gęstościowo połączone z } q \text{ względem } Eps \text{ i } MinPts.$

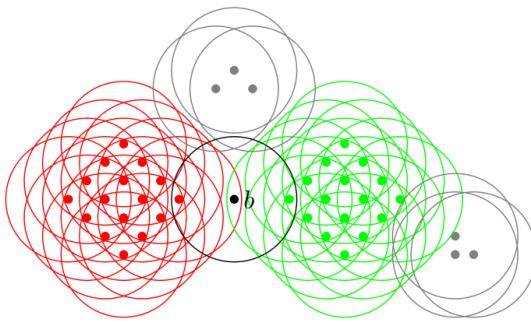
Niech C_1, \dots, C_k będą grupami zbioru punktów D względem Eps i $MinPts$. Szumem nazywamy podzbiór punktów zbioru D nie należących do żadnej z grup C_i , $i = 1, \dots, k$, czyli:

$$szum = \{p \in D \mid \forall i: p \notin C_i\}.$$

Algorytm DBSCAN iteruje wejściowy zbiór punktów i uruchamia procedurę wyznaczania nowej grupy *ExpandCluster* dla każdego punktu, który nie został jeszcze przypisany do żadnej z grup lub zidentyfikowany jako szum. *ExpandCluster* w pierwszej kolejności wyznacza otoczenie epsilonowe danego punktu i buduje nową grupę, jeśli ów punkt jest punktem rdzeniowym; w przeciwnym przypadku oznacza go jako szum. Proces tworzenia nowej grupy rozpoczyna się od dodania do niej punktów należących do otoczenia epsilonowego danego punktu. Następnie wszystkie punkty epsilonowego sąsiedztwa² dodawane są do zbioru *seeds* zawierającego punkty, które potencjalnie mogą rozszerzyć budowaną grupę. Algorytm iteruje zbiór *seeds* wyznaczając epsilonowe otoczenie dla każdego jego punktu. Jeżeli dany punkt jest punktem rdzeniowym, to wszystkie punkty należące do jego otoczenia epsilonowego, nie mające przypisanej żadnej grupy również dodawane są do nowoutworzonej grupy. Te z nich, które nie są oznaczone jako szum dodawane są do zbioru *seeds*. Opisany algorytm został zapisany w formie pseudokodu na wydruku 1.

Analiza kodu pozwala zauważyć, że algorytm DBSCAN jest deterministyczny z dokładnością do punktów brzegowych. Nie uwzględnia on, że punkty brzegowe znajdujące się między leżącymi blisko siebie grupami mogą należeć do kilku z grup. Taka sytuacja została przedstawiona na rys. 7.

² Epsilonowe sąsiedztwo jest epsilonowym otoczeniem danego punktu bez niego samego.



Rys. 7. Ilustracja sytuacji, w której przynależność do jednej z grup (czerwonej bądź zielonej) punktu brzegowego b zależy od kolejności w jakiej DBSCAN będzie badał punkty

```

DBSCAN (SetOfPoints, Eps, MinPts)
    // SetOfPoints is UNCLASSIFIED
    ClusterId := nextId(NOISE);
    for i from 1 TO SetOfPoints.size do
        Point := SetOfPoints.get(i);
        if Point.CliId = UNCLASSIFIED then
            if ExpandCluster(SetOfPoints, Point, ClusterId, Eps, MinPts) then
                ClusterId := nextId(ClusterId);
            endif;
        endif;
    endfor;
end; //DBSCAN

ExpandCluster (SetOfPoints, Point, CliId, Eps, MinPts) : Boolean
    seeds := SetOfPoints.regionQuery(Point, Eps);
    if seeds.size < MinPts THEN // no core point
        SetOfPoints.changeCliId(Point, NOISE);
        return false;
    else // all points in seeds are density reachable from Point
        SetOfPoints.changeCliId(seeds, CliId);
        seeds.delete(Point);
        while seeds <> Empty do
            currentP := seeds.first();
            result := SetOfPoints.regionQuery(currentP, Eps);
            if result.size >= MinPts then
                for i from 1 to result.size do
                    resultP := result.get(i);
                    if resultP.CliId IN (UNCLASSIFIED, NOISE) then
                        if resultP.CliId = UNCLASSIFIED then
                            seeds.append(resultP);
                        endif;
                        SetOfPoints.changeCliId(resultP, CliId);
                    endif;
                endfor;
            endif;
            seeds.delete(currentP);
        endwhile;
        return true;
    endif;
end; //ExpandCluster

```

Wydruk 1. Zapis algorytmu DBSCAN w formie pseudokodu

Wynik wykonania algorytmu DBSCAN zależy od kolejności przeglądania punktów, ponieważ punkt brzegowy zakwalifikowany do pewnej grupy, w rezultacie rozbudowy kolejnych grup, może zostać przypisany do innych grup. Problem ten może zostać rozwiązany poprzez przechowywanie w każdym punkcie zamiast jednego identyfikatora *clusterId* zbioru identyfikatorów. Jednakże podobnie jak autorzy algorytmu, problem ten uznaję za pomijalny.

4. Szacowanie odległości

Szacowanie odległości jest przybliżonym określaniem jej wartości. Działanie to pozwala uniknąć wielokrotnego jej obliczania między pewnym wektorem q a wszystkimi wektorami danego zbioru wektorów D . W następujących podrozdziałach opisałem użyte przeze mnie metody szacowania odległości między wektorami. Wykorzystanie nierówności trójkąta zostało opisane na podstawie artykułów [8], [7] oraz [10], a indeks metryczny został opisany na podstawie artykułów [15] i [3].

4.1. Wykorzystanie nierówności trójkąta

Na początek warto przypomnieć nierówność trójkąta.

Twierdzenie 1. Dla dowolnych wektorów u, v i r :

$$\text{distance}(u, v) \geq \text{distance}(u, r) - \text{distance}(v, r).$$

Stąd, odległości u i v do arbitralnie wybranego wektora r (czyli różnica $\text{distance}(u, r) - \text{distance}(v, r)$) zapewniają pesymistyczne oszacowanie odległości między u i v . Owa pesymistyczna odległość między wektorami u i v w odniesieniu do wektora r będzie oznaczana jako $\text{distance}^r(u, v)$, czyli:

$$\text{distance}^r(u, v) = \text{distance}(u, r) - \text{distance}(v, r).$$

Wektor r niezbędny do wyznaczania pesymistycznego oszacowania będzie nazywany *wektorem referencyjnym*. Oczywiście, wartość pesymistycznego oszacowania zależy od wyboru wektora referencyjnego.

Załóżmy, że dla każdego rozważanego wektora odległość do punktu referencyjnego r została już obliczona. W takiej sytuacji określenie pesymistycznego oszacowania $\text{distance}^r(u, v)$ odległości między wektorami u i v jest niezwykle szybkie, jako że wymaga

jedynie odjęcia uprzednio obliczonych wartości $distance(v, r)$ od $distance(u, r)$. Jeżeli szukane są wektory v nie dalsze od danego wektora u niż Eps , to gdy pesymistyczne oszacowanie $distance^r(u, v)$ jest większe niż Eps , wtedy odległość wektora v od u jest również większa niż Eps , czyli:

$$distance^r(u, v) > Eps \Rightarrow distance(u, v) > Eps.$$

W takim przypadku nie jest konieczne obliczanie odległości między wektorami u i v , którego złożoność zależy liniowo od liczby wymiarów, aby upewnić się, że jest większa od Eps .

Rozważmy wektory u , q_f i q takie, że $distance(q, r) > distance(q_f, r) > distance(u, r)$. Wtedy pesymistyczne oszacowanie odległości między wektorami q i u :
 $distance^r(q, u) = distance^r(q, r) - distance^r(u, r) >$
 $distance^r(q_f, r) - distance^r(u, r) = distance^r(q_f, u)$. Stąd, $distance(q, r) > distance(q_f, r) > distance(u, r)$ implikuje $distance^r(q, u) > distance^r(q_f, u)$. Zatem, jeśli $distance^r(q_f, u) > Eps$, to bez żadnych dodatkowych obliczeń wiadomo, że $distance^r(q, u) > Eps$, z czego wynika, że odległość między q i u jest większa od Eps .

Analogicznie, rozważmy wektory u , q_p i q takie, że $distance(q, r) < distance(q_p, r) < distance(u, r)$. Wtedy, $distance^r(u, q) = distance^r(u, r) - distance^r(q, r) > distance^r(u, r) - distance^r(q_p, r) = distance^r(u, q_p)$. Stąd, $distance(q, r) < distance(q_p, r) < distance(u, r)$ implikuje $distance^r(u, q) > distance^r(u, q_p)$. Czyli, jeżeli $distance^r(u, q_p) > Eps$, wtedy bez żadnych dodatkowych obliczeń wiadomo, że $distance^r(u, q) > Eps$, z czego wynika, że odległość między q i u jest większa od Eps .

Powyższe obserwacje prowadzą do następującego wniosku.

Wniosek 1. Niech r będzie dowolnym wektorem, a D zbiorem wektorów posortowanych niemalejąco względem ich odległości do r . Niech q będzie dowolnym wektorem z D , q_f będzie wektorem następującym po q w D takim, że $distance^r(q_f, q) > Eps$, a q_p będzie wektorem poprzedzającym q w D takim, że $distance^r(q, q_p) > Eps$. Wtedy:

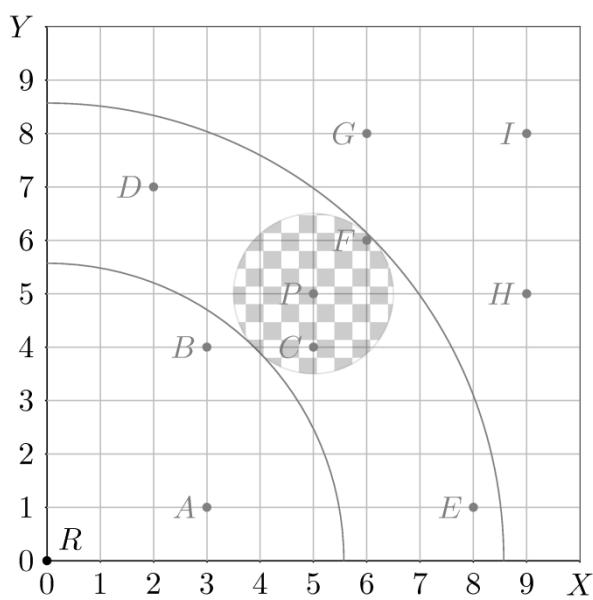
- a) q_f i wszystkie wektory następujące po q_f w D nie należą do otoczenia epsilonowego q w D ;
- b) q_p i wszystkie wektory poprzedzające q_p w D nie należą do otoczenia epsilonowego q w D .

Zatem, warto jest uporządkować wektory zbioru D względem odległości do wektora referencyjnego r , ponieważ umożliwia to prostą eliminację potencjalnie liczniego podzbioru wektorów nie należących do otoczenia epsilonowego rozpatrywanego wektora.

Przykład 3. Niech R będzie wektorem referencyjnym o współrzędnych $(0,0)$. Na rys. 8 przedstawiono zbiór wektorów D przestrzeni dwuwymiarowej. Tab. 3 przedstawia zbiór wektorów D uporządkowany niemalejąco względem odległości do wektora referencyjnego R . Rozważmy wyznaczenie Eps sąsiedztwa wektora P o $Eps = 1,5$. Odległość pomiędzy wektorem P a wektorem referencyjnym jest równa $7,07$ ($distance(P, R) = 7,07$). Pierwszym wektorem q_f następującym po wektorze P w D takim, że $distance(q_f, R) - distance(P, R) > Eps$ jest wektor G . Natomiast pierwszym wektorem q_p poprzedzającym wektor P w D takim, że $distance(P, R) - distance(q_p, R) > Eps$ jest wektor B . Z uwagi na wniosek 1, tylko wektory następujące po B i poprzedzające G w D (czyli wektory C, P, D, E, F) mogą należeć do $N_{Eps}(P)$. Zatem, C, D, E oraz F są jedynymi wektorami, dla których należy obliczyć odległość do wektora P w celu właściwego wyznaczenia otoczenia $N_{Eps}(P)$. Przestrzeń potencjalnych sąsiadów wektora P wyznaczona w oparciu o wektor referencyjny R została oznaczona na rys. 8 jako pole ograniczone przez okręgi o środkach w R . Otoczenie $N_{Eps}(P)$ zostało oznaczone na rys. 8 jako koło o środku w P , pokryte szachownicą.

Tab. 3. Zbiór wektorów D , wraz z odległościami do wektora P i wektora referencyjnego R [14]

| <i>Nazwa punktu</i> | <i>X</i> | <i>Y</i> | <i>odl. do R</i> | <i>odl. do P</i> |
|---------------------|----------|----------|------------------|------------------|
| A | 3 | 1 | 3,16 | 4,47 |
| B | 3 | 4 | 5,00 | 0,24 |
| C | 5 | 4 | 6,40 | 1,00 |
| P | 5 | 5 | 7,07 | 0,00 |
| D | 2 | 7 | 7,28 | 3,61 |
| E | 8 | 1 | 8,06 | 5,00 |
| F | 6 | 6 | 8,48 | 1,41 |
| G | 6 | 8 | 10,00 | 3,16 |
| H | 9 | 5 | 10,30 | 4,00 |
| I | 9 | 9 | 12,73 | 5,66 |

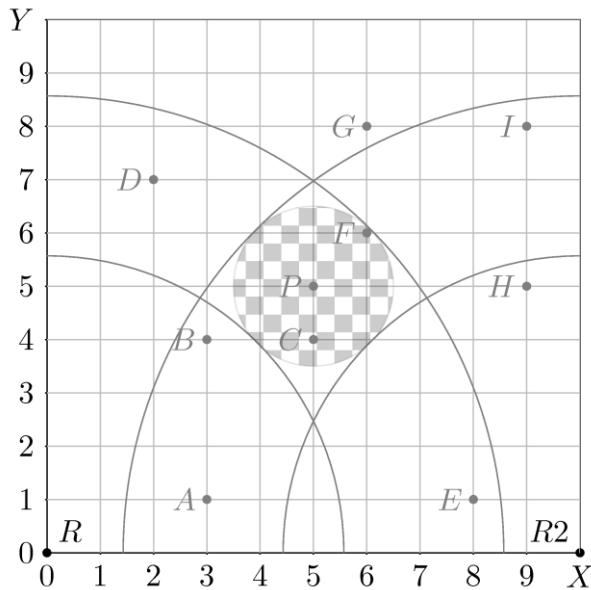


Rys. 8. Zbiór wektorów D oraz wektor referencyjny R [14]

Przykład 4. Przykład ten stanowi rozszerzenie przykładu 3 o wektor referencyjny $R2$. Na rys. 9 przedstawiono zbiór wektorów D . W tab. 4 zamieszczono zbiór wektorów D uporządkowany niemalejąco względem odległości jego wektorów do wektora referencyjnego R .

Tab. 4. Zbiór wektorów D , wraz z odległościami do wektora P i wektorów referencyjnych R i $R2$ [14]

| Nazwa punktu | X | Y | odl. do R | odl. do P | odl. do R2 |
|--------------|---|---|-----------|-----------|------------|
| A | 3 | 1 | 3,16 | 4,47 | 7,07 |
| B | 3 | 4 | 5,00 | 0,24 | 8,06 |
| C | 5 | 4 | 6,40 | 1,00 | 6,40 |
| P | 5 | 5 | 7,07 | 0,00 | 7,07 |
| D | 2 | 7 | 7,28 | 3,61 | 10,63 |
| E | 8 | 1 | 8,06 | 5,00 | 2,24 |
| F | 6 | 6 | 8,48 | 1,41 | 7,21 |
| G | 6 | 8 | 10,00 | 3,16 | 8,94 |
| H | 9 | 5 | 10,30 | 4,00 | 5,10 |
| I | 9 | 9 | 12,73 | 5,66 | 9,06 |



Rys. 9. Zbiór wektorów D oraz wektory referencyjne R i $R2$ [14]

Rozważmy wyznaczenie Eps sąsiedztwa wektora P o $Eps = 1,5$. Z przykładu 3 wiemy, że wektor referencyjny R ogranicza zbiór potencjalnych rozwiązań do wektorów C, D, E oraz F . Zgodnie z opisem w artykule [8] rzeczywiste odległości muszą być obliczone jedynie dla wektorów spełniających nierówność trójkąta w stosunku do wszystkich wektorów referencyjnych. W rozpatrywanym przypadku, ze zbioru wektorów, dla których należy obliczyć rzeczywiste odległości ze względu na wektor referencyjny R , na podstawie niespełnienia nierówności trójkąta dla punktu $R2$ wykluczone są wektory D i E .

Nierówność trójkąta można również zastosować w określaniu k sąsiedztwa dowolnego wektora q w zbiorze wektorów D , ponieważ problem ten można sprowadzić do wyznaczania otoczenia epsilonowego. Dla każdego wektora q , można określić wartość Eps w taki sposób, że $N_{Eps}(q) = kNB(q)$. Najmniejsza wartość Eps taka, że $N_{Eps}(q) = kNB(q)$, będzie nazywana *promieniem* $kNB(q)$.

Twierdzenie 2. Niech $Eps = \max(\{distance(u, q) | u \in kNB(q)\})$. Wtedy $kNB(q) = N_{Eps}(q)$ i Eps jest promieniem $kNB(q)$.

Twierdzenie 3. Jeżeli $|N_{Eps}(q)| \geq k$, to $N_{Eps}(q) \supseteq kNB(p)$.

W praktyce odległość Eps , w zasięgu której gwarantowane jest znalezienie k sąsiadów wektora q , jest zmniejszana w trakcie obliczania odległości między q a kolejnymi wektorami z D , różnymi od q .

4.2. Wykorzystanie indeksu metrycznego VP-Tree

W celu efektywnego wyznaczania wektorów podobnych zostały zaproponowane specyficzne indeksy, takie jak kd drzewo [1] czy VP-Tree [15]. Poniżej zamieszczam ich krótką charakterystykę.

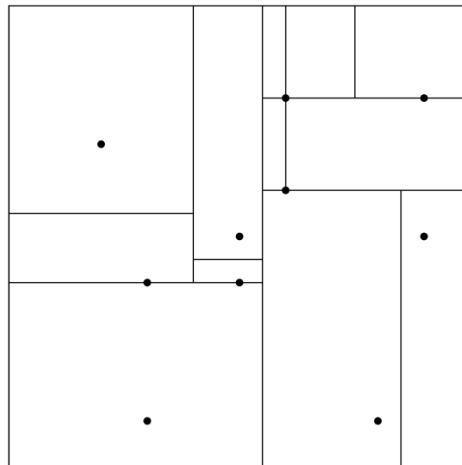
Drzewo kd jest drzewem BSP³ tworzonym poprzez rekurencyjną bisekcję zbioru wektorów na podstawie ich położenia względem hiperpłaszczyzny tnącej. W każdym przebiegu rekurencji zbiór wektorów dzielony jest na podzbiory względem mediany rozkładu tworzonego przez rzutowanie zbioru wektorów na k-ty wymiar. Numer wymiaru, na który dokonywane jest rzutowanie, zmienia się cyklicznie i ma tę samą wartość na danym poziomie drzewa kd. Niestety struktura ta posiada wadę - gdy liczba wymiarów wzrasta, to wyszukiwanie w drzewie kd szybko zaczyna się sprowadzać do odwiedzania wszystkich jego węzłów.

Podobnie jak drzewo kd, indeks metryczny VP-Tree⁴ (drzewo z punktami/wektorami obserwacyjnymi) jest drzewem BSP. Każdy węzeł tego indeksu dzieli przestrzeń na dwie podprzestrzenie. W procesie podziału zamiast korzystać ze współrzędnych, indeks metryczny VP-Tree posługuje się odległością do wybranego *wektora obserwacyjnego*. Wektory bliskie wektorowi obserwacyjnemu tworzą *lewą (wewnętrzna)* podprzestrzeń, podczas gdy *prawa (zewnętrzna)* podprzestrzeń składa się z dalszych wektorów. Rekurencyjne stosowanie wyżej opisanego podziału prowadzi do utworzenia drzewa binarnego. Każdy węzeł tego indeksu zawiera wektor obserwacyjny danej przestrzeni, a także odległość progową, na podstawie której dokonano podziału na podprzestrzenie oraz wskazania na wektory obserwacyjne podprzestrzeni – swoich potomków. W dalszej części pracy będę posługiwał się terminem indeks metryczny w odniesieniu do indeksu metrycznego VP-Tree z wyłączeniem rozdziału 7, który stanowi podsumowanie pracy.

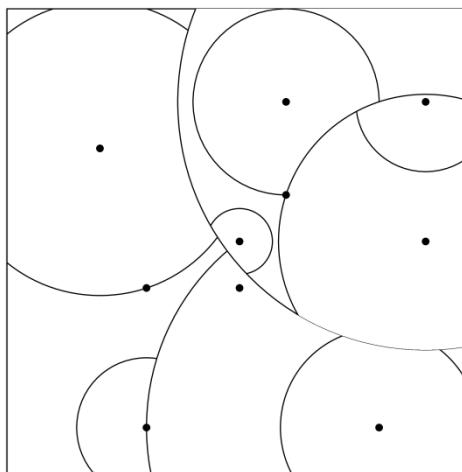
W procesie budowy indeksu metrycznego przestrzeń metryczna dekomponowana jest przy użyciu sferycznych cięć o średnikach w punktach obserwacyjnych. Rozwiążanie to kontrastuje z wykorzystaniem podziału hiperpłaszczyznami w drzewie kd. Obie metody dekompozycji zostały zilustrowane na przykładzie pewnego zbioru punktów D przestrzeni dwuwymiarowej na rysunkach rys. 10 i rys. 11.

³ BSP – (ang. Binary Search Partitioning) metoda dokonująca rekurencyjnego podziału przestrzeni na podprzestrzenie za pomocą hiperpłaszczyzn. Podział ten tworzy reprezentację obiektów w przestrzeni zwanej drzewem BSP. Wyszukiwanie w drzewie BSP jest wyszukiwaniem binarnym.

⁴ VP – Vantage Point



Rys. 10. Dekompozycja przykładowego zbioru punktów D za pomocą drzewa kd



Rys. 11. Dekompozycja przykładowego zbioru punktów D za pomocą indeksu metrycznego

Poniżej prezentuję wykorzystanie indeksu VP-Tree do znajdowania najbliższego wektora, które zaproponowano w [15]. Założymy, że dana jest pewna przestrzeń metryczna $(S, distance)$ oraz skończony podzbiór $D \subseteq S$ reprezentujący zbiór wektorów, wśród których wyszukiwani są najbliżsi sąsiedzi. Dla wektora $q \in S$ problem wyznaczania najbliższego sąsiada sprowadza się do znalezienia wektora najmniej odległego od q i należącego do D . Operacja ta będzie dalej oznaczana jako $1NN(q, D)$. Ponieważ wektor najbliższy wektorowi q może być od niego dość odległy, warto wprowadzić odległość progową τ , poza którą nie jesteśmy zainteresowani istnieniem sąsiadów q . Należy zwrócić uwagę, że w czasie obliczania $1NN(q, D)$ wartość τ

może być redukowana z każdym kolejnym napotkanym bliższym sąsiadem q . Wyszukiwanie sąsiedztwa ograniczane w wyżej wymieniony sposób będzie oznaczane przez $1NN|_\tau(q, D)$.

W dalszej części rozważań założymy, że zasięg funkcji odległości przestrzeni jest równy przedziałowi $[0; 1]$. Ponieważ każdy metryczny zasięg może być sprowadzony do przedziału $[0; 1]$ bez wpływu na relację sąsiedztwa⁵, obostrzenie to może zostać wprowadzone bez straty ogólności.

Wprowadzimy także funkcje $\Pi_p: S \rightarrow [0; 1]$ i $distance_p: S \times S \rightarrow [0; 1]$, przydatne do wyjaśnienia metody wyznaczania najbliższego sąsiada. Niech $(S, distance)$ będzie ograniczoną przestrzenią metryczną $[0; 1]$. Dla danego wektora $p \in S$ oraz $a, b \in S$:

1. $\Pi_p: S \rightarrow [0; 1], \Pi_p(a) = distance(a, p)$
2. $distance_p: S \times S \rightarrow [0; 1], distance_p(a, b) = |\Pi_p(a) - \Pi_p(b)| = |distance(a, p) - distance(b, p)|$

Funkcja $distance_p$ jest symetryczna oraz spełnia nierówność trójkąta, stąd $distance(a, b) \geq |distance(a, p) - distance(b, p)| = distance_p(a, b)$, a konsekwencją tej relacji jest implikacja $distance_p(a, b) \geq \tau \Rightarrow distance(a, b) \geq \tau$. Czyli, jeśli w procesie poszukiwania napotkano już wektor x w odległości τ od q , to w dalszej części poszukiwań nie należy brać pod uwagę elementów, dla których $distance_p(q, x) \geq \tau$.

Dla pewnego wektora p rozważmy przeciwdziedzinę $\Pi_p(D)$ dziedziny D w $[0; 1]$. Przez μ oznaczymy medianę $\Pi_p(D)$ dzielącą $[0; 1]$ na $[0; \mu]$ i $[\mu; 1]$. Pierwszy z tych przedziałów leży wewnątrz sfery $S(p, \mu)$, natomiast drugi z nich składa się z wektorów leżących na powierzchni oraz poza sferą. Dziedziny pierwszego i drugiego przedziału oznaczymy odpowiednio przez S_{pL} i S_{pR} . Innymi słowy wektor obserwacyjny p dzieli zbiór wektorów D na podzbiory S_{pL} (lewy/wewnętrzny) i S_{pR} (prawy/zewnętrzny).

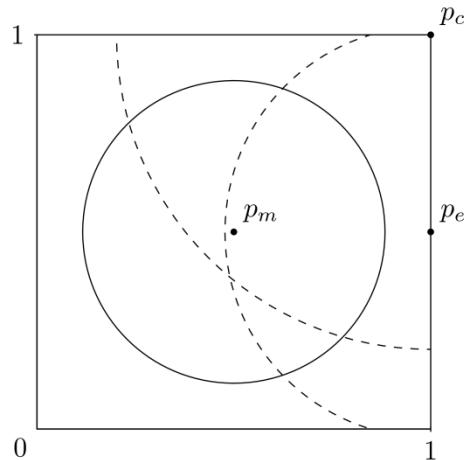
Niech N_L oznacza licznosć podzbioru S_{pL} , a N_R licznosć podzbioru S_{pR} . W ogólnosci niewiele można powiedzieć o relacji między N_L i N_R nie czyniąc żadnych założeń co do natury przestrzeni metrycznej. Wiadomo jednak, że podział wektorów z S_D jest najlepszy, gdy $N_L = N_R$, czyli gdy nie więcej niż jeden z wektorów S_D leży na sferze $S(p, \mu)$.

Na tym etapie rozważań powinno już być zrozumiałe, że jedne wektory obserwacyjne mogą być lepsze od innych. Jako przykład rozważmy dwuwymiarową przestrzeń unormowaną, w której znajduje się równomiernie rozłożony zbiór wektorów. W zadanej sytuacji należy

⁵ Ograniczone przestrzenie metryczne mogą zostać w prosty sposób przeskalowane. Nieograniczone przestrzenie metryczne mogą zostać dostosowane dzięki zastosowaniu wzoru:

$$\overline{distance}(a, b) = \frac{distance(a, b)}{1 + distance(a, b)}$$
 [15]

wybrać μ w taki sposób, aby fragment powstałego wycinka koła zajmował połowę powierzchni przestrzeni. Rozważmy trzy przykładowe wektory obserwacyjne p_c , p_e i p_m . Rozmieszczenie wektorów obserwacyjnych wraz z przynależnymi im liniami podziału zilustrowano na rys. 12. W tab. 5 znajdują się własności wektorów obserwacyjnych.



Rys. 12. Przykład rozmieszczenia wektorów obserwacyjnych [15]

Tab. 5. Własności przykładowych wektorów obserwacyjnych [15]

| Wektor obserwacyjny | Promień linii podziału | Długość linii podziału |
|---------------------|------------------------|------------------------|
| p_c | 0,7979 | 1,2533 |
| p_e | 0,5225 | 1,338 |
| p_m | 0,3989 | 2,5066 |

Wiemy już, że najlepszy wektor obserwacyjny p , to taki, dla którego co najwyżej jeden z wektorów D leży na sferze $S(p, \mu)$. Oczywistym jest, że prawdopodobieństwo położenia wektora x na powierzchni $S(p, \mu)$ jest proporcjonalne do powierzchni $S(p, \mu)$, a dla rozpatrywanego przypadku proporcjonalne do długości linii podziału. Stąd najlepszym z przykładowych wektorów obserwacyjnych jest wektor p_c , dla którego linia podziału jest najkrótsza. Z powyższego przykładu płynie intuicyjny wniosek, że wektory znajdujące się blisko skrajnych punktów przestrzeni są najlepszymi wektorami obserwacyjnymi.

5. Algorytmy gęstościowego grupowania danych i wyszukiwania k sąsiedztwa z użyciem nierówności trójkąta

Analiza złożoności algorytmów przedstawionych w rozdziale 3 prowadzi do wniosku, że największy wpływ na wydajność procesu grupowania ma obliczanie odległości między wektorami. W celu poprawienia wydajności grupowania, czyli ograniczenia liczby obliczanych odległości, można posłużyć się metodami szacowania odległości opisanymi w rozdziale 4. W niniejszym rozdziale opisałem badane przeze mnie odmiany algorytmów grupowania gęstościowego. Odmiany te zostały oparte na uprzednio opisanych algorytmach i metodach szacowania odległości.

5.1. Zastosowanie nierówności trójkąta w ujęciu podstawowym

Użycie nierówności trójkąta w ujęciu podstawowym jako metody szacowania odległości poprawia wydajność algorytmu gęstościowego grupowania danych. W kolejnych podrozdziałach opisałem jej zastosowanie w algorytmie DBSCAN oraz w wyszukiwaniu k najbliższych sąsiadów.

5.1.1. Algorytm TI-DBSCAN

W artykule [8] po raz pierwszy przedstawiono wykorzystanie nierówności trójkąta jako sposobu zwiększenia wydajności algorytmu DBSCAN. Opublikowaną wersję algorytmu autorzy nazwali TI-DBSCAN. Jego pseudokod został zamieszczony na wydruku 2 i wydruku 3. W algorytmie tym, po znanej z DBSCAN inicjalizacji punktów i oznaczeniem ich jako nieprzypisane do żadnej z grup, obliczana jest odległość każdego z punktów do uprzednio wybranego punktu referencyjnego. Następnie w oparciu o te wartości punkty grupowanego zbioru sortowane są niemalejąco.

```

algorithm TI-DBSCAN(D, Eps, MinPts)
    D' := empty set of points;
    TI-Init(D);
    ClusterId := nextId(NOISE);
    for each point p in the ordered set D starting from the first point until
        last point in D do
            if TI-ExpandCluster(D, D', p, ClusterId, Eps, MinPts) then
                ClusterId := nextId(ClusterId);
            endif;
        endfor;
        return D'; //D' is clustered set of points
    end; //TI-DBSCAN

function TI-ExpandCluster(D, D', p, ClusterId, Eps, MinPts)
    seeds := TI-Neighborhood(D, p, Eps);
    p.neighborsNr := p. neighborsNr + |seeds|;
    if p.neighborsNr < MinPts then
        p.clusterId := NOISE;
        for each point q in seeds do
            add p to q.border; q.neighborsNr := q.neighborsNr + 1;
        endfor;
        p.border := Ø; move p from D to D'; return false;
    else
        assign ClusterId to all b in p.border;
        assign ClusterId to p; assign ClusterId to all q in seeds;
        for each point q in seeds do
            q.neighborsNr := q.neighborsNr + 1;
        endfor;
        p.border := Ø; move p from D to D';
        while |seeds| > 0 do
            curP := first point in seeds;
            curSeeds := TI-Neighborhood(D, curP, Eps);
            curP.neighborsNr := curP.neighborsNr + |curSeeds|;
            if curP.neighborsNr < MinPts then
                for each point q in curSeeds do
                    q.neighborsNr := q.neighborsNr + 1;
                endfor;
            else
                for each point q in curSeeds do
                    q.neighborsNr := q.neighborsNr + 1;
                    if q.clusterId = UNCLASSIFIED then
                        assign ClusterId to q; move q from curSeeds to seeds;
                    else
                        delete q from curSeeds;
                    endif;
                endfor;
                assign ClusterId to all b in curP.border;
            endif;
            curP.border := Ø; move curP from D to D';
            delete curP from seeds;
        endwhile;
        return true;
    endif;
end; //TI-ExpandCluster

```

Wydruk 2. Pseudokod funkcji TI-DBSCAN i TI-ExpandCluster algorytmu TI-DBSCAN

```

function TI-Init(D)
    rPoint := selectReferencePoint();
    for each point p in D do
        p.clusterId := UNCLASSIFIED; p.neighborsNr := 1; p.border := Ø;
        p.dist := distance(p, rPoint);
    end for
    sort D non-decreasingly w.r.t. p.dist;
end; //TI-Init

function TI-Neighborhood(D, p, Eps)
    return TI-Backward-Neighborhood(D, p, Eps) ∪ TI-Forward-Neighborhood(D,
        p, Eps);
end; //TI-Neighborhood

function TI-Backward-Neighborhood(D, p, Eps)
    seeds := {};
    backwardThreshold := Eps - p.dist;
    for each point q in the ordered set D starting from the point immediately
        preceding point p until first point in D do
        if q.dist < backwardThreshold then
            break;
        endif;
        if distance(q,p) ≤ Eps then
            append q to seeds;
        endif;
    endfor;
    return seeds;
end; // TI-Backward-Neighborhood

function TI-Forward-Neighborhood(D, p, Eps)
    Seeds := {};
    forwardThreshold := Eps + p.dist;
    for each point q in the ordered set D starting from the point immediately
        following point p until the last point in D do
        if q.dist > forwardThreshold then
            break;
        endif;
        if distance(p,q) ≤ Eps then
            append q to seeds;
        endif;
    endfor;
    return seeds;
end; // TI-Forward-Neighborhood

```

Wydruk 3. Pseudokod funkcji TI-Init, TI-Neighborhood, TI-Backward-Neighborhood i TI-Forward-Neighborhood algorytmu TI-DBSCAN

Istotną różnicą między algorytmem TI-DBSCAN a DBSCAN jest zastosowanie i rozszerzenie opisanej w [9] koncepcji usuwania na bieżąco ze zbioru D przeanalizowanych punktów. Podejście to zakłada, że każdy punkt dodatkowo przechowuje liczbę dotychczas znalezionych sąsiadów oraz zbiór punktów brzegowych. Informacje te pozwalają na bieżąco usuwać z analizowanego zbioru danych D wszystkie zbadane punkty.

Dzięki zastosowanemu rozwiązaniu funkcja TI-ExpandCluster iteruje jedynie po zbiorze dotychczas niezbadanych punktów. Największa korzyść płynącą z działania na ograniczonym zbiorze punktów występuje w procesie wyznaczania sąsiedztwa punktu, ponieważ możliwe jest uniknięcie wielu obliczeń rzeczywistych odległości między punktami. Algorytm zapewnia, że operacja wyznaczania odległości między dwoma punktami zostanie wykonana najwyżej raz. Jednakże takie zapewnienie nie przychodzi bez ceny, którą jest wzrost zapotrzebowania na pamięć oraz skomplikowania algorytmu.

Kluczową modyfikacją algorytmu TI-DBSCAN względem DBSCAN jest użycie funkcji TI-Neighborhood, która dla zadanego punktu zwraca jego epsilonowe sąsiedztwo. Wynik tej funkcji stanowi sumę teoriomnogościową zbiorów będących rezultatami wywołań funkcji TI-Backward-Neighborhood i TI-Forward-Neighborhood wyszukujących punkty należące do epsilonowego sąsiedztwa danego punktu znajdujące się w uporządkowanym zbiorze punktów odpowiednio przed i po danym punkcie. Obie funkcje przeglądają ten zbiór odpowiednio w tył i przód, do momentu napotkania punktu, którego odległość do punktu referencyjnego różni się od odległości badanego punktu do punktu referencyjnego o więcej niż wartość Eps. Dalsze przeglądanie punktów indeksu w danym kierunku jest zbędne, ponieważ, zgodnie z teorią przedstawioną w rozdziale 4.1., nie należą one do epsilonowego sąsiedztwa weryfikowanego punktu. Pseudokod dotyczący omówionych funkcji zamieściłem na wydruku 3.

5.1.2. Algorytm TI-DBSCAN-REF

TI-DBSCAN-REF jest odmianą algorytmu TI-DBSCAN opisaną w artykule [8] wykorzystującą wiele punktów referencyjnych do estymacji odległości między dwoma punktami. Dodatkowe punkty referencyjne używane są tylko wtedy, gdy podstawowy punkt referencyjny, względem którego posortowany jest zbiór punktów, nie wystarcza do oszacowania, czy dany punkt należy do epsilonowego otoczenia badanego punktu. Rzeczywista odległość między dwoma punktami obliczana jest tylko wtedy, gdy żaden z punktów referencyjnych nie pozwala na oszacowanie przynależności do otoczenia epsilonowego. Dodatkowym kosztem wynikającym z posłużenia się wieloma punktami referencyjnymi jest wyznaczanie odległości między nimi a punktami badanego zbioru.

Na wydruku 4 zamieściłem pseudokod funkcji składających się na algorytm TI-DBSCAN-REF różnych od funkcji algorytmu TI-DBSCAN. Szarym tłem wyróżniłem fragmenty pseudokodu, różne od odpowiedniego pseudokodu algorytmu TI-DBSCAN.

```

function TI-REF-Init(D)
    rPoints := selectReferencePoints();
    for each point p in D
        p.clusterId := UNCLASSIFIED; p.neighborsNr := 1; p.border := Ø;
        for each i := 1..rPoints.size() do
            p.dists[i] := distance(p, rPoints[i]);
        endfor;
    end for;
    sort D non-decreasingly w.r.t. p.dists[0];
end; //TI-REF-Init

function TI-REF-Neighborhood(D, p, Eps)
    return TI-REF-Backward-Neighborhood(D, p, Eps)  $\cup$  TI-REF-Forward-
        Neighborhood(D, p, Eps);
end; //TI-REF-Neighborhood

function TI-REF-Backward-Neighborhood(D, p, Eps)
    backwardThreshold := Eps - p.dists[0]; seeds := {};
    for each point q in the ordered set D starting from the point immediately
        preceding point p until first point in D do
        if q.dists[0] < backwardThreshold then
            break;
        endif;
        candidateNeighbor := true; i := 1;
        while candidateNeighbor and (i ≤ |p.dists|) do
            if |q.dists[i] - p.dists[i]| > Eps then
                candidateNeighbor := false;
            else
                i := i + 1;
            endif;
        endwhile;
        if candidateNeighbor && (distance(q, p) ≤ Eps) then
            append q to seeds;
        endif;
    endfor;
    return seeds;
end; // TI-REF-Backward-Neighborhood

function TI-Forward-Neighborhood(D, p, Eps)
    forwardThreshold := Eps + p.dists[0]; seeds := {};
    for each point q in the ordered set D starting from the point immediately
        following point p until the last point in D do
        if q.dists[0] > forwardThreshold then
            break;
        endif;
        candidateNeighbor := true; i := 1;
        while candidateNeighbor and (i ≤ |p.dists|) do
            if |q.dists[i] - p.dists[i]| > Eps then
                candidateNeighbor := false;
            else
                i := i + 1;
            endif;
        endwhile;
        if candidateNeighbor && (distance(p, q) ≤ Eps) then
            append q to seeds;
        endif;
    endfor;
    return seeds;
end; // TI-Forward-Neighborhood

```

Wydruk 4. Pseudokod funkcji algorytmu TI-DBSCAN-REF różnych od funkcji algorytmu TI-DBSCAN

5.1.3. Algorytm TI-k-Neighborhood-Index

W artykule [7] zaproponowano wykorzystanie nierówności trójkąta w wyszukiwaniu k sąsiedztwa. Opisanemu algorytmowi twórcy nadali nazwę TI-k-Neighborhood-Index. Dla k sąsiedztwa zastosowanie nierówności trójkąta jest bardziej zawiłe niż w przypadku DBSCAN. Skomplikowanie to wynika z faktu, że na początku wykonania algorytmu promień Eps sąsiedztwa nie jest znany, w dodatku należy wyznaczyć go niezależnie dla każdego punktu w oparciu o dystans do k-tego najbliższego sąsiada.

Pierwszym krokiem algorytmu jest wyznaczenie odległości do punktu referencyjnego dla wszystkich punktów zbioru, po czym wykonywane jest jego sortowanie w porządku niemalejącym względem obliczonych odległości. Następnie główna pętla algorytmu tworzy indeks, poprzez wyznaczenie wszystkim punktom ich k sąsiedztwa za pomocą funkcji TI-k-Neighborhood. Pierwszym etapem wyznaczania k sąsiedztwa jest szacowanie minimalnej wartości Eps gwarantującej znalezienie k sąsiadów. Punkty znajdujące się w promieniu o oszacowanej wartości tworzą zbiór kandydatów. Faktyczny promień k sąsiedztwa jest mniejszy bądź równy oszacowanej wartości Eps. Aby wyznaczyć rzeczywiste k sąsiedztwo, w kolejnym etapie dokonuje się weryfikacji zbioru kandydatów. Zaczynając od badanego punktu, posortowany zbiór danych iterowany jest zarówno w przód jak i w tył, do momentu napotkania punktów, których odległość od punktu referencyjnego różni się od odległości badanego punktu do punktu referencyjnego o wartość większą niż Eps. Wartość Eps aktualizowana jest za każdym razem gdy znajdowany jest nowy sąsiad, który wstawiany jest do zbioru kandydatów. Pseudokod algorytmu TI-k-Neighborhood-Index i jego funkcji zamieściłem na wydrukach 5 6 i 7.

```
algorithm TI-k-Neighborhood-Index(D, k)
    TI-Init(D);
    for each point p in ordered set D starting from the first point until
        last point in D do
        insert(position od point p, TI-k-Neighborhood(D, p, k))
        into k-Neighborhood-Index
    endfor;
end; //TI-k-Neighborhood-Index

function TI-Init(D)
    rPoint := selectReferencePoint();
    for each point p in D do
        p.dist := distance(p, rPoint);
    endfor;
    sort D non-decreasingly w.r.t. p.dist;
end; //TI-Init
```

Wydruk 5. Pseudokod funkcji TI-k-Neighborhood-Index i TI-Init algorytmu TI-k-Neighborhood-Index

```

function TI-k-Neighborhood(D, p, k)
    b := p;
    f := p;
    backwardSearch := PrecedingPoint(D, b);
    forwardSearch := FollowingPoint(D, b);
    k-Neighborhood := {};
    i := 0;
    Find-k-Candidate-Neighbours(D, p, k, i, b, f, k-Neighborhood,
        backwardSearch, forwardSearch);
    p.Eps := max({e.dist | e ∈ k-Neighborhood});
    Verify-k-Candidate-Neighbours-Backward(D, p, b, backwardSearch, k,
        k-Neighborhood);
    Verify-k-Candidate-Neighbours-Forward(D, p, f, forwardSearch, k,
        k-Neighborhood);
    return k-Neighborhood;
end; //TI-k-Neighborhood

function Find-k-Candidate-Neighbours(D, p, k, i, b, f, k-Neighborhood,
    backwardSearch, forwardSearch)
    while (backwardSearch and forwardSearch and i < k) do
        if p.dist - b.dist < f.dist - p.dist then
            dist := distance(b, p);
            i := i + 1;
            e := (b, dist);
            insert e into k-Neighborhood holding it sorted wrt. e.dist;
            backwardSearch := PrecedingPoint(D, b);
        else
            dist := distance(f, p);
            i := i + 1;
            e := (f, dist);
            insert e into k-Neighborhood holding it sorted wrt. e.dist;
            forwardSearch := FollowingPoint(D, b);
        endif;
    endwhile;
    while backwardSearch and i < k) do
        dist := distance(b, p);
        i := i + 1;
        e := (b, dist);
        insert e into k-Neighborhood holding it sorted wrt. e.dist;
        backwardSearch := PrecedingPoint(D, b);
    endwhile;
    while forwardSearch and i < k) do
        dist := distance(f, p);
        i := i + 1;
        e := (f, dist);
        insert e into k-Neighborhood holding it sorted wrt. e.dist;
        forwardSearch := FollowingPoint(D, b);
    endwhile;
end; //Find-k-Candidate-Neighbours

function PrecedingPoint(D, p)
    if there is a point in D preceding p then
        p := point immediately preceding p in D;
        return true;
    endif;
    return false;
end; //PrecedingPoint

```

Wydruk 6. Pseudokod funkcji TI-k-Neighborhood, Find-k-Candidate-Neighbours i PrecedingPoint algorytmu TI-k-Neighborhood-Index

```

function FollowingPoint(D, p)
    if there is a point in D following p then
        p := point immediately following p in D;
        return true;
    end if;
    return false;
end; //FollowingPoint

function Verify-k-Candidate-Neighbours-Backward(D, p, b,
backwardSearch, k, k-Neighborhood)
    while backwardSearch and ((p.dist - b.dist) ≤ p.Eps) do
        dist := distance(b, p);
        e := (b, dist);
        if dist < p.Eps then
            i := |{e ∈ k-Neighborhood | e.dist = p.Eps}|;
            if |k-Neighborhood| - i ≥ k - 1 then
                delete each element e with e.dist = p.Eps from k-Neighborhood;
                insert e into k-Neighborhood holding it sorted wrt. e.dist;
                p.Eps := max({e.dist | e ∈ k-Neighborhood});
            else
                insert e into k-Neighborhood holding it sorted wrt. e.dist;
            endif;
        else
            if dist = p.Eps then
                insert e into k-Neighborhood holding it sorted wrt. e.dist;
            endif;
        endif;
        backwardSearch := PrecedingPoint(D, b);
    endwhile;
end; //Verify-k-Candidate-Neighbours-Backward

function Verify-k-Candidate-Neighbours-Forward(D, p, f, forwardSearch,
k, k-Neighborhood)
    while forwardSearch and ((f.dist - p.dist) ≤ p.Eps) do
        dist := distance(f, p);
        e := (f, dist);
        if dist < p.Eps then
            i := |{e ∈ k-Neighborhood | e.dist = p.Eps}|;
            if |k-Neighborhood| - i ≥ k - 1 then
                delete each element e with e.dist = p.Eps from k-Neighborhood;
                insert e into k-Neighborhood holding it sorted wrt. e.dist;
                p.Eps := max({e.dist | e ∈ k-Neighborhood});
            else
                insert e into k-Neighborhood holding it sorted wrt. e.dist;
            endif;
        else
            if dist = p.Eps then
                insert e into k-Neighborhood holding it sorted wrt. e.dist;
            endif;
        endif;
        forwardSearch := FollowingPoint(D, b);
    endwhile;
end; //Verify-k-Candidate-Neighbours-Forward

```

Wydruk 7. Pseudokod funkcji FollowingPoint, Verify-k-Candidate-Neighbours-Backward i Verify-k-Candidate-Neighbours-Forward algorytmu TI-k-Neighborhood-Index

5.1.4. Algorytm TI-k-Neighborhood-Index-Ref

Oprócz TI-k-Neighborhood-Index artykułu [7] proponuje jego odmianę stosującą wiele punktów referencyjnych. Odmiana ta będzie dalej nazywana TI-k-Neighborhood-Index-Ref. Zastosowanie wielu punktów referencyjnych pozwala na przyspieszenie weryfikacji zbioru kandydatów do k sąsiedztwa. Usprawnienie tą metodą wcześniejszych kroków algorytmu nie jest możliwe, ponieważ bez szacunkowej znajomości Eps nie można zaostrzyć warunku wykluczania elementów ze zbioru kandydatów. Dodatkowy warunek spełnienia nierówności trójkąta został dodany w funkcjach Verify-k-Candidate-Neighbours-Backward i Verify-k-Candidate-Neighbours-Forward. Pseudokod algorytmu TI-k-Neighborhood-Index-Ref zamieściłem na wydrukach 8 i 9. Szarym tłem wyróżniłem fragmenty różne od odpowiedniego pseudokodu algorytmu TI-k-Neighborhood-Index.

```
function TI-REF-Init(D)
    rPoints := selectReferencePoints();
    for each point p in D do
        for each i := 1..rPoints.size() do
            p.dists[i] := distance(p, rPoints[i]);
        endfor;
    endfor;
    sort D non-decreasingly w.r.t. p.dists[1];
end; //TI-REF-Init

function Is-Candidate-Neighbor-By-Additional-Reference-Points(p, q,
    Eps)
    candidateNeighbor := true;
    i := 2;
    while candidateNeighbor and (i ≤ |p.dists|) do
        if |q.dists[i] - p.dists[i]| > Eps then
            candidateNeighbor := false;
        else
            i := i + 1;
        endif;
    endwhile;
    return candidateNeighbor;
end; //Is-Candidate-Neighbor-By-Additional-Reference-Points
```

Wydruk 8. Pseudokod funkcji TI-REF-Init i Is-Candidate-Neighbor-By-Additional-Reference-Points algorytmu TI-k-Neighborhood-Index-Ref różnych od funkcji algorytmu TI-k-Neighborhood-Index

```

function Verify-k-Candidate-Neighbours-Forward(D, p, f, forwardSearch,
k, k-Neighborhood)
while forwardSearch and ((f.dists[1] - p.dists[1]) ≤ p.Eps) do
    if Is-Candidate-Neighbor-By-Additional-Reference-Points(p, b, p.Eps)
    then
        dist := distance(f, p);
        e := (f, dist);
        if dist < p.Eps then
            i := |{e ∈ k-Neighborhood | e.dist = p.Eps}|;
            if |k-Neighborhood| - i ≥ k - 1 then
                delete each element e with e.dist = p.Eps from k-Neighborhood;
                insert e into k-Neighborhood holding it sorted wrt. e.dist;
                p.Eps := max({e.dist | e ∈ k-Neighborhood});
            else
                insert e into k-Neighborhood holding it sorted wrt. e.dist;
            endif;
        else
            if dist = p.Eps then
                insert e into k-Neighborhood holding it sorted wrt. e.dist;
            endif;
        endif;
    endif;
    forwardSearch := FollowingPoint(D, b);
endwhile;
end; //Verify-k-Candidate-Neighbours-Forward

function Verify-k-Candidate-Neighbours-Backward(D, p, b,
backwardSearch, k, k-Neighborhood)
while backwardSearch and ((p.dists[1] - b.dists[1]) ≤ p.Eps) do
    if Is-Candidate-Neighbor-By-Additional-Reference-Points(p, b, p.Eps)
    then
        dist := distance(b, p);
        e := (b, dist);
        if dist < p.Eps then
            i := |{e ∈ k-Neighborhood | e.dist = p.Eps}|;
            if |k-Neighborhood| - i ≥ k - 1 then
                delete each element e with e.dist = p.Eps from k-Neighborhood;
                insert e into k-Neighborhood holding it sorted wrt. e.dist;
                p.Eps := max({e.dist | e ∈ k-Neighborhood});
            else
                insert e into k-Neighborhood holding it sorted wrt. e.dist;
            endif;
        else
            if dist = p.Eps then
                insert e into k-Neighborhood holding it sorted wrt. e.dist;
            endif;
        endif;
    endif;
    backwardSearch := PrecedingPoint(D, b);
endwhile;
end; //Verify-k-Candidate-Neighbours-Backward

```

Wydruk 9. Pseudokod funkcji Verify-k-Candidate-Neighbours-Forward i Verify-k-Candidate-Neighbours-Backward algorytmu TI-k-Neighborhood-Index-Ref różnych od funkcji algorytmu TI-k-Neighborhood-Index

5.2. Zastosowanie rzutowania

Alternatywnym sposobem usprawnienia algorytmów gęstościowego grupowania jest zastosowanie rzutowania na wymiar [12]. Łatwo zauważyć, że dla każdego wymiaru $l, l \in [1, \dots, n]$, i każdych dwóch wektorów u i v prawdziwe jest, że:

$$|u_l - v_l| = \sqrt{(u_l - v_l)^2} \leq \sqrt{\sum_{i=1 \dots n} (u_i - v_i)^2} = Euclidean(u, v).$$

Stąd, jeśli $|u_l - v_l| > Eps$, wtedy $Euclidean(u, v) > Eps$; czyli $u \notin N_{Eps}(v) \wedge v \notin N_{Eps}(u)$. Obserwacja ta prowadzi następującego twierdzenia.

Twierdzenie 4. Niech l będzie indeksem wymiaru l , gdzie $l \in [1, \dots, n]$, i D jest zbiorem wektorów posortowanych niemalejąco względem wartości l -tego wymiaru. Niech $u \in D$, f będzie takim wektorem następującym po u w D , że $f_l - u_l > Eps$, i p będzie takim wektorem poprzedzającym u w D , że $u_l - p_l > Eps$. Wtedy:

- f i wszystkie wektory następujące po f w D nie należą do $N_{Eps}(u)$,
- p i wszystkie wektory poprzedzające p w D nie należą do $N_{Eps}(u)$.

Zgodnie z twierdzeniem 2 opisany w rozdziale 4.1 twierdzenie 5 można zastosować również do wyszukiwania k sąsiedztwa.

5.2.1. Algorytm DBSCAN-PROJECTION

Zaproponowana w ramach tej pracy dyplomowej modyfikacja przedstawionego uprzednio algorytmu TI-DBSCAN stosująca rzutowanie na wymiar będzie w dalszej części pracy nazywana DBSCAN-PROJECTION. W stosunku do bazowego algorytmu zmianie ulega jedynie funkcja inicjalizująca dane, której pseudokod przedstawiłem na wydruku 10. Szarym tłem wyróżniłem fragmenty różne od odpowiedniego kodu algorytmu TI-DBSCAN.

```

function Init(D)
    pDimension := selectProjectionDimension();
    for each point p in D do
        p.clusterId := UNCLASSIFIED; p.neighborsNr := 1; p.border := Ø;
        p.dist := p[pDimension];
    end for
    sort D non-decreasingly w.r.t. p.dist;
end; // Init

```

Wydruk 10. Pseudokod funkcji algorytmu DBSCAN-PROJECTION różnych od funkcji algorytmu TI-DBSCAN

5.2.2. Algorytm k-Neighborhood-Index-Projection

W dalszej części pracy modyfikacja przedstawionego uprzednio algorytmu TI-k-Neighborhood-Index stosująca rzutowanie na wymiar będzie nazywana k-Neighborhood-Index-Projection. Podobnie jak w przypadku DBSCAN-PROJECTION zmianie uległa jedynie funkcja inicjalizująca dane, której pseudokod przedstawiłem na wydruku 11. Szarym tłem wyróżniłem fragmenty różne od odpowiedniego kodu algorytmu TI-k-Neighborhood-Index.

```

function Init(D)
    pDimension := selectProjectionDimension();
    for each point p in D do
        p.dist := p[pDimension];
    end for
    sort D non-decreasingly w.r.t. p.dist;
end; //TI-Init

```

Wydruk 11. Pseudokod funkcji algorytmu k-Neighborhood-Index-Projection różnych od funkcji algorytmu TI-k-Neighborhood-Index

5.3. Zastosowanie indeksu metrycznego w algorytmie wyszukiwania k sąsiadów

Zastosowanie indeksu metrycznego pozwala na sprawne wyszukiwanie najbliższego sąsiada danego wektora w pewnym zbiorze wektorów. W oparciu o implementację zaproponowaną w artykule [15] stworzyłem algorytm gęstościowego grupowania danych poszukujący k najbliższych sąsiadów, który wykorzystuje indeks metryczny. Algorytm ten nazwałem kNN-Index-Vp-Tree, a jego pseudokod zamieściłem na wydrukach 12, 13 i 14.

Pierwszym krokiem algorytmu jest budowa indeksu metrycznego w oparciu o zbiór wektorów D . Korzeń indeksu odnosi się do całej rozpatrywanej przestrzeni wektorów. Jego punkt obserwacyjny dzieli przestrzeń na lewą i prawą podprzestrzenie, które odpowiadają lewemu i prawemu potomkowi korzenia. Każdy kolejny węzeł drzewa nawiązuje do coraz to mniejszych podprzestrzeni. Niech najmniejsza wartość większa od mediany będzie nazywana prawym ograniczeniem, a największa wartość mniejsza od mediany będzie nazywana lewym ograniczeniem. W przedstawionym algorytmie w węźle indeksu metrycznego oprócz wektora obserwacyjnego przechowywana jest wartość mediany oraz lewe i prawe ograniczenie w celu opisania metrycznej relacji między punktem obserwacyjnym a lewą i prawą podprzestrzenią. Algorytm budowy indeksu metrycznego korzysta z funkcji *select_vp*, której celem jest obieranie lepszych niż losowe wektorów obserwacyjnych. Funkcja ta w sposób stochastyczny konstruuje zbiór kandydatów P do wektora obserwacyjnego. Następnie dla każdego wektora p zbioru kandydatów P losowo konstruowany jest podzbiór D przestrzeni, dla którego wyznaczana jest mediana oraz odchylenie standardowe. Spośród zbioru kandydatów P , wybierany jest ten o największym odchyleniu standardowym.

Kolejnym krokiem algorytmu kNN-Index-Vp-Tree jest wyszukanie k sąsiadów przy użyciu uprzednio zbudowanego indeksu dla każdego wektora zbioru zapytań Q . k sąsiadzi są przechowywani w formie kolekcji par: (odległość między wektorem p a zapytaniem q ; wektor p). Dla każdego wektora zbioru zapytań Q kolekcja k sąsiadów kNN inicjalizowana jest k parami (maksymalna wartość odległości; null), a następnie przekazywana jest wraz zapytaniem i korzeniem indeksu metrycznego do funkcji *search-kNN*. Funkcja ta przegląda w głąb indeksu metrycznego w poszukiwaniu k sąsiadów zapytania, które zapisywane jest w kolekcji kNN .

W stworzonym algorytmie proponuję dwie metody przeszukiwania indeksu metrycznego w poszukiwaniu k sąsiadów: Pierwszą, która korzysta z mediany jako kryterium przeszukiwania poddrzew indeksu. Jej pseudokod przedstawiłem na wydruku 13. Oraz drugą, która wykorzystuje ograniczenie górne i dolne w celu wyboru poddrzewa do przeszukania. Jej pseudokod zamieściłem na wydruku 14.

```

algorithm kNN-Index-Vp-Tree (D, Q, k)
    vp_tree := make-Vp-Tree(D);
    for each point q in Q do
        kNN :=  $\emptyset$ ;
        for i := 1..k do
            insert(MAX_DOUBLE, null) into kNN;
        endfor;
        search-kNN(vp_tree, q, kNN)
        insert(position of point q, kNN) into kNN-Index
    endfor;
end; // kNN-Index-Vp-Tree

function make-Vp-Tree(D)
    if D =  $\emptyset$  then
        return  $\emptyset$ ;
    endif;
    vantage_point := select-Vp(D);
    new(node);
    node.p := vantage_point;
    node. $\mu$  := mediand ∈ D distance(vantage_point, d);
    node.leftLimit := leftLimitd ∈ D distance(vantage_point, d);
    node.rightLimit := rightLimitd ∈ D distance(vantage_point, d)';
    L := {d ∈ D - vantage_point | distance(vantage_point, d) <  $\mu$ };
    R := {d ∈ D - vantage_point | distance(vantage_point, d) ≥  $\mu$ };
    node.left := make-Vp-Tree(L);
    node.right := make-Vp-Tree(R);
    return node;
end; //make-Vp-Tree

function select-Vp(D)
    P := random sample of D;
    best_spread := 0;
    best_vp := null;
    for each point p in P do
        S := random sample of D;
         $\mu$  := medians ∈ S distance(p, s);
        spread := 2nd - Moments ∈ S (distance(p, s) -  $\mu$ );
        if spread > best_spread then
            best_spread := spread;
            best_p := p;
        endif;
    endfor;
    return best_p;
end; //select-Vp

```

Wydruk 12. Pseudokod funkcji algorytmu kNN-Index-Vp-Tree

```

function search-kNN(vp_tree_p, p, kNN)
    if p is null then
        return;
    endif;
    tau := get distance to the most distant neighbor from kNN;
    dist := distance(vp_tree_p.p, p);
    leftBoundary := dis - tau;
    rightBoundary := dist + tau;
    if dist ≤ tau do
        erase the most distant neighbor from kNN;
        insert(dist, vp_tree_p.p) into kNN;
    endif;
    if rightBoundary ≥ vp_tree_p.μ then
        search-kNN(vp_tree_p.right, p, kNN);
    endif;
    if leftBoundary < vp_tree_p.μ then
        search-kNN(vp_tree_p.left, p, kNN);
    endif;
end; //search-kNN

```

Wydruk 13. Pseudokod funkcji wyszukującej k sąsiadów danego punktu w indeksie metrycznym z wykorzystaniem mediany

```

function search-kNN(vp_tree_p, p, kNN)
    if p is null then
        return;
    endif;
    tau := get distance to the most distant neighbor from kNN;
    dist := distance(vp_tree_p.p, p);
    leftBoundary := dis - tau; rightBoundary := dist + tau;
    if dist ≤ tau do
        erase the most distant neighbor from kNN;
        insert(dist, vp_tree_p.p) into kNN;
    endif;
    if leftBoundary ≤ vp_tree_p.leftLimit then
        if rightBoundary ≥ vp_tree_p.rightLimit then
            leftBuffer := vp_tree_p.leftLimit - leftBoundary;
            rightBuffer := rightBoundary - tree.rightLimit;
            if leftBuffer < rightBuffer do
                search-kNN(vp_tree_p.left, p, kNN);
                search-kNN(vp_tree_p.right, p, kNN);
            else
                search-kNN(vp_tree_p.right, p, kNN);
                search-kNN(vp_tree_p.left, p, kNN);
            endif;
        else
            search-kNN(vp_tree_p.left, p, kNN);
        endif;
    else
        if rightBoundary ≥ vp_tree_p.rightLimit then
            search-kNN(vp_tree_p.right, p, kNN);
        endif;
    endif;
end; //search-kNN

```

Wydruk 14. Pseudokod funkcji wyszukującej k sąsiadów danego punktu w indeksie metrycznym z wykorzystaniem prawego i lewego ograniczenia

6. Badania eksperymentalne

W zrealizowanych przeze mnie eksperymentach badałem wydajność wyszukiwania k sąsiedztwa / k sąsiadów oraz sąsiedztwa epsilonowego w zależności od implementacji, przyjętej miary podobieństwa oraz zastosowanej metody szacowania odległości. W rozdziałach od 6.1. do 6.5. zamieściłem rezultaty badań algorytmów wykorzystujących miarę euklidesową jako miarę podobieństwa, a w rozdziałach od 6.6. do 6.8. zamieściłem wyniki badań algorytmów wykorzystujących miarę kosinusową. W celu przeprowadzenia eksperymentów zaprojektowałem i zimplementowałem program służący do analizy algorytmów.

Program ten został napisany w języku C++, jest przeznaczony dla systemu operacyjnego Microsoft Windows i implementuje wszystkie algorytmy opisane w poprzednich rozdziałach. Stworzona aplikacja wykonywana jest w trybie wsadowym. Jej wejście stanowią pliki parametrów uruchomienia algorytmu. Każdy z nich definiuje wykonanie danego algorytmu na danym zbiorze danych. Rezultat pojedynczego wykonania algorytmu oraz raport z jego uruchomienia zapisywane są do pliku tekstowego. W trakcie jednego uruchomienia aplikacji możliwe jest sekwencyjne wykonanie wielu uruchomień algorytmów, dlatego aplikacja na bieżąco tworzy plik zbiorczego raportu w formacie CSV, w którym każdy rekord odpowiada raportowi z pojedynczego wykonania algorytmu. Dokładny opis uruchamiania aplikacji, możliwości jej konfiguracji oraz opis plików parametrów znajduje się w dodatku A.

Eksperymenty przeprowadziłem w środowisku Windows 7 x64 z procesorem Intel® i7™ 950 z dostępną pamięcią RAM równą 6GB. Aplikacja, którą posłużyłem się do zebrania wyników eksperymentów, została skompilowana dla środowiska WIN32 z podniesioną flagą LARGEADDRESSAWARE, która umożliwia użycie do 3GB pamięci wirtualnej. Eksperymenty uruchamiałem jako proces o priorytecie ‘Wysoki’, aby uniknąć wywłaszczenia procesu z rdzenia, zmniejszając tym samym ryzyko losowego zakłócenia wyników.

W trakcie pierwszych eksperymentów zauważylem, że kolejne uruchomienia danego algorytmu z tymi samymi parametrami, nazwijmy je serią, wykonują się w różnym czasie. Ze względu na charakter wspomnianych różnic wyszczególnię dwa przypadki odchyleń. Pierwszy, w którym różnice czasów wykonania algorytmów w danej serii są niewielkie oraz drugi, w którym owe różnice są znaczące. W pierwszym przypadku powodem odchyleń były

losowe zakłócenia. W drugim przypadku zauważylem pewną prawidłowość, według której pierwsze uruchomienie algorytmu w serii trwa najdłużej z wszystkich, kolejne trochę krócej a różnice czasów wykonania następnych uruchomień odpowiadają tym z przypadku pierwszego.

Dzieje się tak, ponieważ cache procesora (w moim przypadku 8MB, L3) z każdym wykonaniem algorytmu w serii działa sprawniej do chwili osiągnięcia pewnej sprawności granicznej dla danej serii. W rezultacie, czas wykonania algorytmu skraca się aż do momentu, gdy różnice trwania kolejnych wykonień przypominają te z przypadku pierwszego. Aby zmniejszyć wpływ wyżej opisanych zjawisk na zbieranie wyników czasowych poszczególnych kroków algorytmów pomiar dokonywany jest w następujący sposób: spośród wszystkich uruchomień algorytmu w serii odrzucane są te, których czas wykonania był najdłuższy i najkrótszy, a z pozostałych wartości obliczany jest średni czas wykonania każdego z kroków algorytmu. W prezentowanych dalej wynikach eksperymentów, czas wykonania algorytmu jest sumą średnich czasów wykonania każdego z jego kroków. W każdej tabeli prezentującej czasy trwania algorytmów zamieszczono również średnie czasy kroków nań składających się.

Słownik opisu rezultatów badań

W celu zmniejszenia szerokości tabel zawierających rezultaty badań, kolumny zostały opisane za pomocą skróconych wyrażeń, których rozwinięcia zamieściłem w tab. 6.

Tab. 6. Słownik opisów rezultatów badań

| Skrcone wyrażenie | Pełne wyrażenie | Znaczenie |
|--------------------------|-------------------------------------|--|
| bud. ind. | budowa indeksu | czas budowy indeksu [s] |
| grup. | grupowanie | czas wykonania grupowania [s] |
| l. p. | liczba punktów | liczność zbioru, na którym wykonywany jest algorytm |
| norm. | normalizacja | czas normalizacji wektorów zbioru danych [s] |
| obl. odł. | obliczanie odległości | czas obliczania odległości wektorów do wektorów referencyjnych lub czas wykonania rzutowania wektorów na dany wymiar [s] |
| sort. | sortowanie | czas sortowania zbioru danych [s] |
| wysz. sąsiad. | wyszukiwanie sąsiadów/sąsiedztwa | czas wyszukiwania sąsiadów/sąsiedztwa [s] |
| wyk. alg. | wykonanie algorytmu | czas wykonania algorytmu [s] |
| - | brak danych | rezultat eksperymentu nie został osiągnięty w czasie krótszym niż 3h lub z powodu wyczerpania pamięci |

6.1. Dane testowe

W eksperymetach wykorzystałem zbiory danych stosowane w dziedzinowej literaturze. Ich odmienne charakterystyki pozwoliły na sprawdzenie działania algorytmów w różnych warunkach. Zamieszczone poniżej opisy dotyczą zbiorów, które wykorzystałem do przeprowadzenia badań.

Covtype [2] to zbiór udostępniany przez US Forest Service. Jego pełna nazwa to Forest Cover Type. Baza ta zawiera informacje o gatunkach drzew w amerykańskich lasach. Zbiór posiada 581012 rekordów w formacie gęstym o 55 atrybutach. 10 pierwszych atrybutów to wartości zmiennych ilościowych, kolejne 44 atrybuty posiadają wartości 0 lub 1, z kolei ostatni determinuje jeden z siedmiu gatunków drzew. Należy zwrócić uwagę, że każdy z rekordów spośród atrybutów binarnych przyjmuje wartość 1 dokładnie dwa razy, a wartość zero 42 razy. W przypadku 10 pierwszych atrybutów zróżnicowanie wartości jest znacznie większe, z których pierwsze dwa posiadają najszerzą dziedziną (ponad 7000), a pozostałe charakteryzuje znacznie mniejsza różnorodność i zakres.

KDD-Cup98 [13] jest zbiorem opublikowanym podczas Drugiego Międzynarodowego Konkursu Odkrywania Wiedzy i Eksploracji Danych, który miał miejsce podczas KDD-98, Czwartej Międzynarodowej Konferencji Odkrywania Wiedzy i Eksploracji Danych w 1998 roku. Zbiór posiada 96367 rekordów w formacie gęstym o 56 atrybutach, o wartościach będącymi liczbami naturalnymi. Najszerzą dziedzinę [0; 6000] posiadają dwa atrybuty, najwyższą dziedzinę [0; 13] również tylko dwa atrybuty, natomiast aż 44 atrybuty posiadają taką samą dziedzinę [0; 99].

Karypis_sport to zbiór danych wchodzący w skład bazy zbiorów [6] udostępnianych na stronie profesora George Karypis z Department of Computer Science & Engineering, University of Minnesota. Zbiór karypis_sport zawiera 8580 rekordów w formacie rzadkim o 126373 atrybutach przyjmujących wartości będące liczbami naturalnymi. Baza ta zawiera informacje o dokumentach dotyczących sportu. Średnia niezerowych wartości atrybutów zbioru wynosi 1,58, natomiast najwyższa wartość to 129. Mimo, że punkty posiadają 126373 atrybutów, to średnio 129 z nich jest niezerowych. Punkt o największej liczbie niezerowych atrybutów posiada ich 1174. Liczba punktów posiadających nie więcej niezerowych atrybutów niż 56 wynosi 1478.

Karypis_review jest zbiorem danych wchodzącym w ten sam skład bazy zbiorów co karypis_sport [6]. Zawiera on 4069 rekordów w formacie rzadkim o 126373 atrybutach

przyjmujących wartości będące liczbami naturalnymi. Baza ta skupia informacje o dokumentach dotyczących recenzji. Średnia niezerowych wartości atrybutów zbioru wynosi 1,49, natomiast najwyższa wartość to 131. Mimo, że punkty posiadają 126373 atrybutów, to średnio 191 z nich jest niezerowych. Punkt o największej liczbie niezerowych atrybutów posiada ich 1385. Liczba punktów posiadających nie więcej niezerowych atrybutów niż 56 wynosi 254.

6.2. Badania algorytmu k-Neighborhood-Index

W niniejszym rozdziale opisałem rezultaty badań algorytmów wyznaczających k sąsiedztwo, które stosują nierówność trójkąta w ujęciu klasycznym lub rzutowanie w celu oszacowania odległości między punktami. W każdym z kolejnych podrozdziałów testowałem jedną z odmian algorytmu lub porównywałem wcześniej zbadane. W ostatnim podrozdziale dokonałem porównania zbadanych algorytmów.

6.2.1. Badania algorytmu TI-k-Neighborhood-Index

W pierwszym z kolejnych podrozdziałów skupiłem się na badaniu czasu wykonania *TI-k-Neighborhood-Index* w zależności od implementacji algorytmu oraz w zależności od implementacji punktu. Następnie, w oparciu o uzyskane rezultaty eksperymentów zbadałem wpływ wyboru punktu referencyjnego na wydajność algorytmu.

6.2.1.1. Implementacja algorytmu

Na przykładowych zbiorach danych przetestowałem dwie wersje algorytmu *TI-k-Neighborhood-Index*: wersję korzystającą ze struktury indeksu zbudowanego z iteratorów oraz wersję opartą wyłącznie na oryginalnym zbiorze danych. W przypadku pierwszej implementacji sortowaniu zostaje poddany indeks, który następnie używany jest w celu dostępu do zbioru danych. W drugim przypadku sortowany jest zbiór danych i żaden indeks nie jest wykorzystywany w dostępie do jego elementów. W tab. 7 zamieściłem czasy wykonania obu

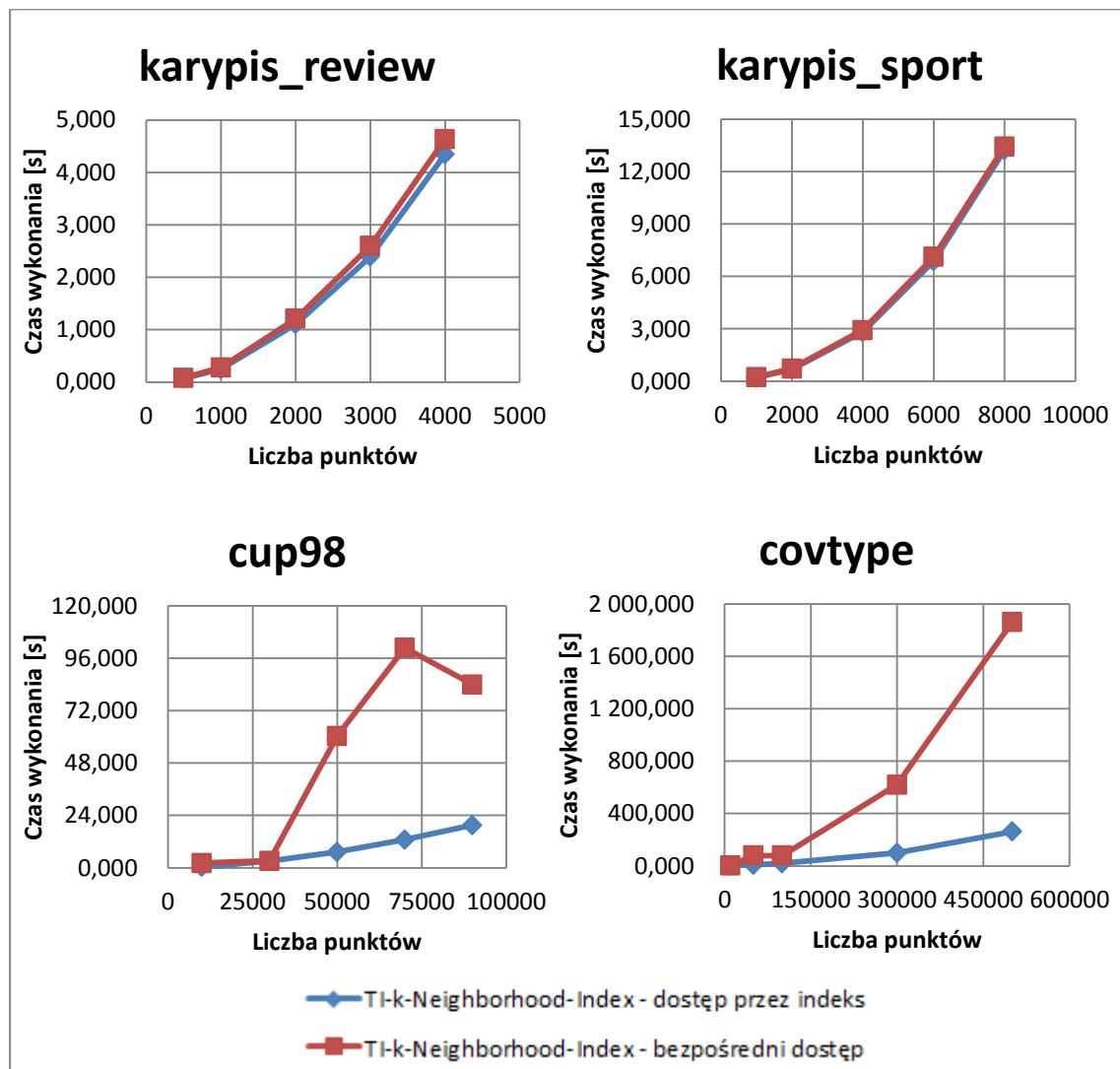
implementacji algorytmów wraz ze składającymi się na nie operacjami. Rys. 13 przedstawia porównanie wydajności badanych implementacji algorytmu *Tik-k-Neighborhood-Index*.

Analiza wyników z tab. 7 pozwala stwierdzić, że koszt budowy indeksu dla implementacji z indeksem jest pomijalnie mały. Kolejną ważną cechą jest porównywalny dla obu implementacji czas obliczania odległości do punktu referencyjnego. Jednakże, w porównaniu z implementacją bez indeksu implementacja z indeksem radzi sobie z wyznaczaniem k sąsiedztwa tym lepiej im więcej rekordów posiada zbiór, w którym wyznaczane jest k sąsiedztwo. Największa różnica w wydajności algorytmu wystąpiła przy 50000 punktów zbioru covtype. Dla tego przypadku implementacja z bezpośrednim dostępem do danych wykonuje się blisko 7 razy wolniej niż implementacja korzystająca z indeksu.

Tab. 7. Porównanie wydajności implementacji algorytmu TI-k-Neighborhood-Index w zależności od zastosowanej metody dostępu do danych przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

| zbior | l.p. | TI-k-Neighborhood-Index dostęp przez indeks | | | | | TI-k-Neighborhood-Index bezpośredni dostęp | | | | |
|----------------|--------|---|---------|-------|--------|-------|--|---------|--------|-----------|--|
| | | wyk. | wysz. | bud. | obl. | sort. | wyk. | wysz. | obl. | sort. | |
| | | alg. | sąsiad. | ind. | odl. | | alg. | sąsiad. | odl. | | |
| kartpis_sport | 1000 | 0,204 | 0,204 | 0,000 | 0,000 | 0,000 | 0,235 | 0,206 | 0,004 | 0,025 | |
| | 2000 | 0,678 | 0,674 | 0,000 | 0,004 | 0,000 | 0,729 | 0,660 | 0,004 | 0,065 | |
| | 4000 | 2,824 | 2,813 | 0,000 | 0,010 | 0,001 | 2,932 | 2,755 | 0,010 | 0,166 | |
| | 6000 | 6,887 | 6,870 | 0,000 | 0,016 | 0,001 | 7,122 | 6,570 | 0,015 | 0,537 | |
| | 8000 | 13,228 | 13,202 | 0,000 | 0,023 | 0,002 | 13,440 | 12,736 | 0,024 | 0,680 | |
| | 500 | 0,068 | 0,066 | 0,000 | 0,002 | 0,000 | 0,079 | 0,066 | 0,002 | 0,012 | |
| kartpis_review | 1000 | 0,249 | 0,246 | 0,000 | 0,003 | 0,000 | 0,279 | 0,247 | 0,003 | 0,029 | |
| | 2000 | 1,105 | 1,099 | 0,000 | 0,006 | 0,000 | 1,208 | 1,106 | 0,006 | 0,096 | |
| | 3000 | 2,394 | 2,385 | 0,000 | 0,008 | 0,001 | 2,598 | 2,380 | 0,008 | 0,210 | |
| | 4000 | 4,346 | 4,334 | 0,000 | 0,011 | 0,001 | 4,637 | 4,303 | 0,012 | 0,322 | |
| | 10000 | 0,547 | 0,391 | 0,000 | 0,153 | 0,003 | 1,364 | 0,184 | 0,058 | 1,122 | |
| | 50000 | 7,750 | 5,400 | 0,001 | 2,333 | 0,015 | 78,517 | 4,147 | 2,089 | 72,281 | |
| covtype | 100000 | 19,989 | 15,052 | 0,002 | 4,900 | 0,035 | 79,311 | 11,357 | 4,636 | 63,318 | |
| | 300000 | 99,592 | 83,380 | 0,004 | 16,079 | 0,130 | 619,451 | 50,367 | 15,277 | 553,806 | |
| | 500000 | 261,458 | 234,340 | 0,005 | 26,861 | 0,252 | 1860,230 | 120,760 | 25,600 | 1 713,870 | |
| | 10000 | 0,392 | 0,272 | 0,000 | 0,117 | 0,003 | 2,241 | 0,140 | 0,062 | 2,039 | |
| | 30000 | 3,073 | 2,197 | 0,000 | 0,865 | 0,010 | 3,224 | 1,191 | 0,530 | 1,503 | |
| | 50000 | 7,291 | 5,371 | 0,001 | 1,900 | 0,019 | 60,378 | 3,198 | 1,555 | 55,625 | |
| cup98 | 70000 | 12,886 | 9,563 | 0,001 | 3,293 | 0,029 | 100,792 | 5,673 | 2,772 | 92,347 | |
| | 90000 | 19,547 | 15,316 | 0,002 | 4,185 | 0,044 | 83,934 | 9,064 | 4,649 | 70,221 | |

Kluczowym krokiem mającym wpływ na powstałą różnicę jest sortowanie punktów. Obie implementacje do sortowania obiektów wykorzystują standardową dla języka C++ funkcję sortowania wektorów, tj. `std::sort`, która oparta jest na algorytmie quicksort. Funkcja sortowania biblioteki standardowej korzysta z operacji `swap` zamieniającej miejscami elementy w wektorze, która wywołuje konstruktor kopiący obiektu będącego elementem sortowanego wektora. W przypadku, gdy głęboko kopowany jest obiekt klasy `Punkt`, uruchamiane są konstruktory kopiące wszystkich obiektów będących jego polami, w szczególności kolekcji przechowujących wartości wymiarów.



Rys. 13. Porównanie wydajności implementacji algorytmu TI-k-Neighborhood-Index Index w zależności od zastosowanej metody dostępu do danych przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

Dlatego wykonanie konstruktora kopiącego iteratora jest szybsze niż wykonanie konstruktora kopiącego obiektu punktu. Im większy zbiór jest sortowany tym częściej funkcja *swap* jest wywoływana, tym samym częściej wykonywany jest konstruktor kopiący. W skutek tego, czas sortowania zbioru obiektów punktu jest dłuższy niż czas sortowania zbioru iteratorów. Zatem mimo, że dostęp do nieposortowanych danych przez posortowany zbiór iteratorów jest wolniejszy niż bezpośredni dostęp do posortowanego zbioru, to czas bezpośredniego sortowania zbioru danych jest na tyle duży, że różnice czasowe wynikające ze sposobu dostępu do danych stają się pomijalne. W kolejnych rozważaniach będę się odnosił do algorytmu *TI-k-Neighborhood-Index* jako do implementacji korzystającej z indeksu iteratorów.

6.2.1.2. Implementacja struktury punktu

W literaturze dziedzinowej punkt przestrzeni n wymiarowej może zostać zapisany jako lista wartości w jednej z dwóch reprezentacji:

- *gęstej*, w której każda kolejna wartość w liście odpowiada wartościom kolejnych wymiarów przestrzeni,
- *rzadkiej*, w której każda kolejna nieparzysta wartość w liście odpowiada numerowi wymiaru przestrzeni, a parzysta wartość wartości tego wymiaru.

Przykładowo w przestrzeni czterowymiarowej punkt P o współrzędnych $(w, x, y, z) = (3, 0, 0, 5)$ zostanie zapisany w reprezentacji gęstej jako [3 0 0 5], a w reprezentacji rzadkiej jako [1 3 4 5], gdzie kolejne liczby oznaczają: numer pierwszego wymiaru niezerowego, wartość pierwszego wymiaru niezerowego, numer drugiego wymiaru niezerowego, wartość drugiego wymiaru niezerowego.

Początkowo naturalnym podejściem do implementacji struktury punktu wydawało mi się przechowywanie wartości wymiarów w postaci tablicy lub wektora o liczbie elementów równej liczbie wymiarów przestrzeni danych. Pierwsze eksperymenty z danymi tekstowymi pokazały, że przy zastosowaniu wspomnianego sposobu postępowania implementacja zbioru punktów szybko wyczerpywała pamięć RAM maszyny, na której dokonywano eksperymentów. Zachowanie to wynikało z dużej liczby wymiarów przestrzeni danych. Jeden punkt przestrzeni danych tekstowych (10^5 wymiarowej) zajmował tyle pamięci co $5 \cdot 10^4$ punktów przestrzeni dwuwymiarowej.

Na szczęście, przechowywanie punktów przestrzeni danych tekstowych można ulepszyć w oparciu o ich cechy charakterystyczne. Kluczową własnością danych tekstowych jest duża liczba wymiarów, których wartość jest równa 0. Wartość ta nie wnosi żadnej informacji w procesie wyznaczania odległości między dwoma punktami, dlatego postanowiłem jej nie przechowywać. W tym celu zaimplementowałem rzadką reprezentację punktu w postaci listy par \langle numer wymiaru, wartość wymiaru \rangle posortowanej względem numeru wymiaru.

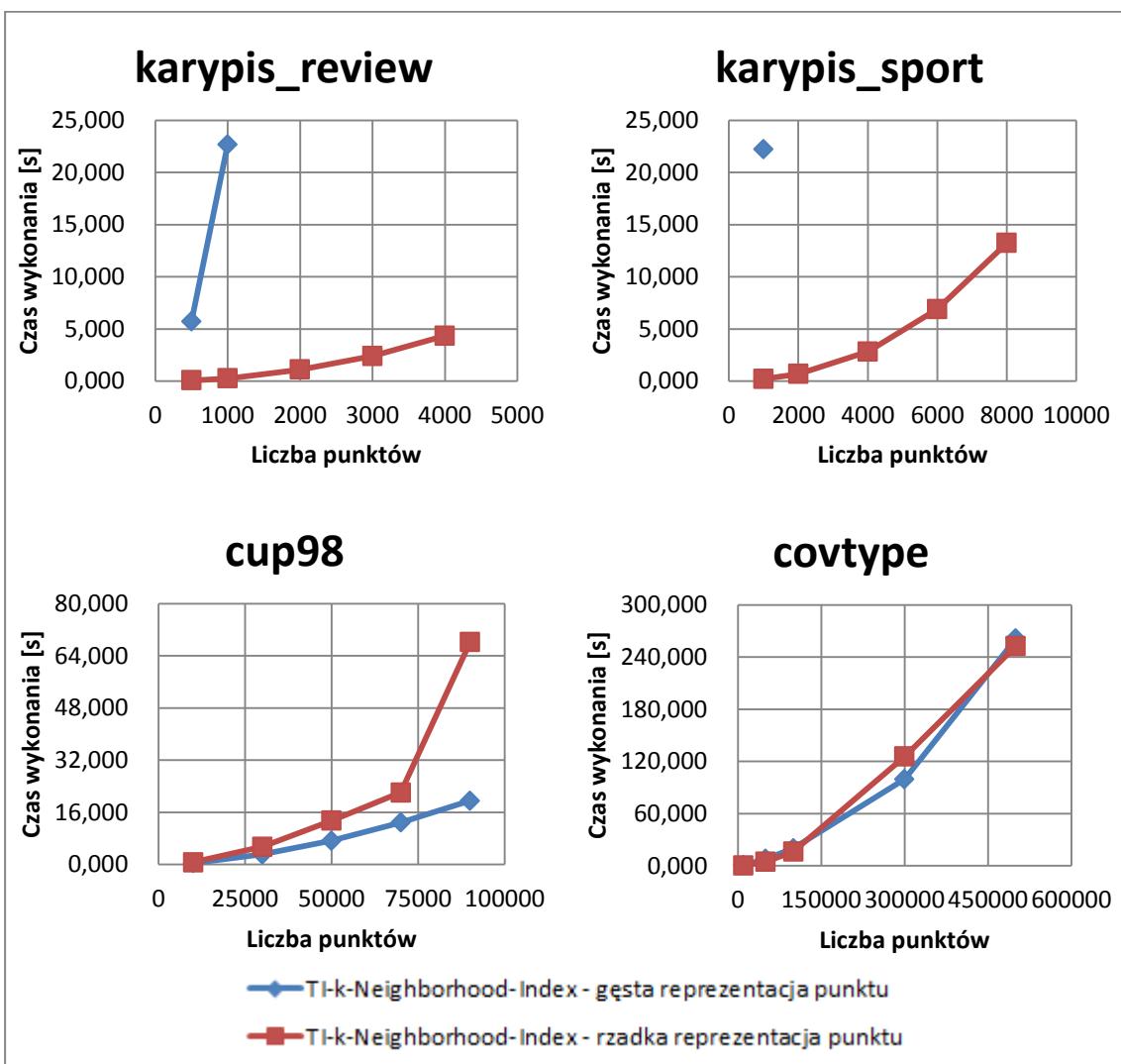
W tab. 8 oraz na rys. 14 zmieściłem wyniki uruchomień algorytmu *TI-k-Neighborhood-Index* z użyciem implementacji zarówno gęstego jak i rzadkiego punktu. Rezultaty obserwacji znajdujące się w tab. 8 pozwalają stwierdzić, że dla zbiorów cup98 i covtype, uruchomienia korzystające z implementacji punktu gęstego wykonują się szybciej niż uruchomienia posługujące się implementacją punktu rzadkiego. Odwrotne zjawisko można zauważać dla danych tekstowych, tj. zbiorów karypis sport i karypis review, w których przypadku, uruchomienia korzystające z implementacji punktu rzadkiego wykonują się o dwa rzędy wielkości szybciej niż uruchomienia korzystające z implementacji punktu gęstego. Niestety, z powodu wyczerpania pamięci RAM nie można było uzyskać rezultatów badań dla implementacji gęstego punktu na zbiorach danych tekstowych gdy liczba punktów przewyższała 1000. Na podstawie przeprowadzonych eksperymentów wysnułem następujące wnioski:

- w porównaniu z implementacją gęstego punktu, stosowanie implementacji rzadkiego punktu, w uruchomieniach algorytmu pracującego na danych tekstowych, które są rzadkie przyspiesza jego wykonanie oraz zmniejsza wielkość zajętej pamięci operacyjnej;
- w porównaniu z implementacją rzadkiego punktu, stosowanie implementacji gęstego punktu, w uruchomieniach algorytmu działającego na danych gęstych o niewielkim wymiarze, przyspiesza jego wykonanie.

W dalszej części pracy badania algorytmów *TI-k-Neighborhood-Index* pracujących na danych tekstowych będą wykonywane z użyciem implementacji punktu rzadkiego, a działające z danymi gęstymi o niewielkim wymiarze będą wykonywane z zastosowaniem implementacji punktu gęstego.

Tab. 8. Porównanie wydajności algorytmu TI-k-Neighborhood-Index w zależności od implementacji punktu przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

| zbior | I. p. | TI-k-Neighborhood-Index gęsta reprezentacja | | | | | TI-k-Neighborhood-Index rzadka reprezentacja | | | | |
|----------------|--------|---|------------------|--------------|--------------|-------|--|------------------|--------------|--------------|-------|
| | | punktu | | | | | punktu | | | | |
| | | wyk. alg. | wysz. sąsiad. | bud. ind. | obl. odl. | sort. | wyk. alg. | wysz. sąsiad. | bud. ind. | obl. odl. | sort. |
| karypis_sport | 1000 | 22,208 | 21,959 | 0,000 | 0,249 | 0,000 | 0,204 | 0,204 | 0,000 | 0,000 | 0,000 |
| | 2000 | - | - | - | - | - | 0,678 | 0,674 | 0,000 | 0,004 | 0,000 |
| | 4000 | - | - | - | - | - | 2,824 | 2,813 | 0,000 | 0,010 | 0,001 |
| | 6000 | - | - | - | - | - | 6,887 | 6,870 | 0,000 | 0,016 | 0,001 |
| | 8000 | - | - | - | - | - | 13,228 | 13,202 | 0,000 | 0,023 | 0,002 |
| | 500 | 5,710 | 5,585 | 0,000 | 0,125 | 0,000 | 0,068 | 0,066 | 0,000 | 0,002 | 0,000 |
| karypis_review | 1000 | 22,677 | 22,428 | 0,000 | 0,249 | 0,000 | 0,249 | 0,246 | 0,000 | 0,003 | 0,000 |
| | 2000 | - | - | - | - | - | 1,105 | 1,099 | 0,000 | 0,006 | 0,000 |
| | 3000 | - | - | - | - | - | 2,394 | 2,385 | 0,000 | 0,008 | 0,001 |
| | 4000 | - | - | - | - | - | 4,346 | 4,334 | 0,000 | 0,011 | 0,001 |
| | 10000 | 0,547 | 0,391 | 0,000 | 0,153 | 0,003 | 0,348 | 0,286 | 0,000 | 0,062 | 0,000 |
| | 50000 | 7,750 | 5,400 | 0,001 | 2,333 | 0,015 | 4,394 | 3,417 | 0,000 | 0,962 | 0,016 |
| covtype | 100000 | 19,989 | 15,052 | 0,002 | 4,900 | 0,035 | 16,406 | 12,995 | 0,000 | 3,380 | 0,031 |
| | 300000 | 99,592 | 83,380 | 0,004 | 16,079 | 0,130 | 125,538 | 115,939 | 0,005 | 9,453 | 0,141 |
| | 500000 | 261,458 | 234,340 | 0,005 | 26,861 | 0,252 | 252,783 | 235,820 | 0,016 | 16,682 | 0,265 |
| | 10000 | 0,392 | 0,272 | 0,000 | 0,117 | 0,003 | 0,670 | 0,608 | 0,000 | 0,062 | 0,000 |
| | 30000 | 3,073 | 2,197 | 0,000 | 0,865 | 0,010 | 5,403 | 4,763 | 0,000 | 0,629 | 0,010 |
| | 50000 | 7,291 | 5,371 | 0,001 | 1,900 | 0,019 | 13,442 | 12,054 | 0,000 | 1,367 | 0,021 |
| cup98 | 70000 | 12,886 | 9,563 | 0,001 | 3,293 | 0,029 | 22,068 | 21,028 | 0,000 | 1,009 | 0,031 |
| | 90000 | 19,547 | 15,316 | 0,002 | 4,185 | 0,044 | 68,218 | 65,275 | 0,000 | 2,891 | 0,052 |



Rys. 14. Porównanie wydajności algorytmu TI-k-Neighborhood-Index w zależności od implementacji punktu przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

6.2.1.3. Wybór punktu referencyjnego

W poszukiwaniu właściwego punktu referencyjnego dla algorytmu *TI-k-Neighborhood-Index* przeprowadziłem badania z różnymi ich przykładami. W swoich eksperymentach skupiłem się na punktach skrajnych, takich jak:

- punkt maksymalny, którego każdy z wymiarów przyjmuje maksymalną wartość z dziedziny danego wymiaru - oznaczany dalej jako [max];

- punkt minimalny, którego każdy z wymiarów przyjmuje minimalną wartość z dziedziny danego wymiaru - oznaczany dalej jako [min];
- punkt, którego każdy parzysty wymiar przyjmuje minimalną wartość z dziedziny danego wymiaru, a każdy nieparzysty wymiar przyjmuje maksymalną wartość z dziedziny danego wymiaru - oznaczany dalej jako [max_min];

W rozważaniach uwzględniałem również punkt losowy, którego każdy z wymiarów przyjmuje losową wartość z dziedziny danego wymiaru nie większą niż wartość maksymalna z dziedziny danego wymiaru - oznaczany dalej jako [rand].

W tab. 9 i tab. 10 zamieściłem czasy uruchomień algorytmu *TI-k-Neighborhood-Index*.

Na rys. 15 znajdują się wykresy czasu wykonania algorytmu w funkcji liczby punktów.

Tab. 9. Porównanie wydajności TI-k-Neighborhood-Index w zależności od punktu referencyjnego [min] lub [max_min] przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy poszukiwań k=5 sąsiedztwa w przykładowych zbiorach dla 10% losowo wybranych punktów

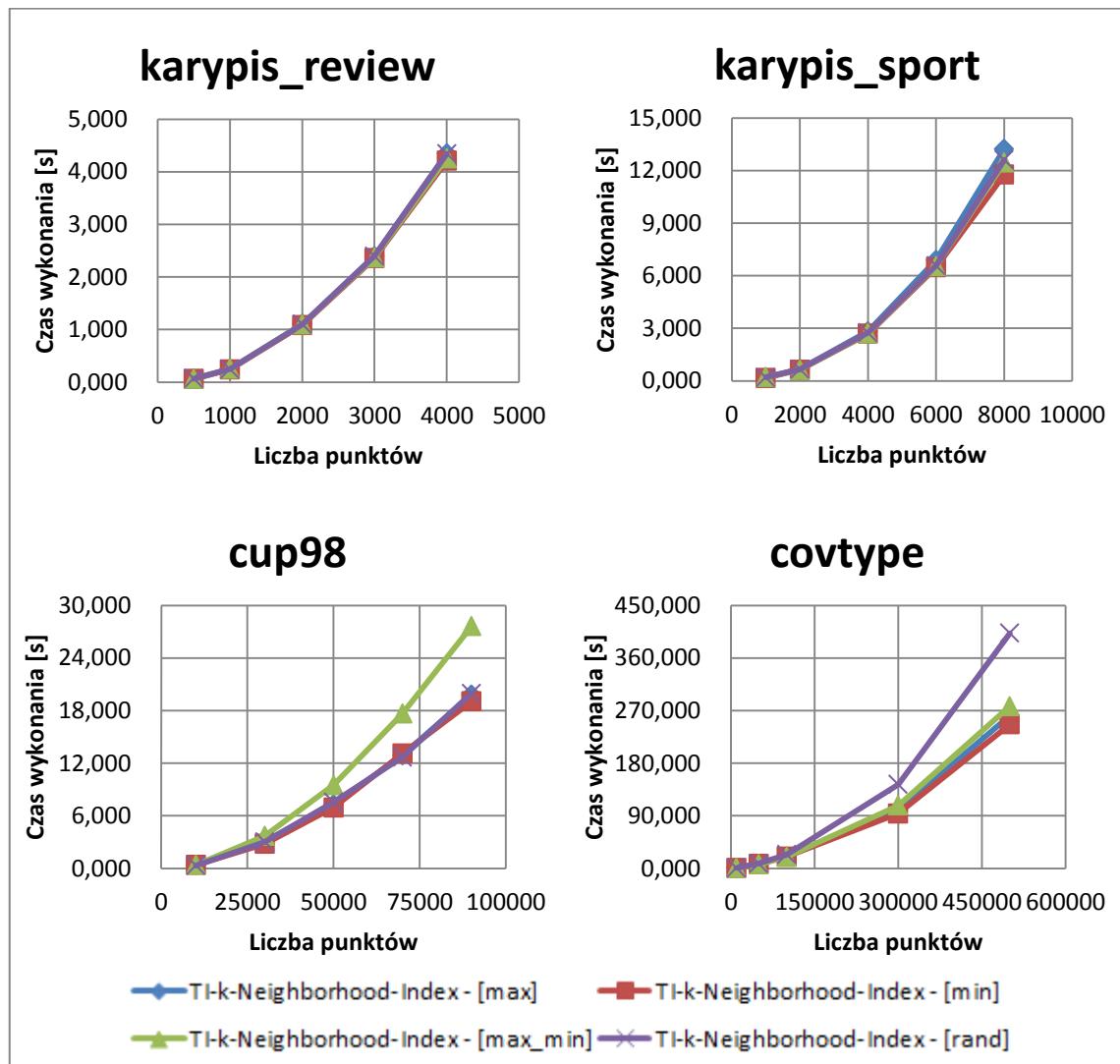
| zbior | I. p. | TI-k-Neighborhood-Index [min] | | | | | TI-k-Neighborhood-Index [max_min] | | | | |
|----------------|--------|-------------------------------|---------|-------|--------|-------|-----------------------------------|---------|-------|--------|-------|
| | | wyk. | wysz. | bud. | obl. | sort. | wyk. | wysz. | bud. | obl. | sort. |
| | | alg. | sąsiad. | ind. | odl. | | alg. | sąsiad. | ind. | odl. | |
| karypis_sport | 1000 | 0,171 | 0,169 | 0,000 | 0,002 | 0,000 | 0,208 | 0,203 | 0,000 | 0,005 | 0,000 |
| | 2000 | 0,636 | 0,632 | 0,000 | 0,004 | 0,000 | 0,634 | 0,634 | 0,000 | 0,000 | 0,000 |
| | 4000 | 2,684 | 2,675 | 0,000 | 0,008 | 0,001 | 2,719 | 2,709 | 0,000 | 0,010 | 0,000 |
| | 6000 | 6,470 | 6,455 | 0,000 | 0,013 | 0,002 | 6,537 | 6,526 | 0,000 | 0,011 | 0,000 |
| | 8000 | 11,770 | 11,751 | 0,000 | 0,017 | 0,002 | 12,455 | 12,439 | 0,000 | 0,016 | 0,000 |
| karypis_review | 500 | 0,062 | 0,061 | 0,000 | 0,001 | 0,000 | 0,062 | 0,062 | 0,000 | 0,000 | 0,000 |
| | 1000 | 0,241 | 0,239 | 0,000 | 0,002 | 0,000 | 0,250 | 0,250 | 0,000 | 0,000 | 0,000 |
| | 2000 | 1,084 | 1,079 | 0,000 | 0,005 | 0,000 | 1,097 | 1,092 | 0,000 | 0,005 | 0,000 |
| | 3000 | 2,356 | 2,348 | 0,000 | 0,007 | 0,001 | 2,371 | 2,366 | 0,000 | 0,005 | 0,000 |
| | 4000 | 4,209 | 4,198 | 0,000 | 0,010 | 0,001 | 4,253 | 4,243 | 0,000 | 0,010 | 0,000 |
| covtype | 10000 | 0,551 | 0,370 | 0,000 | 0,179 | 0,002 | 0,416 | 0,333 | 0,000 | 0,083 | 0,000 |
| | 50000 | 7,804 | 5,526 | 0,001 | 2,262 | 0,015 | 6,775 | 5,086 | 0,000 | 1,674 | 0,015 |
| | 100000 | 19,962 | 15,025 | 0,001 | 4,901 | 0,035 | 19,568 | 15,262 | 0,000 | 4,274 | 0,031 |
| | 300000 | 93,868 | 78,047 | 0,003 | 15,686 | 0,132 | 107,443 | 91,785 | 0,000 | 15,527 | 0,130 |
| | 500000 | 246,482 | 219,916 | 0,005 | 26,312 | 0,249 | 277,321 | 250,843 | 0,000 | 26,229 | 0,249 |
| cup98 | 10000 | 0,388 | 0,259 | 0,000 | 0,126 | 0,003 | 0,389 | 0,296 | 0,000 | 0,093 | 0,000 |
| | 30000 | 2,785 | 2,105 | 0,001 | 0,669 | 0,010 | 3,697 | 3,011 | 0,000 | 0,671 | 0,016 |
| | 50000 | 6,942 | 5,287 | 0,001 | 1,635 | 0,019 | 9,486 | 7,972 | 0,000 | 1,498 | 0,016 |
| | 70000 | 13,095 | 9,933 | 0,001 | 3,132 | 0,029 | 17,643 | 14,809 | 0,000 | 2,803 | 0,031 |
| | 90000 | 19,041 | 14,683 | 0,001 | 4,316 | 0,041 | 27,622 | 23,228 | 0,005 | 4,347 | 0,042 |

Tab. 10. Porównanie wydajności algorytmu TI-k-Neighborhood-Index w zależności od wybranego punktu referencyjnego [rand] lub [max] przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

| zbior | l. p. | TI-k-Neighborhood-Index [rand] | | | | | TI-k-Neighborhood-Index [max] | | | | |
|----------------|--------|--------------------------------|---------|-------|--------|-------|-------------------------------|---------|-------|--------|-------|
| | | wyk. | wysz. | bud. | obl. | sort. | wyk. | wysz. | bud. | obl. | sort. |
| | | alg. | sąsiad. | ind. | odl. | | alg. | sąsiad. | ind. | odl. | |
| karypis_sport | 1000 | 0,199 | 0,195 | 0,000 | 0,004 | 0,000 | 0,204 | 0,204 | 0,000 | 0,000 | 0,000 |
| | 2000 | 0,664 | 0,660 | 0,000 | 0,004 | 0,000 | 0,678 | 0,674 | 0,000 | 0,004 | 0,000 |
| | 4000 | 2,753 | 2,743 | 0,000 | 0,009 | 0,001 | 2,824 | 2,813 | 0,000 | 0,010 | 0,001 |
| | 6000 | 6,576 | 6,560 | 0,000 | 0,014 | 0,002 | 6,887 | 6,870 | 0,000 | 0,016 | 0,001 |
| | 8000 | 12,612 | 12,587 | 0,000 | 0,023 | 0,002 | 13,228 | 13,202 | 0,000 | 0,023 | 0,002 |
| | 500 | 0,066 | 0,064 | 0,000 | 0,002 | 0,000 | 0,068 | 0,066 | 0,000 | 0,002 | 0,000 |
| karypis_review | 1000 | 0,247 | 0,244 | 0,000 | 0,003 | 0,000 | 0,249 | 0,246 | 0,000 | 0,003 | 0,000 |
| | 2000 | 1,098 | 1,092 | 0,000 | 0,006 | 0,000 | 1,105 | 1,099 | 0,000 | 0,006 | 0,000 |
| | 3000 | 2,394 | 2,385 | 0,000 | 0,008 | 0,001 | 2,394 | 2,385 | 0,000 | 0,008 | 0,001 |
| | 4000 | 4,340 | 4,328 | 0,000 | 0,011 | 0,001 | 4,346 | 4,334 | 0,000 | 0,011 | 0,001 |
| | 10000 | 0,545 | 0,433 | 0,000 | 0,109 | 0,003 | 0,547 | 0,391 | 0,000 | 0,153 | 0,003 |
| | 50000 | 8,104 | 6,078 | 0,001 | 2,009 | 0,016 | 7,750 | 5,400 | 0,001 | 2,333 | 0,015 |
| covtype | 100000 | 23,603 | 18,911 | 0,001 | 4,654 | 0,036 | 19,989 | 15,052 | 0,002 | 4,900 | 0,035 |
| | 300000 | 143,614 | 127,733 | 0,000 | 15,746 | 0,135 | 99,592 | 83,380 | 0,004 | 16,079 | 0,130 |
| | 500000 | 402,611 | 376,226 | 0,005 | 26,114 | 0,265 | 261,458 | 234,340 | 0,005 | 26,861 | 0,252 |
| | 10000 | 0,317 | 0,224 | 0,000 | 0,093 | 0,000 | 0,392 | 0,272 | 0,000 | 0,117 | 0,003 |
| | 30000 | 3,000 | 2,335 | 0,000 | 0,650 | 0,015 | 3,073 | 2,197 | 0,000 | 0,865 | 0,010 |
| | 50000 | 7,555 | 5,881 | 0,000 | 1,653 | 0,021 | 7,291 | 5,371 | 0,001 | 1,900 | 0,019 |
| cup98 | 70000 | 12,672 | 9,672 | 0,000 | 2,969 | 0,031 | 12,886 | 9,563 | 0,001 | 3,293 | 0,029 |
| | 90000 | 19,957 | 15,465 | 0,000 | 4,446 | 0,046 | 19,820 | 15,316 | 0,002 | 4,462 | 0,040 |

Analiza wyników pozwoliła mi zauważyć, że różnice w czasach wykonania algorytmu na danych tekstowych są zdecydowanie mniejsze niż w przypadku pozostałych zbiorów. Dla zbioru karypis_review różnice te są na tyle małe, że nie można na jego podstawie wskazać punktu referencyjnego, który najbardziej przyspiesza wyszukiwanie k sąsiedztwa. Badania wykonane na zbiorach cup98 i covtype pozwalają wyeliminować odpowiednio punkty [max_min] i [rand] z listy potencjalnych kandydatów punktów najbardziej przyspieszających wykonanie algorytmu. Natomiast punkt maksymalny pozwala osiągać jedne z najlepszych rezultatów w przypadku wszystkich zbiorów poza karypis_sport, dla którego algorytm wykonuje się najdłużej dla wszystkich badanych punktów referencyjnych. Punktem, dla którego

w znakomitej większości eksperymentów wyszukiwanie k sąsiedztwa wykonuje się najszybciej jest punkt minimalny.



Rys. 15. Porównanie wydajności algorytmu TI-k-Neighborhood-Index w zależności od wybranego punktu referencyjnego przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

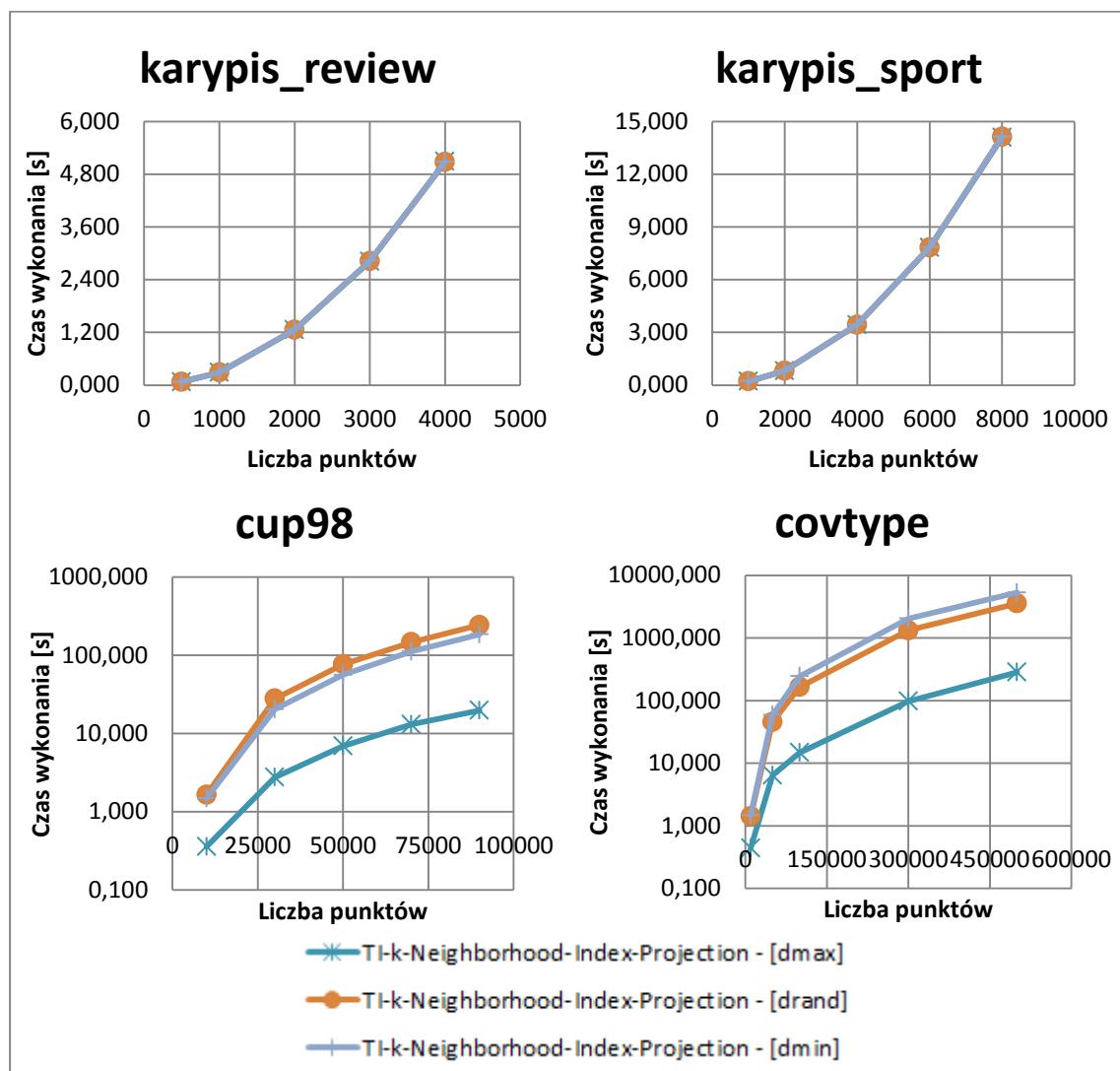
W dalszej części pracy jako wyniki czasowe algorytmu *TI-k-Neighborhood-Index* będą prezentowane rezultaty osiągnięte przy zastosowaniu punktu minimalnego jako punktu referencyjnego.

6.2.2. Badania algorytmu k-Neighborhood-Index-Projection

Dla algorytmu *k-Neighborhood-Index-Projection* zbadałem następujące rodzaje rzutowania:

- rzutowanie na wymiar o najliczniejszej dziedzinie – oznaczany [dmax];
- rzutowanie na niezerowy wymiar o najmniej licznej dziedzinie – oznaczany [dmin];
- rzutowanie na losowy wymiar – oznaczany [drand];

W tab. 11 zamieściłem czasy uruchomień algorytmu *k-Neighborhood-Index-Projection* wraz z trwaniem składających się na niego kroków. Na rys. 16 znajdują się wykresy czasu wykonania algorytmu w funkcji liczby punktów.



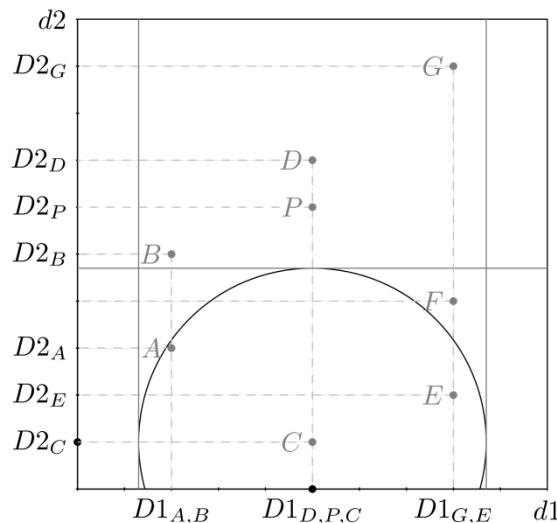
Rys. 16. Porównanie wydajności algorytmu k-Neighborhood-Index-Projection w zależności od wymiaru rzutowania . Wykresy zawierają czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

Tab. 11. Porównanie wydajności *k-Neighborhood-Index-Projection* w zależności od wymiaru rzutowania przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

| zbior | l. p. | <i>k-Neighborhood-Index-Projection [dmax]</i> | | | | | <i>k-Neighborhood-Index-Projection [drand]</i> | | | | | <i>k-Neighborhood-Index-Projection [dmin]</i> | | | | |
|----------------|--------|---|---------|-------|--------|-------|--|----------|-------|--------|-------|---|----------|-------|--------|-------|
| | | wyk. | wysz. | bud. | obl. | sort. | wyk. | wysz. | bud. | obl. | sort. | wyk. | wysz. | bud. | obl. | sort. |
| | | alg. | sąsiad. | ind. | rz. | | alg. | sąsiad. | ind. | rz. | | alg. | sąsiad. | ind. | rz. | |
| karypis_sport | 1000 | 0,214 | 0,213 | 0,000 | 0,001 | 0,000 | 0,215 | 0,214 | 0,000 | 0,000 | 0,000 | 0,214 | 0,214 | 0,000 | 0,000 | 0,000 |
| | 2000 | 0,809 | 0,807 | 0,000 | 0,002 | 0,000 | 0,810 | 0,809 | 0,000 | 0,001 | 0,000 | 0,812 | 0,811 | 0,000 | 0,001 | 0,000 |
| | 4000 | 3,436 | 3,432 | 0,000 | 0,004 | 0,000 | 3,433 | 3,429 | 0,000 | 0,004 | 0,000 | 3,448 | 3,444 | 0,000 | 0,003 | 0,000 |
| | 6000 | 7,830 | 7,823 | 0,000 | 0,006 | 0,000 | 7,829 | 7,822 | 0,000 | 0,007 | 0,000 | 7,850 | 7,844 | 0,000 | 0,006 | 0,000 |
| | 8000 | 14,077 | 14,068 | 0,000 | 0,008 | 0,000 | 14,125 | 14,116 | 0,000 | 0,008 | 0,000 | 14,128 | 14,122 | 0,000 | 0,006 | 0,000 |
| | 500 | 0,073 | 0,073 | 0,000 | 0,000 | 0,000 | 0,073 | 0,073 | 0,000 | 0,000 | 0,000 | 0,074 | 0,074 | 0,000 | 0,000 | 0,000 |
| karypis_review | 1000 | 0,288 | 0,287 | 0,000 | 0,001 | 0,000 | 0,289 | 0,289 | 0,000 | 0,000 | 0,000 | 0,289 | 0,289 | 0,000 | 0,001 | 0,000 |
| | 2000 | 1,258 | 1,257 | 0,000 | 0,001 | 0,000 | 1,259 | 1,258 | 0,000 | 0,001 | 0,000 | 1,263 | 1,261 | 0,000 | 0,001 | 0,000 |
| | 3000 | 2,813 | 2,811 | 0,000 | 0,002 | 0,000 | 2,822 | 2,820 | 0,000 | 0,002 | 0,000 | 2,826 | 2,824 | 0,000 | 0,002 | 0,000 |
| | 4000 | 5,088 | 5,085 | 0,000 | 0,003 | 0,000 | 5,074 | 5,071 | 0,000 | 0,003 | 0,000 | 5,079 | 5,076 | 0,000 | 0,003 | 0,000 |
| | 10000 | 0,440 | 0,320 | 0,000 | 0,118 | 0,002 | 1,412 | 1,311 | 0,000 | 0,101 | 0,000 | 1,473 | 1,386 | 0,000 | 0,085 | 0,002 |
| | 50000 | 6,376 | 4,026 | 0,001 | 2,335 | 0,013 | 45,292 | 43,381 | 0,001 | 1,907 | 0,004 | 58,654 | 56,548 | 0,001 | 2,097 | 0,008 |
| covtype | 100000 | 14,753 | 9,833 | 0,001 | 4,891 | 0,028 | 164,137 | 159,576 | 0,002 | 4,557 | 0,002 | 243,611 | 238,447 | 0,002 | 5,145 | 0,018 |
| | 300000 | 97,557 | 82,223 | 0,003 | 15,230 | 0,101 | 1318,407 | 1302,450 | 0,003 | 15,947 | 0,006 | 2032,951 | 2016,690 | 0,003 | 16,179 | 0,079 |
| | 500000 | 283,132 | 256,854 | 0,005 | 26,091 | 0,182 | 3559,763 | 3532,690 | 0,005 | 27,013 | 0,055 | 5290,897 | 5263,930 | 0,005 | 26,815 | 0,147 |
| | 10000 | 0,357 | 0,239 | 0,000 | 0,116 | 0,002 | 1,648 | 1,537 | 0,000 | 0,110 | 0,001 | 1,483 | 1,379 | 0,000 | 0,103 | 0,001 |
| | 30000 | 2,756 | 2,094 | 0,000 | 0,653 | 0,008 | 27,807 | 27,113 | 0,001 | 0,690 | 0,004 | 20,347 | 19,668 | 0,001 | 0,677 | 0,002 |
| | 50000 | 6,906 | 5,230 | 0,001 | 1,660 | 0,015 | 76,375 | 74,429 | 0,001 | 1,940 | 0,006 | 55,897 | 54,189 | 0,001 | 1,703 | 0,003 |
| cup98 | 70000 | 13,061 | 10,041 | 0,001 | 2,997 | 0,022 | 145,992 | 142,618 | 0,001 | 3,362 | 0,011 | 111,705 | 108,640 | 0,001 | 3,059 | 0,004 |
| | 90000 | 19,727 | 15,146 | 0,001 | 4,549 | 0,032 | 242,929 | 238,615 | 0,001 | 4,300 | 0,012 | 184,648 | 180,108 | 0,001 | 4,533 | 0,006 |

Różnice w czasach wykonania algorytmu dla strategii rzutowania wykonanego na zbiorach tekstowych są na tyle niewielkie, że nie pozwalają na wyciągnięcie wniosków na temat użyteczności danej metody. Rezultaty eksperymentów wykonanych na gęstych zbiorach danych świadczą, że projekcja na wymiar o największej dziedzinie pozwala na przyspieszenie wyznaczania k-sąsiedztwa o rząd wielkości w porównaniu do pozostałych strategii rzutowania. Liczność dziedziny wymiaru, na który wykonywana jest projekcja, ma kluczowy wpływ na sprawność algorytmu *k-Neighborhood-Index-Projection*. Następujące wyjaśnienie jest oparte na Eps-sąsiedztwie bez straty wartości merytorycznej dla k-sąsiedztwa, ponieważ problem k-sąsiedztwa można sprowadzić do problemu Eps-sąsiedztwa.

Na rys. 17 przedstawiono przykładowy zbiór D dwuwymiarowej przestrzeni ($d1, d2$). Przerywanymi liniami zaznaczono rzuty punktów zbioru D odpowiednio na wymiary $d1$ i $d2$. Zbiór punktów $D1$ ($D1 = \{D1_{A,B}, D1_{D,P,C}, D1_{G,E}\}$) będących wynikiem rzutowania zbioru D na $d1$ ma liczbę 3, natomiast zbiór $D2$ ($D2 = \{D2_C, D2_E, D2_A, D2_F, D2_B, D2_P, D2_D, D2_G\}$) powstały w wyniku rzutowania D na $d2$ jest licząc 8.



Rys. 17. Zbiór punktów D

Załóżmy, że szukamy pewnego otoczenia epsilonowego punktu C (na rys. 17 epsilonowe otoczenie zostało oznaczone fragmentem okręgu). Gdy posłużymy się projekcją na $d1$, to żaden punkt zbioru D nie zostanie odrzucony w procesie wyznaczania potencjalnych sąsiadów na podstawie kryterium rzutowania na dany wymiar, ponieważ wszystkie punkty zbioru $D1$ należą

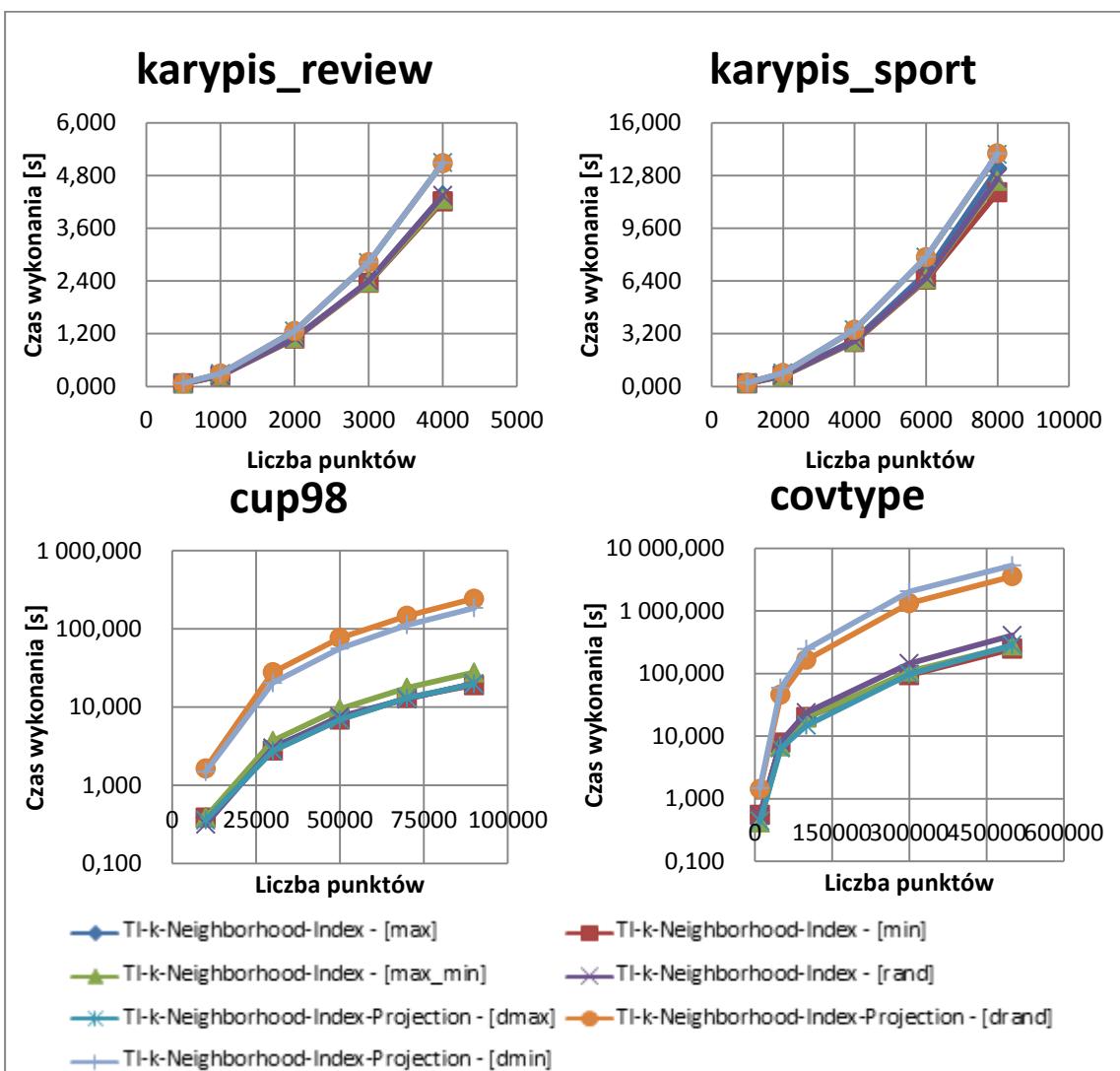
do otoczenia epsilonowego punktu powstałego po projekcji P na $d1$ (krawędzie rzutów otoczenia epsilonowego punktu C na wymiary $d1$ i $d2$ zostały oznaczone na rys. 17 odpowiednio szarymi ciągłymi liniami pionowymi i linią poziomą). Tym samym zastosowanie rzutowania w tym szczególnym przypadku nie przyspiesza wyznaczania sąsiedztwa. Gdy posłużymy się rzutowaniem na $d2$, to aż połowa punktów zbioru D zostanie odrzuconych w procesie wyznaczania potencjalnych sąsiadów na podstawie kryterium rzutowania na dany wymiar co znacząco przyspieszy wyznaczanie sąsiedztwa.

W algorytmie *k-Neighborhood-Index-Projection* rzutowanie tym mocniej wspiera selektywność wyznaczania potencjalnych sąsiadów im liczniejsza jest dziedzina wymiaru, na który wykonywana jest projekcja. Dlatego w dalszej części pracy jako wyniki czasowe algorytmu *k-Neighborhood-Index-Projection* będą prezentowane rezultaty osiągnięte przy zastosowaniu rzutowania na wymiar [dmax].

6.2.3. Porównanie algorytmów *k-Neighborhood-Index-Projection* z *TI-k-Neighborhood-Index*

W celu porównania podejścia korzystającego z nierówności trójkąta z podejściem stosującym rzutowanie, zamieściłem na rys. 18 wykresy czasów wykonania algorytmów *k-Neighborhood-Index-Projection* i *TI-k-Neighborhood-Index* w funkcji liczby punktów we wszystkich badanych przypadkach zastosowania punktu referencyjnego i strategii rzutowania na wymiar. Dane zamieszczone na wykresach odpowiadają tym zgromadzonym w tabelach tab. 9, tab. 10 i tab. 11.

Z rezultatów eksperymentów wynika, że dla danych tekstowych algorytm *k-neighborhood-Index-Projection* wykorzystujący rzutowanie wykonuje się nieznacznie wolniej niż *TI-k-Neighborhood-Index*. W przypadku gęstych zbiorów danych wyszukiwanie sąsiedztwa z zastosowaniem rzutowania na [drand] i [dmin] realizuje się o rząd wielkości wolniej niż w pozostałych przypadkach. Natomiast sprawność *k-neighborhood-Index-Projection* z rzutowaniem na [dmax] jest porównywalna z *TI-k-Neighborhood-Index* dla badanych punktów referencyjnych. Na niekorzyść rzutowania w porównaniu z nierównością trójkąta przemawia zależność tej metody od liczności dziedzin wymiarów.

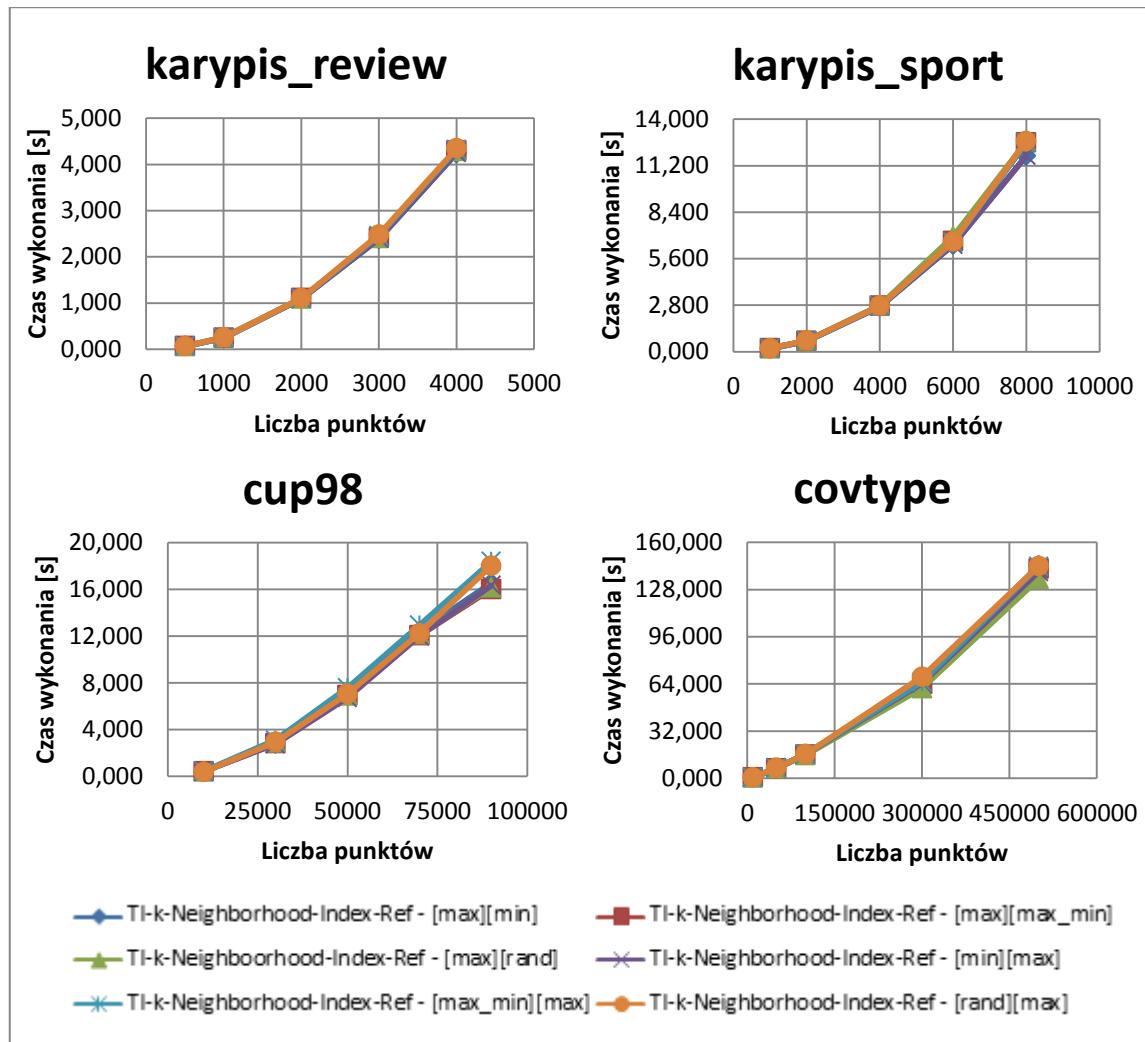


Rys. 18. Porównanie wydajności algorytmu k-Neighborhood-Index-Projection z TI-k-Neighborhood-Index w zależności od wybranego wymiaru rzutowania i punktu referencyjnego przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań poszukiwań $k=5$ sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

6.2.4. Badania algorytmu TI-k-Neighborhood-Index-Ref – wybór dwóch punktów referencyjnych

W celu poznania wpływu doboru punktów referencyjnych na wydajność algorytmu *TI-k-Neighborhood-Index-Ref* przeprowadziłem eksperymenty testujące różne pary punktów referencyjnych. Podobnie jak w rozdziale 6.2.1.3. w swoich badaniach skupiłem się na punktach [max], [min], [max_min] oraz [rand]. W tabelach tab. 12 i tab. 13 zamieściłem czasy

uruchomień algorytmu *TI-k-Neighborhood-Index -Ref* wraz z trwaniem składających się na niego kroków. Na rys. 19 zaprezentowałem wykresy czasów wykonania algorytmu w funkcji liczby punktów.



Rys. 19. Porównanie wydajności algorytmu *TI-k-Neighborhood-Index-Ref* w zależności od wybranej pary punktów referencyjnych przy zastosowaniu odległości euklidesowej jako miary podobieństwa Wykresy zawierają czasy wykonania poszukiwań $k=5$ sąsiadztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

Tab. 12. Porównanie wydajności TI-k-Neighborhood-Index-Ref w zależności od pary punktów referencyjnych przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

| zbior | l. p. | TI-k-Neighborhood-Index-Ref [max][min] | | | | | TI-k-Neighborhood-Index-Ref [max][max_min] | | | | | TI-k-Neighborhood-Index-Ref [max][rand] | | | | |
|----------------|--------|--|---------|-------|--------|-------|--|---------|-------|--------|-------|---|---------|-------|--------|-------|
| | | wyk. | wysz. | bud. | obl. | sort. | wyk. | wysz. | bud. | obl. | sort. | wyk. | wysz. | bud. | obl. | sort. |
| | | alg. | sąsiad. | ind. | odl. | | alg. | sąsiad. | ind. | odl. | | alg. | sąsiad. | ind. | odl. | |
| karypis_sport | 1000 | 0,176 | 0,171 | 0,000 | 0,005 | 0,000 | 0,202 | 0,196 | 0,000 | 0,006 | 0,000 | 0,202 | 0,194 | 0,000 | 0,008 | 0,000 |
| | 2000 | 0,634 | 0,629 | 0,000 | 0,005 | 0,000 | 0,647 | 0,638 | 0,000 | 0,008 | 0,000 | 0,663 | 0,655 | 0,000 | 0,008 | 0,000 |
| | 4000 | 2,694 | 2,678 | 0,000 | 0,016 | 0,000 | 2,757 | 2,738 | 0,000 | 0,018 | 0,001 | 2,796 | 2,775 | 0,000 | 0,020 | 0,001 |
| | 6000 | 6,427 | 6,396 | 0,000 | 0,031 | 0,000 | 6,646 | 6,617 | 0,000 | 0,028 | 0,002 | 6,927 | 6,911 | 0,000 | 0,015 | 0,002 |
| | 8000 | 11,783 | 11,726 | 0,000 | 0,057 | 0,000 | 12,563 | 12,483 | 0,000 | 0,077 | 0,003 | 12,656 | 12,589 | 0,000 | 0,067 | 0,000 |
| | 500 | 0,062 | 0,062 | 0,000 | 0,000 | 0,000 | 0,067 | 0,064 | 0,000 | 0,003 | 0,000 | 0,062 | 0,062 | 0,000 | 0,000 | 0,000 |
| karypis_review | 1000 | 0,244 | 0,239 | 0,000 | 0,005 | 0,000 | 0,249 | 0,243 | 0,000 | 0,006 | 0,000 | 0,260 | 0,250 | 0,000 | 0,010 | 0,000 |
| | 2000 | 1,092 | 1,076 | 0,000 | 0,016 | 0,000 | 1,103 | 1,092 | 0,000 | 0,011 | 0,000 | 1,087 | 1,076 | 0,000 | 0,010 | 0,000 |
| | 3000 | 2,381 | 2,366 | 0,000 | 0,015 | 0,000 | 2,418 | 2,401 | 0,000 | 0,016 | 0,001 | 2,398 | 2,377 | 0,000 | 0,016 | 0,005 |
| | 4000 | 4,238 | 4,212 | 0,000 | 0,026 | 0,000 | 4,303 | 4,280 | 0,000 | 0,022 | 0,001 | 4,285 | 4,264 | 0,000 | 0,021 | 0,000 |
| | 10000 | 0,515 | 0,276 | 0,000 | 0,229 | 0,010 | 0,461 | 0,294 | 0,000 | 0,164 | 0,003 | 0,484 | 0,317 | 0,000 | 0,167 | 0,000 |
| | 50000 | 6,661 | 4,227 | 0,000 | 2,418 | 0,015 | 6,775 | 4,340 | 0,001 | 2,419 | 0,015 | 6,682 | 4,222 | 0,000 | 2,444 | 0,016 |
| covtype | 100000 | 16,260 | 10,956 | 0,005 | 5,268 | 0,031 | 16,161 | 10,943 | 0,002 | 5,180 | 0,035 | 15,678 | 10,380 | 0,000 | 5,262 | 0,036 |
| | 300000 | 65,057 | 48,391 | 0,005 | 16,526 | 0,135 | 64,109 | 47,364 | 0,004 | 16,609 | 0,132 | 61,146 | 44,642 | 0,000 | 16,374 | 0,130 |
| | 500000 | 144,289 | 115,575 | 0,005 | 28,454 | 0,255 | 142,420 | 113,920 | 0,005 | 28,234 | 0,260 | 135,617 | 107,516 | 0,005 | 27,841 | 0,255 |
| | 10000 | 0,406 | 0,271 | 0,000 | 0,135 | 0,000 | 0,411 | 0,244 | 0,000 | 0,164 | 0,003 | 0,405 | 0,265 | 0,000 | 0,140 | 0,000 |
| | 30000 | 2,783 | 1,919 | 0,000 | 0,848 | 0,016 | 2,831 | 1,953 | 0,000 | 0,867 | 0,010 | 2,876 | 2,028 | 0,000 | 0,832 | 0,015 |
| | 50000 | 7,057 | 5,070 | 0,000 | 1,971 | 0,016 | 6,952 | 4,885 | 0,001 | 2,047 | 0,019 | 6,905 | 4,898 | 0,000 | 1,991 | 0,016 |
| cup98 | 70000 | 12,397 | 8,829 | 0,000 | 3,536 | 0,031 | 12,060 | 8,416 | 0,001 | 3,612 | 0,031 | 12,157 | 8,429 | 0,000 | 3,697 | 0,031 |
| | 90000 | 16,552 | 12,210 | 0,000 | 4,300 | 0,042 | 16,250 | 11,939 | 0,001 | 4,268 | 0,041 | 16,312 | 12,074 | 0,000 | 4,196 | 0,042 |

Tab. 13. Porównanie wydajności TI-k-Neighborhood-Index-Ref w zależności od pary punktów referencyjnych przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 10 % losowo wybranych punktów zbioru danych

| zbior | l. p. | TI-k-Neighborhood-Index-Ref [min][max] | | | | | TI-k-Neighborhood-Index-Ref [max_min][max] | | | | | TI-k-Neighborhood-Index-Ref [rand][max] | | | | |
|----------------|--------|--|---------|-------|--------|-------|--|---------|-------|--------|-------|---|---------|-------|--------|-------|
| | | wyk. | wysz. | bud. | obl. | sort. | wyk. | wysz. | bud. | obl. | sort. | wyk. | wysz. | bud. | obl. | sort. |
| | | alg. | sąsiad. | ind. | odl. | | alg. | sąsiad. | ind. | odl. | | alg. | sąsiad. | ind. | odl. | |
| karypis_sport | 1000 | 0,174 | 0,168 | 0,000 | 0,005 | 0,001 | 0,202 | 0,196 | 0,000 | 0,006 | 0,000 | 0,202 | 0,194 | 0,000 | 0,008 | 0,000 |
| | 2000 | 0,640 | 0,632 | 0,000 | 0,008 | 0,000 | 0,648 | 0,640 | 0,000 | 0,008 | 0,000 | 0,662 | 0,654 | 0,000 | 0,008 | 0,000 |
| | 4000 | 2,719 | 2,698 | 0,000 | 0,019 | 0,001 | 2,746 | 2,726 | 0,000 | 0,019 | 0,001 | 2,752 | 2,732 | 0,000 | 0,019 | 0,001 |
| | 6000 | 6,439 | 6,410 | 0,000 | 0,027 | 0,002 | 6,582 | 6,552 | 0,000 | 0,029 | 0,002 | 6,623 | 6,591 | 0,000 | 0,030 | 0,002 |
| | 8000 | 11,795 | 11,744 | 0,000 | 0,048 | 0,002 | 12,589 | 12,483 | 0,000 | 0,103 | 0,002 | 12,684 | 12,626 | 0,000 | 0,055 | 0,003 |
| | 500 | 0,064 | 0,061 | 0,000 | 0,003 | 0,000 | 0,067 | 0,064 | 0,000 | 0,003 | 0,000 | 0,068 | 0,064 | 0,000 | 0,004 | 0,000 |
| karypis_review | 1000 | 0,245 | 0,239 | 0,000 | 0,005 | 0,001 | 0,248 | 0,242 | 0,000 | 0,006 | 0,000 | 0,250 | 0,243 | 0,000 | 0,006 | 0,000 |
| | 2000 | 1,088 | 1,077 | 0,000 | 0,011 | 0,000 | 1,107 | 1,095 | 0,000 | 0,012 | 0,000 | 1,105 | 1,092 | 0,000 | 0,012 | 0,001 |
| | 3000 | 2,393 | 2,377 | 0,000 | 0,016 | 0,000 | 2,471 | 2,454 | 0,000 | 0,016 | 0,001 | 2,479 | 2,461 | 0,000 | 0,017 | 0,001 |
| | 4000 | 4,254 | 4,232 | 0,000 | 0,021 | 0,001 | 4,345 | 4,322 | 0,000 | 0,022 | 0,001 | 4,347 | 4,323 | 0,000 | 0,023 | 0,001 |
| | 10000 | 0,481 | 0,338 | 0,000 | 0,140 | 0,003 | 0,596 | 0,381 | 0,000 | 0,211 | 0,003 | 0,571 | 0,342 | 0,000 | 0,226 | 0,003 |
| | 50000 | 6,968 | 4,470 | 0,001 | 2,481 | 0,015 | 6,922 | 4,477 | 0,001 | 2,427 | 0,016 | 7,157 | 4,594 | 0,001 | 2,546 | 0,016 |
| covtype | 100000 | 16,343 | 10,977 | 0,002 | 5,328 | 0,035 | 16,451 | 11,038 | 0,002 | 5,374 | 0,036 | 16,581 | 11,310 | 0,002 | 5,232 | 0,037 |
| | 300000 | 64,407 | 47,668 | 0,003 | 16,601 | 0,135 | 65,557 | 48,845 | 0,003 | 16,571 | 0,137 | 69,293 | 52,593 | 0,003 | 16,551 | 0,145 |
| | 500000 | 140,108 | 111,918 | 0,005 | 27,926 | 0,259 | 143,659 | 115,806 | 0,005 | 27,585 | 0,262 | 144,438 | 116,489 | 0,005 | 27,660 | 0,285 |
| | 10000 | 0,397 | 0,254 | 0,000 | 0,140 | 0,003 | 0,440 | 0,291 | 0,000 | 0,146 | 0,003 | 0,371 | 0,251 | 0,001 | 0,117 | 0,003 |
| | 30000 | 2,749 | 1,845 | 0,001 | 0,892 | 0,011 | 3,106 | 2,205 | 0,001 | 0,890 | 0,011 | 2,935 | 2,109 | 0,000 | 0,815 | 0,011 |
| | 50000 | 6,734 | 4,939 | 0,000 | 1,776 | 0,019 | 7,575 | 5,525 | 0,001 | 2,030 | 0,019 | 7,049 | 5,107 | 0,001 | 1,922 | 0,019 |
| cup98 | 70000 | 11,994 | 8,572 | 0,001 | 3,392 | 0,029 | 12,877 | 9,588 | 0,001 | 3,258 | 0,030 | 12,175 | 8,987 | 0,001 | 3,157 | 0,030 |
| | 90000 | 16,410 | 12,269 | 0,001 | 4,098 | 0,041 | 18,446 | 14,035 | 0,001 | 4,367 | 0,042 | 17,986 | 13,539 | 0,001 | 4,404 | 0,042 |

W oparciu o rezultaty przeprowadzonych eksperymentów trudno jednoznacznie wskazać, która para punktów referencyjnych najbardziej przyspiesza wyznaczenie k sąsiedztwa. Wyniki świadczą, że najkorzystniej jako pierwszy punkt referencyjny wybrać [max], ponieważ w większości przypadków eksperymenty z punktem maksymalnym jako pierwszym punktem referencyjnym wykonują się szybciej niż dla punktu losowego [rand] czy innego punktu skrajnego – [min].

Różnice w czasach wykonania algorytmu dla poszczególnych zestawów punktów referencyjnych są na tyle niewielkie, że nie pozwalają jednoznacznie stwierdzić, która z badanych kombinacji punktów referencyjnych najbardziej przyspiesza wykonanie algorytmu. Tym co wyniki badań pozwalają stwierdzić jest wniosek, że jako pierwszy punkt referencyjny najlepiej jest wybrać punkt maksymalny.

Ponieważ dla algorytmu *TI-k-Neighborhood-Index* najlepszym punktem referencyjnym okazał się być [min], dlatego w dalszej części pracy jako wyniki czasowe algorytmu *TI-k-Neighborhood-Index-Ref* będą prezentowane rezultaty osiągnięte przy zastosowaniu pary punktów referencyjnych [max][min].

6.2.5. Porównanie implementacji odmian algorytmu k-Neighborhood-Index

W tab. 14, tab. 15 i na rys. 20 przedstawiłem rezultaty badań odmian algorytmu *k-Neighborhood-Index*. Wyniki algorytmu *TI-k-Neighborhood-Index* zostały zebrane dla punktu referencyjnego [min], *k-Neighborhood-Index-Projection* dla [dmax], natomiast rezultaty *TI-k-Neighborhood-Ref* dla pary punktów referencyjnych [max][min]. Algorytm *k-Neighborhood-Index-Brute-Force* jest naiwną implementacją wyszukiwania k sąsiedztwa o złożoności kwadratowej.

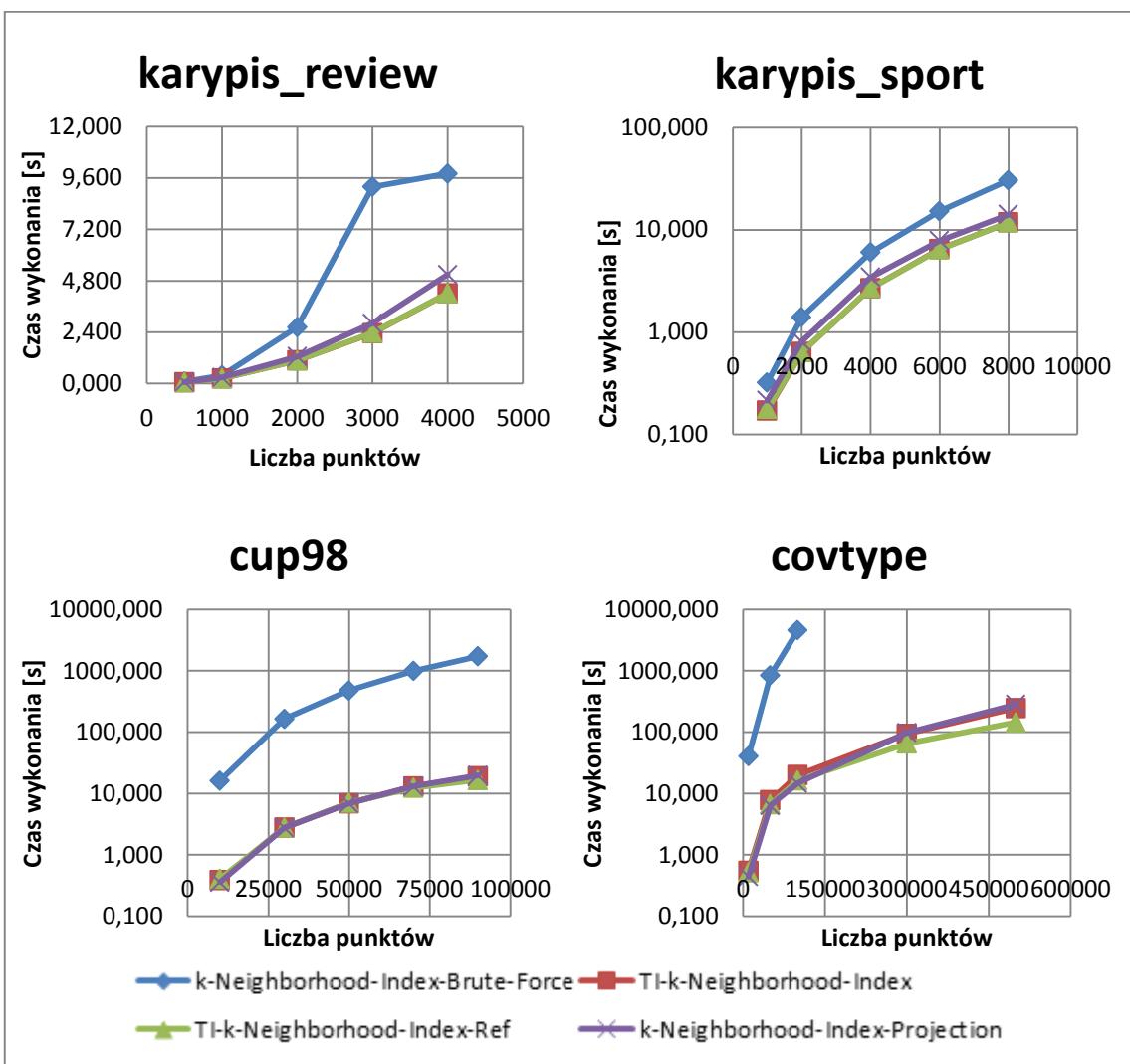
Uzyskane rezultaty jednoznacznie wskazują na większą wydajność *TI-k-Neighborhood-Index*, *TI-k-Neighborhood-Index-Ref* oraz *k-Neighborhood-Index-Projection* w stosunku do algorytmu *k-Neighborhood-Index-Brute-Force*. Zastosowanie nierówności trójkąta pozwala uzyskiwać wyniki o dwa rzędy wielkości szybciej. Wzrost ten jest mniej widoczny dla danych tekstowych niż dla pozostałych zbiorów z uwagi na ich rzadki charakter. Zwiększenie liczby punktów referencyjnych z jednego do dwóch nie przyniosło znaczącej poprawy sprawności wyznaczania k sąsiedztwa. Dalsze zwiększanie liczby punktów referencyjnych może spowolnić wykonanie algorytmu, ponieważ koszt obsługi wielu punktów referencyjnych może przewyższyć zysk z ich zastosowania.

Tab. 14. Porównanie wydajności algorytmów k-Neighborhood-Index-Brute-Force i TI-k-Neighborhood-Index przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

| zbior | l. p. | k-Neighborhood-Index-Brute-Force | | | | | TI-k-Neighborhood-Index | | | | |
|----------------|--------|----------------------------------|----------|-------|-------|-------|-------------------------|---------|-------|--------|-------|
| | | wyk. | wysz. | bud. | obl. | sort. | wyk. | wysz. | bud. | obl. | sort. |
| | | alg. | sąsiad. | ind. | odl. | | alg. | sąsiad. | ind. | odl. | |
| karypis_sport | 1000 | 0,322 | 0,320 | 0,000 | 0,002 | 0,000 | 0,171 | 0,169 | 0,000 | 0,002 | 0,000 |
| | 2000 | 1,397 | 1,394 | 0,000 | 0,004 | 0,000 | 0,636 | 0,632 | 0,000 | 0,004 | 0,000 |
| | 4000 | 6,014 | 6,005 | 0,000 | 0,008 | 0,001 | 2,684 | 2,675 | 0,000 | 0,008 | 0,001 |
| | 6000 | 15,178 | 15,162 | 0,000 | 0,014 | 0,002 | 6,470 | 6,455 | 0,000 | 0,013 | 0,002 |
| | 8000 | 30,516 | 30,497 | 0,000 | 0,016 | 0,002 | 11,770 | 11,751 | 0,000 | 0,017 | 0,002 |
| karypis_review | 500 | 0,093 | 0,092 | 0,000 | 0,001 | 0,000 | 0,062 | 0,061 | 0,000 | 0,001 | 0,000 |
| | 1000 | 0,385 | 0,383 | 0,000 | 0,002 | 0,000 | 0,241 | 0,239 | 0,000 | 0,002 | 0,000 |
| | 2000 | 2,631 | 2,626 | 0,000 | 0,005 | 0,000 | 1,084 | 1,079 | 0,000 | 0,005 | 0,000 |
| | 3000 | 9,183 | 9,175 | 0,000 | 0,007 | 0,001 | 2,356 | 2,348 | 0,000 | 0,007 | 0,001 |
| | 4000 | 9,803 | 9,793 | 0,000 | 0,009 | 0,001 | 4,209 | 4,198 | 0,000 | 0,010 | 0,001 |
| covtype | 10000 | 40,337 | 40,115 | 0,000 | 0,219 | 0,003 | 0,551 | 0,370 | 0,000 | 0,179 | 0,002 |
| | 50000 | 844,863 | 842,475 | 0,001 | 2,372 | 0,015 | 7,804 | 5,526 | 0,001 | 2,262 | 0,015 |
| | 100000 | 4575,759 | 4570,870 | 0,002 | 4,851 | 0,036 | 19,962 | 15,025 | 0,001 | 4,901 | 0,035 |
| | 300000 | - | - | - | - | - | 93,868 | 78,047 | 0,003 | 15,686 | 0,132 |
| | 500000 | - | - | - | - | - | 246,482 | 219,916 | 0,005 | 26,312 | 0,249 |
| cup98 | 10000 | 16,170 | 15,947 | 0,000 | 0,219 | 0,003 | 0,388 | 0,259 | 0,000 | 0,126 | 0,003 |
| | 30000 | 164,166 | 163,038 | 0,001 | 1,116 | 0,011 | 2,785 | 2,105 | 0,001 | 0,669 | 0,010 |
| | 50000 | 475,873 | 473,576 | 0,001 | 2,276 | 0,019 | 6,942 | 5,287 | 0,001 | 1,635 | 0,019 |
| | 70000 | 996,087 | 992,978 | 0,001 | 3,078 | 0,030 | 13,095 | 9,933 | 0,001 | 3,132 | 0,029 |
| | 90000 | 1717,090 | 1712,660 | 0,002 | 4,387 | 0,041 | 19,041 | 14,683 | 0,001 | 4,316 | 0,041 |

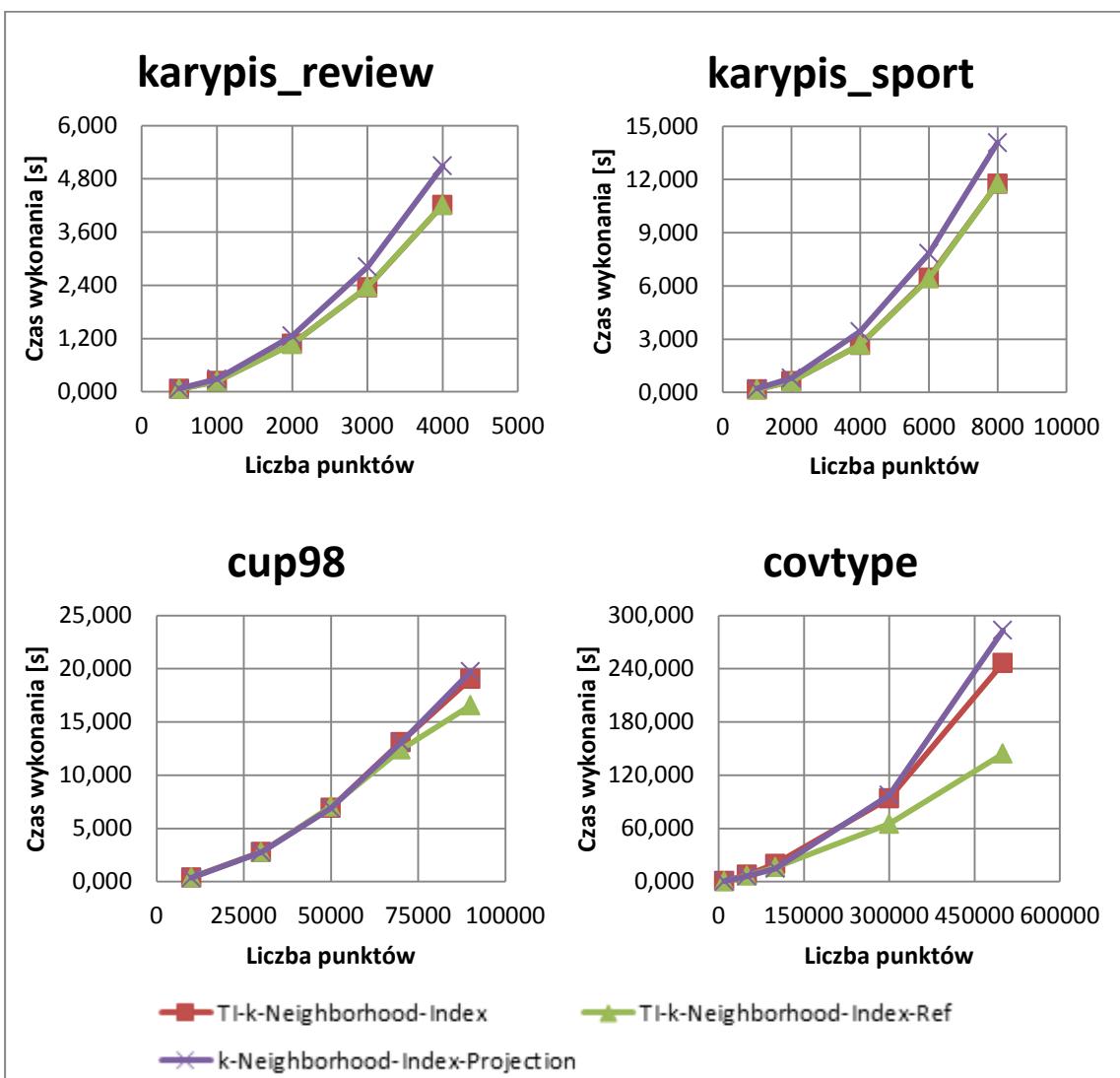
Tab. 15. Porównanie wydajności odmian algorytmów k-Neighborhood-Index-Projection i TI-k-Neighborhood-Index-Ref przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiadztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

| zbior | l. p. | k-Neighborhood-Index-Projection | | | | | TI-k-Neighborhood-Index-Ref | | | | |
|---------------|--------|---------------------------------|---------|-------|--------|-------|-----------------------------|---------|-------|--------|-------|
| | | wyk. | wysz. | bud. | obl. | sort. | wyk. | wysz. | bud. | obl. | sort. |
| | | alg. | sąsiad. | ind. | rz. | | alg. | sąsiad. | ind. | odl. | |
| karyps_sport | 1000 | 0,214 | 0,213 | 0,000 | 0,001 | 0,000 | 0,176 | 0,171 | 0,000 | 0,005 | 0,000 |
| | 2000 | 0,809 | 0,807 | 0,000 | 0,002 | 0,000 | 0,634 | 0,629 | 0,000 | 0,005 | 0,000 |
| | 4000 | 3,436 | 3,432 | 0,000 | 0,004 | 0,000 | 2,694 | 2,678 | 0,000 | 0,016 | 0,000 |
| | 6000 | 7,830 | 7,823 | 0,000 | 0,006 | 0,000 | 6,427 | 6,396 | 0,000 | 0,031 | 0,000 |
| | 8000 | 14,077 | 14,068 | 0,000 | 0,008 | 0,000 | 11,783 | 11,726 | 0,000 | 0,057 | 0,000 |
| | 500 | 0,073 | 0,073 | 0,000 | 0,000 | 0,000 | 0,062 | 0,062 | 0,000 | 0,000 | 0,000 |
| karyps_review | 1000 | 0,288 | 0,287 | 0,000 | 0,001 | 0,000 | 0,244 | 0,239 | 0,000 | 0,005 | 0,000 |
| | 2000 | 1,258 | 1,257 | 0,000 | 0,001 | 0,000 | 1,092 | 1,076 | 0,000 | 0,016 | 0,000 |
| | 3000 | 2,813 | 2,811 | 0,000 | 0,002 | 0,000 | 2,381 | 2,366 | 0,000 | 0,015 | 0,000 |
| | 4000 | 5,088 | 5,085 | 0,000 | 0,003 | 0,000 | 4,238 | 4,212 | 0,000 | 0,026 | 0,000 |
| | 10000 | 0,440 | 0,320 | 0,000 | 0,118 | 0,002 | 0,515 | 0,276 | 0,000 | 0,229 | 0,010 |
| | 50000 | 6,376 | 4,026 | 0,001 | 2,335 | 0,013 | 6,661 | 4,227 | 0,000 | 2,418 | 0,015 |
| covtype | 100000 | 14,753 | 9,833 | 0,001 | 4,891 | 0,028 | 16,260 | 10,956 | 0,005 | 5,268 | 0,031 |
| | 300000 | 97,557 | 82,223 | 0,003 | 15,230 | 0,101 | 65,057 | 48,391 | 0,005 | 16,526 | 0,135 |
| | 500000 | 283,132 | 256,854 | 0,005 | 26,091 | 0,182 | 144,289 | 115,575 | 0,005 | 28,454 | 0,255 |
| | 10000 | 0,357 | 0,239 | 0,000 | 0,116 | 0,002 | 0,406 | 0,271 | 0,000 | 0,135 | 0,000 |
| | 30000 | 2,756 | 2,094 | 0,000 | 0,653 | 0,008 | 2,783 | 1,919 | 0,000 | 0,848 | 0,016 |
| | 50000 | 6,906 | 5,230 | 0,001 | 1,660 | 0,015 | 7,057 | 5,070 | 0,000 | 1,971 | 0,016 |
| cup98 | 70000 | 13,061 | 10,041 | 0,001 | 2,997 | 0,022 | 12,397 | 8,829 | 0,000 | 3,536 | 0,031 |
| | 90000 | 19,727 | 15,146 | 0,001 | 4,549 | 0,032 | 16,552 | 12,210 | 0,000 | 4,300 | 0,042 |



Rys. 20. Porównanie wydajności odmian algorytmów k-Neighborhood-Index-Brute-Force, k-Neighborhood-Index-Projection, TI-k-Neighborhood-Index i TI-k-Neighborhood-Index-Ref przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

Na rys. 21 zaprezentowałem rezultaty przeprowadzonych eksperymentów z pominięciem algorytmu *k-Neighborhood-Index-Brute-Force* w celu uwidocznienia różnic między pozostałymi eksperymentami. Wyniki wykazują, że zastosowanie rzutowania nie przyspiesza wyszukiwania k sąsiedztwa bardziej niż wykorzystanie nierówności trójkąta.



Rys. 21. Porównanie wydajności odmian algorytmu k-Neighborhood-Index-Projection, k-Neighborhood-Index oraz TI-k-Neighborhood-Index-Ref przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

6.3. Badania algorytmu kNN-Index-Vp-Tree

W bieżącym rozdziale przedstawiłem rezultaty badań implementacji algorytmu *kNN-Index-Vp-Tree*. W swoich eksperymentach skupiłem się na metodach przeszukiwania indeksu metrycznego w celu wyznaczenia k sąsiadów danego zapytania. Badałem również wpływ implementacji punktu na wydajność algorytmu.

6.3.1. Implementacja algorytmu

Na przykładowych zbiorach danych przetestowałem dwie metody wyszukiwania k sąsiadów w oparciu o indeks metryczny:

- pierwszą, której kryterium przeszukiwania kolejnych gałęzi indeksu metrycznego stanowi mediana odległości punktów do punktu obserwacyjnego, zwaną dalej *metodą mediany*;
- drugą, której kryterium przeszukiwania kolejnych gałęzi indeksu metrycznego stanowią lewe i prawe ograniczenie, zwaną dalej *metodą ograniczeń*.

W tab. 16 i na rys. 22 zamieściłem czasy uruchomień algorytmu *kNN-Index-Vp-Tree* dla obu metod wyszukiwania k sąsiadów. Z rezultatów badań wynika, że *metoda ograniczeń* zapewnia szybsze wyszukiwanie k sąsiadów niż *metoda mediany*. Największa różnica w czasach wykonania algorytmów występuje dla wyszukiwania k sąsiadów spośród 500000 punktów zbioru covtype. Implementacja korzystająca z *metody ograniczeń* wykonuje się blisko 1,5 razy szybciej niż implementacja stosująca *metodę mediany*.

Wartym podkreślenia jest fakt, iż mimo, że w obu rozpatrywanych przypadkach indeks metryczny budowany był w ten sam sposób oraz mimo, że prezentowane są uśrednione wyniki, to różnice w czasie budowy indeksu są znaczące. Zachowanie to wynika z heurystyki zastosowanej w procesie wyboru punktu obserwacyjnego. Pewna kombinacja punktów obserwacyjnych pozwala zbudować indeks szybciej niż inna, stąd różnice w czasie budowy indeksu.

Jednakże wariancja ta nie wpływa decydująco na różnice w czasie wykonania obu implementacji. Powodem tych różnic jest kryterium wyszukiwania. Następujące wyjaśnienie jest oparte na Eps-sąsiadztwie bez utraty wartości merytorycznej dla k sąsiadów ponieważ problem wyznaczania k sąsiadów można sprowadzić do problemu wyznaczania Eps-sąsiadztwa o czym pisałem w rozdziale 4.1.

Tab. 16. Porównanie wydajności algorytmu kNN-Index-Vp-Tree w zależności od implementacji metody przeszukiwania indeksu metrycznego przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiadów w przykładowych zbiorach dla 10% losowo wybranych punktów zbioru danych

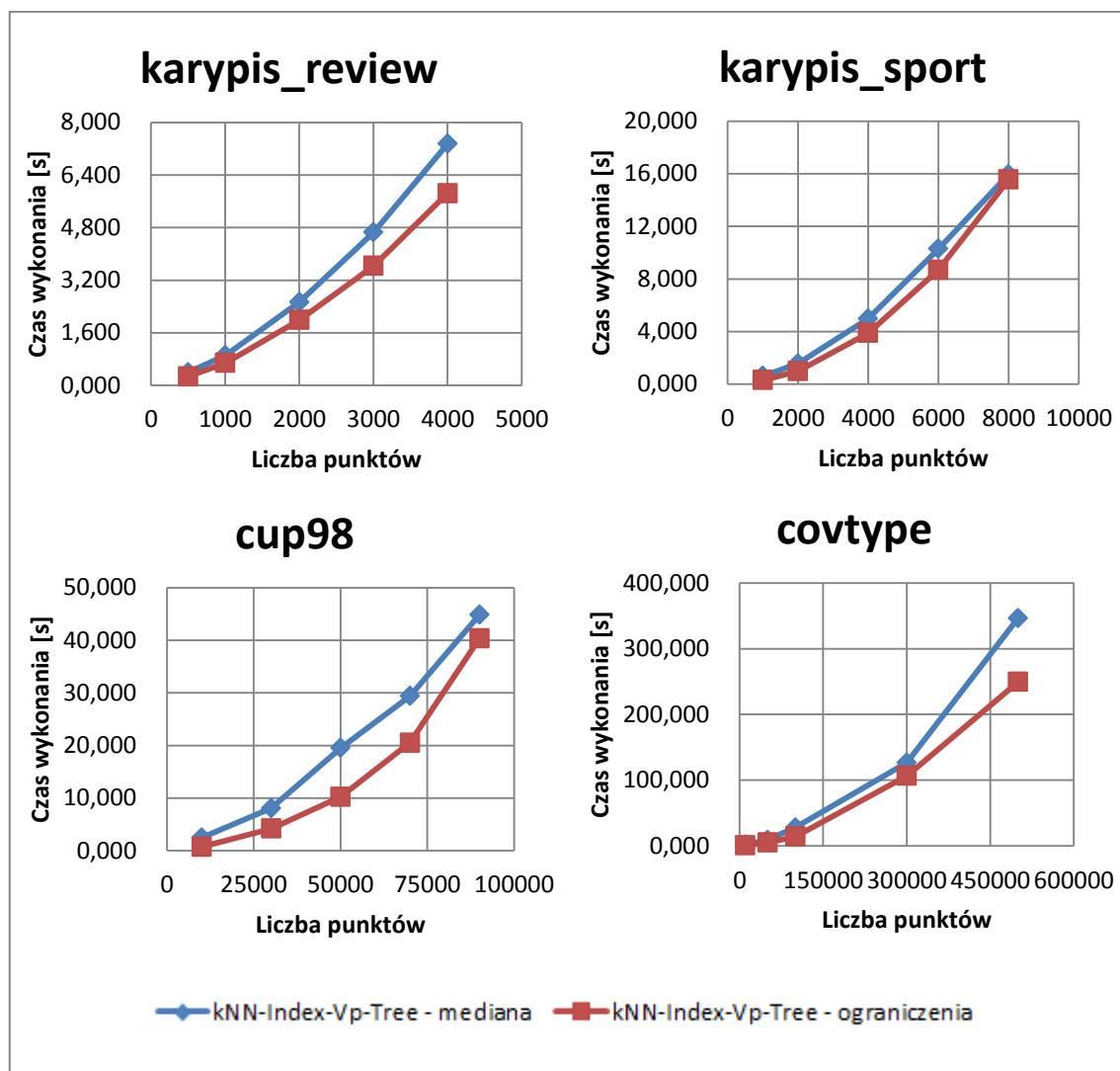
| zbior | l. p. | kNN-Index-Vp-Tree mediana | | | kNN-Index-Vp-Tree ograniczenia | | |
|----------------|--------|---------------------------|---------|---------|--------------------------------|---------|--------|
| | | wyk. | | bud. | wyk. | | wysz. |
| | | alg. | sąsiad. | ind. | alg. | sąsiad. | ind. |
| karypis_sport | 1000 | 0,614 | 0,241 | 0,373 | 0,296 | 0,218 | 0,078 |
| | 2000 | 1,562 | 0,883 | 0,678 | 0,988 | 0,822 | 0,166 |
| | 4000 | 4,976 | 3,747 | 1,229 | 3,879 | 3,510 | 0,369 |
| | 6000 | 10,276 | 8,483 | 1,793 | 8,684 | 8,107 | 0,577 |
| | 8000 | 15,944 | 15,039 | 0,905 | 15,543 | 14,768 | 0,774 |
| karypis_review | 500 | 0,392 | 0,084 | 0,308 | 0,270 | 0,078 | 0,192 |
| | 1000 | 0,911 | 0,318 | 0,593 | 0,675 | 0,296 | 0,379 |
| | 2000 | 2,530 | 1,385 | 1,145 | 1,986 | 1,326 | 0,660 |
| | 3000 | 4,650 | 3,089 | 1,561 | 3,630 | 2,949 | 0,681 |
| | 4000 | 7,344 | 5,462 | 1,882 | 5,840 | 5,247 | 0,593 |
| covtype | 10000 | 1,858 | 0,366 | 1,491 | 0,859 | 0,298 | 0,561 |
| | 50000 | 9,037 | 3,029 | 6,008 | 5,010 | 2,241 | 2,770 |
| | 100000 | 27,845 | 10,687 | 17,158 | 14,204 | 7,703 | 6,501 |
| | 300000 | 126,437 | 67,943 | 58,494 | 106,525 | 70,103 | 36,422 |
| | 500000 | 346,216 | 219,689 | 126,527 | 249,375 | 180,813 | 68,562 |
| cup98 | 10000 | 2,477 | 0,494 | 1,983 | 0,728 | 0,307 | 0,421 |
| | 30000 | 8,068 | 4,054 | 4,014 | 4,178 | 2,812 | 1,366 |
| | 50000 | 19,508 | 11,432 | 8,076 | 10,215 | 7,766 | 2,450 |
| | 70000 | 29,371 | 18,886 | 10,485 | 20,500 | 16,790 | 3,710 |
| | 90000 | 44,818 | 29,800 | 15,018 | 40,305 | 27,149 | 13,156 |

Rys. 23 przedstawia sytuację znajdowania Eps-sąsiedztwa punktu P (obszar pokryty czarno-białą kratą) w węźle VP indeksu metrycznego *metodą mediany*. Fragmentem okręgu zaznaczono medianę odległości punktów od punktu obserwacyjnego. Algorytm znajdowania Eps-sąsiedztwa punktu P *metodą mediany*, odwiedzając węzeł indeksu metrycznego VP , oblicza odległość punktu obserwacyjnego od P , a następnie:

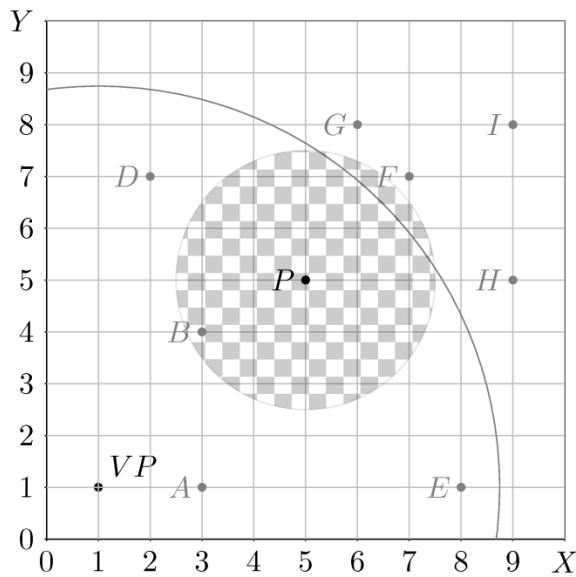
- przeszukuje lewe poddrzewo węzła VP indeksu metrycznego, jeśli $distance(VP, P) - Eps < mediana$ i $distance(VP, P) + Eps < mediana$;

- przeszukuje prawe poddrzewo węzła VP indeksu metrycznego, jeśli $distance(VP, P) + Eps \geq mediana$ i $distance(VP, P) - Eps \geq mediana$;
- przeszukuje lewe i prawe poddrzewo węzła VP indeksu metrycznego, jeśli $distance(VP, P) - Eps < mediana$ i $distance(VP, P) + Eps \geq mediana$;

Zatem, w przypadku przedstawionym na rys. 23 *metoda mediany* przeszukuje zarówno lewe jak i prawe poddrzewo mimo, że żaden z punktów prawego poddrzewa (F, G, H, I) nie należy do Eps-sąsiadztwa punktu P .



Rys. 22. Porównanie wydajności algorytmu kNN-Index-Vp-Tree w zależności od implementacji metody przeszukiwania indeksu metrycznego przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań $k=5$ sąsiadów w przykładowych zbiorach dla 10% losowo wybranych punktów zbioru danych



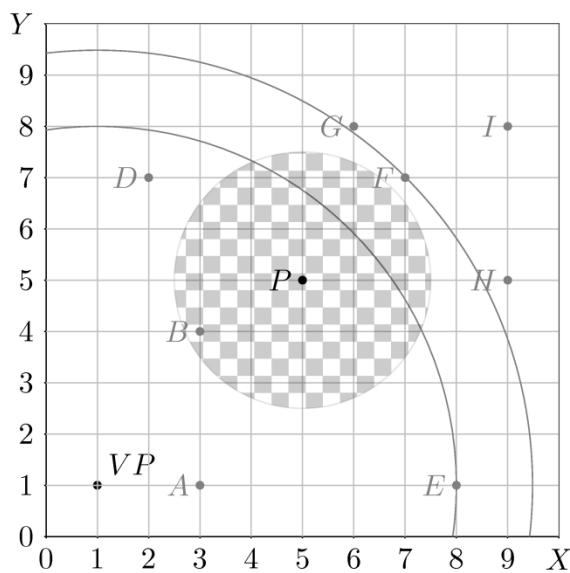
Rys. 23. Ilustracja do metody wyszukiwania k-sąsiadów w indeksie metrycznym z zastosowaniem mediany

Rys. 24 przedstawia sytuację znajdowania Eps-sąsiedztwa punktu P (obszar pokryty czarno-białą kratą) w węźle VP indeksu metrycznego *metodą ograniczeń*. Fragmentami okrągów zostały oznaczone ograniczenie górne bhl i ograniczenie dolne blh . Algorytm znajdowania Eps-sąsiedztwa punktu P *metodą ograniczeń* odwiedzając węzeł VP indeksu oblicza odległość $distance(VP, P)$ a następnie:

- przeszukuje lewe poddrzewo węzła VP indeksu metrycznego, jeśli $distance(VP, P) - Eps \leq blh$ i $distance(VP, P) + Eps < bhl$;
- przeszukuje prawe poddrzewo węzła VP indeksu metrycznego, jeśli $distance(VP, P) + Eps \geq bhl$ i $distance(VP, P) - Eps > blh$;
- przeszukuje lewe i prawe poddrzewo węzła VP indeksu metrycznego, jeśli $distance(VP, P) + Eps \geq bhl$ i $distance(VP, P) - Eps \leq blh$;

Zatem w przypadku przedstawionym na Rys. 24 *metoda ograniczeń* przeszuka jedynie lewe poddrzewo indeksu metrycznego.

Metoda ograniczeń szybciej znajduje k sąsiedztwo niż *metoda mediany*, ponieważ znajomość blh i bhl pozwala w pewnych przypadkach wcześniej zdecydować o braku konieczności eksploracji aktualnie rozpatrywanej gałęzi indeksu metrycznego niż *metoda mediany*.



Rys. 24. Ilustracja do metody wyszukiwania k-sąsiadów w indeksie metrycznym z zastosowaniem ograniczeń

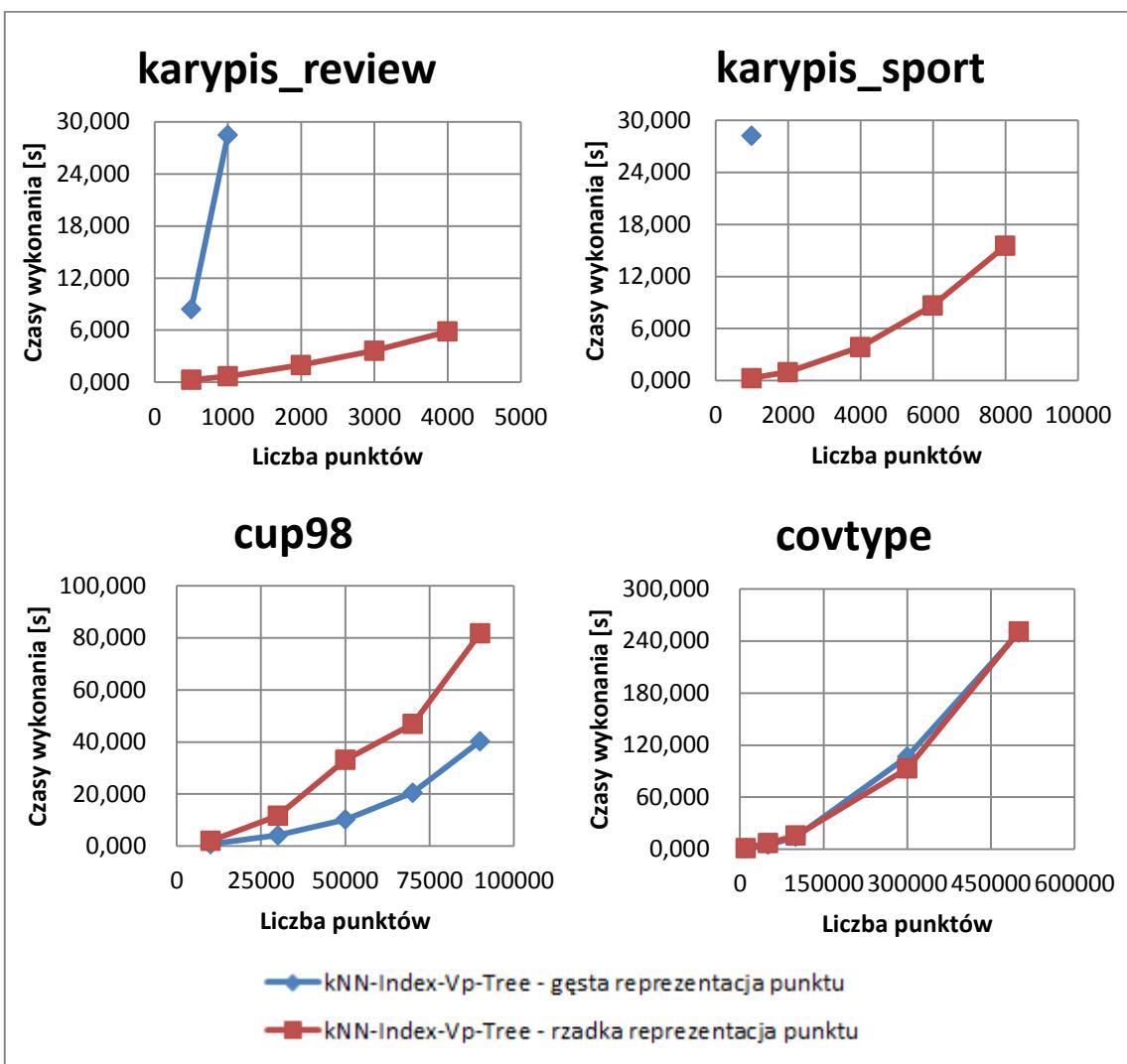
6.3.2. Implementacja struktury punktu

Podobnie jak w przypadku algorytmu *TI-k-Neighborhood-Index* przetestowałem wydajność *kNN-Index-Vp-Tree* w zależności od implementacji punktu. Tak jak poprzednio, zbadałem wpływ wykorzystania implementacji punktu gęstego oraz rzadkiego na wydajność algorytmu wyznaczania k sąsiedztwa. W tab. 17 zamieściłem wyniki uruchomień algorytmu *kNN-Index-Vp-Tree* z użyciem implementacji zarówno gęstego jak i rzadkiego punktu. Na rys. 25 znajdują się wykresy czasu wykonania algorytmów w funkcji liczby punktów.

Wnioski płynące z wykonanych badań są analogiczne do tych uzyskanych dla *TI-k-Neighborhood-Index* w rozdziale 6.2.1.2. Dlatego, w dalszej części pracy jako rezultaty eksperymentów algorytmu *kNN-Index-Vp-Tree* przeprowadzanych na danych tekstowych będą prezentowane wyniki uzyskane dla implementacji punktu rzadkiego, a dla przeprowadzonych na danych gęstych o niewielkim wymiarze będą prezentowane wyniki uzyskane przy zastosowaniu implementacji punktu gęstego.

Tab. 17. Porównanie wydajności algorytmu kNN-Index-Vp-Tree w zależności od implementacji punktu przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiadów w przykładowych zbiorach dla 10% losowo wybranych punktów zbioru danych

| zbior | I.p. | kNN-Index-Vp-Tree | | | kNN-Index-Vp-Tree | | |
|---------------|--------|----------------------------|---------|--------|-----------------------------|---------|--------|
| | | gęsta reprezentacja punktu | | | rzadka reprezentacja punktu | | |
| | | wyk. | wysz. | bud. | wyk. | wysz. | bud. |
| karypis_sport | 1000 | 28,215 | 22,407 | 5,808 | 0,296 | 0,218 | 0,078 |
| | 2000 | - | - | - | 0,988 | 0,822 | 0,166 |
| | 4000 | - | - | - | 3,879 | 3,510 | 0,369 |
| | 6000 | - | - | - | 8,684 | 8,107 | 0,577 |
| | 8000 | - | - | - | 15,543 | 14,768 | 0,774 |
| | 500 | 8,408 | 5,673 | 2,735 | 0,270 | 0,078 | 0,192 |
| | 1000 | 28,465 | 22,657 | 5,808 | 0,675 | 0,296 | 0,379 |
| | 2000 | - | - | - | 1,986 | 1,326 | 0,660 |
| | 3000 | - | - | - | 3,630 | 2,949 | 0,681 |
| | 4000 | - | - | - | 5,840 | 5,247 | 0,593 |
| covtype | 10000 | 0,859 | 0,298 | 0,561 | 1,181 | 0,416 | 0,765 |
| | 50000 | 5,010 | 2,241 | 2,770 | 6,973 | 3,000 | 3,973 |
| | 100000 | 14,204 | 7,703 | 6,501 | 15,558 | 8,574 | 6,984 |
| | 300000 | 106,525 | 70,103 | 36,422 | 93,065 | 65,031 | 28,034 |
| | 500000 | 249,375 | 180,813 | 68,562 | 250,567 | 179,249 | 71,318 |
| | 10000 | 0,728 | 0,307 | 0,421 | 2,044 | 1,388 | 0,656 |
| | 30000 | 4,178 | 2,812 | 1,366 | 11,690 | 9,537 | 2,153 |
| | 50000 | 10,215 | 7,766 | 2,450 | 33,156 | 29,411 | 3,744 |
| | 70000 | 20,500 | 16,790 | 3,710 | 46,935 | 41,204 | 5,731 |
| | 90000 | 40,305 | 27,149 | 13,156 | 81,750 | 73,710 | 8,039 |
| cup98 | 10000 | 0,728 | 0,307 | 0,421 | 2,044 | 1,388 | 0,656 |
| | 30000 | 4,178 | 2,812 | 1,366 | 11,690 | 9,537 | 2,153 |
| | 50000 | 10,215 | 7,766 | 2,450 | 33,156 | 29,411 | 3,744 |
| | 70000 | 20,500 | 16,790 | 3,710 | 46,935 | 41,204 | 5,731 |
| | 90000 | 40,305 | 27,149 | 13,156 | 81,750 | 73,710 | 8,039 |



Rys. 25. Porównanie wydajności algorytmu kNN-Index-Vp-Tree w zależności od implementacji punktu przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań $k=5$ sąsiadów w przykładowych zbiorach dla 10% losowo wybranych punktów zbioru danych

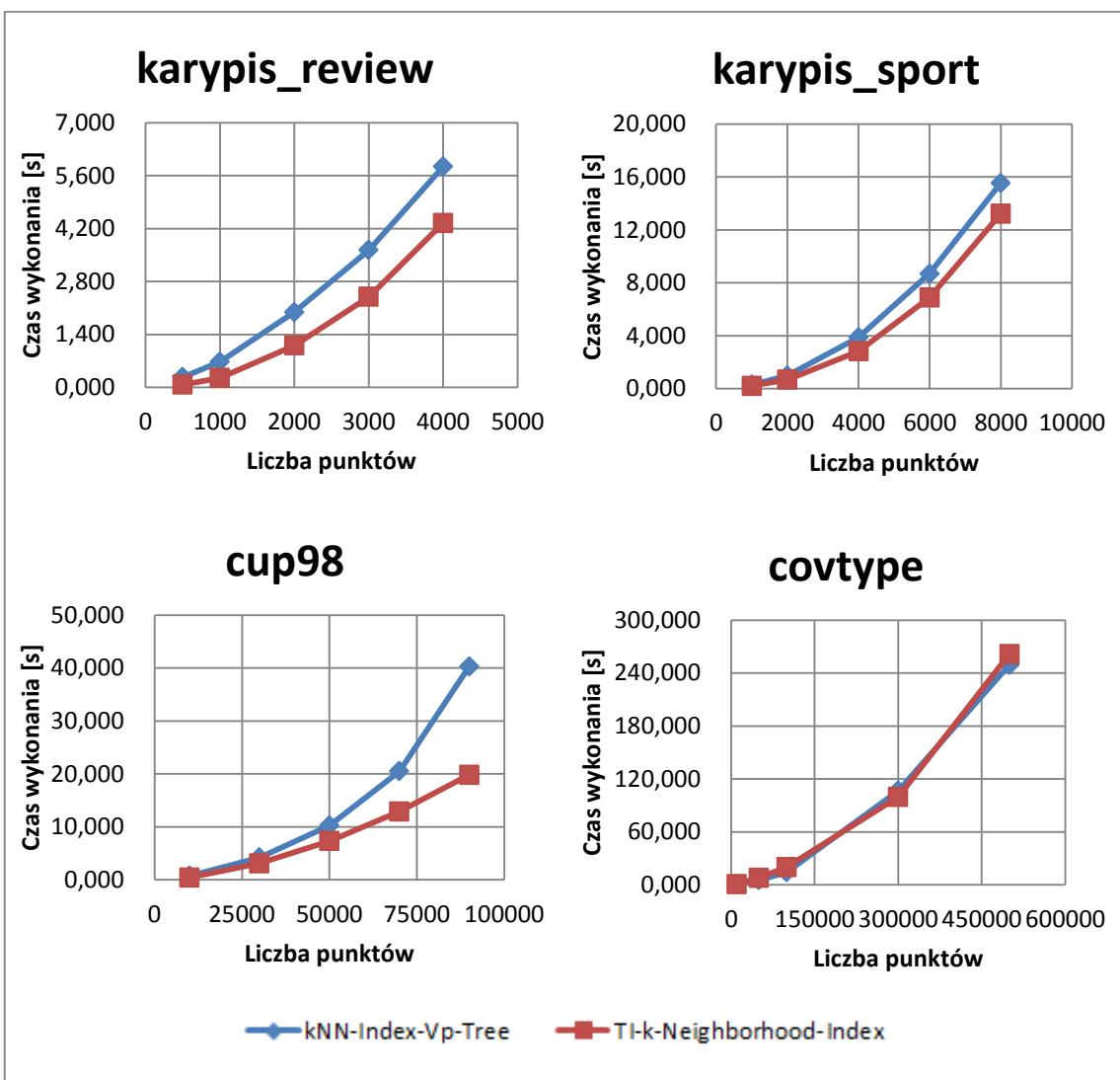
6.4. Porównanie algorytmów TI-k-Neighborhood-Index z kNN-Index-Vp-Tree

Na przykładowych zbiorach danych porównałem algorytmy *TI-k-Neighborhood-Index* oraz *kNN-Index-Vp-Tree*. W tab. 18 i na rys. 26 zamieściłem czasy wyznaczania k sąsiedztwa przez oba algorytmy.

Tab. 18. Porównanie wydajności algorytmów TI-k-Neighborhood-Index i kNN-Index-Vp-Tree przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiadów i k=5 sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

| zbior | I. p. | kNN-Index-Vp-Tree | | | | TI-k-Neighborhood-Index | | | |
|----------------|--------|-------------------|---------|--------|---------|-------------------------|-------|--------|-------|
| | | wyk. | wysz. | bud. | wyk. | wysz. | bud. | obl. | sort. |
| | | alg. | sąsiad. | ind. | alg. | sąsiad. | ind. | odl. | |
| karypis_sport | 1000 | 0,296 | 0,218 | 0,078 | 0,171 | 0,169 | 0,000 | 0,002 | 0,000 |
| | 2000 | 0,988 | 0,822 | 0,166 | 0,636 | 0,632 | 0,000 | 0,004 | 0,000 |
| | 4000 | 3,879 | 3,510 | 0,369 | 2,684 | 2,675 | 0,000 | 0,008 | 0,001 |
| | 6000 | 8,684 | 8,107 | 0,577 | 6,470 | 6,455 | 0,000 | 0,013 | 0,002 |
| | 8000 | 15,543 | 14,768 | 0,774 | 11,770 | 11,751 | 0,000 | 0,017 | 0,002 |
| karypis_review | 500 | 0,270 | 0,078 | 0,192 | 0,062 | 0,061 | 0,000 | 0,001 | 0,000 |
| | 1000 | 0,675 | 0,296 | 0,379 | 0,241 | 0,239 | 0,000 | 0,002 | 0,000 |
| | 2000 | 1,986 | 1,326 | 0,660 | 1,084 | 1,079 | 0,000 | 0,005 | 0,000 |
| | 3000 | 3,630 | 2,949 | 0,681 | 2,356 | 2,348 | 0,000 | 0,007 | 0,001 |
| | 4000 | 5,840 | 5,247 | 0,593 | 4,209 | 4,198 | 0,000 | 0,010 | 0,001 |
| covtype | 10000 | 0,859 | 0,298 | 0,561 | 0,551 | 0,370 | 0,000 | 0,179 | 0,002 |
| | 50000 | 5,010 | 2,241 | 2,770 | 7,804 | 5,526 | 0,001 | 2,262 | 0,015 |
| | 100000 | 14,204 | 7,703 | 6,501 | 19,962 | 15,025 | 0,001 | 4,901 | 0,035 |
| | 300000 | 106,525 | 70,103 | 36,422 | 93,868 | 78,047 | 0,003 | 15,686 | 0,132 |
| | 500000 | 249,375 | 180,813 | 68,562 | 246,482 | 219,916 | 0,005 | 26,312 | 0,249 |
| cup98 | 10000 | 0,728 | 0,307 | 0,421 | 0,388 | 0,259 | 0,000 | 0,126 | 0,003 |
| | 30000 | 4,178 | 2,812 | 1,366 | 2,785 | 2,105 | 0,001 | 0,669 | 0,010 |
| | 50000 | 10,215 | 7,766 | 2,450 | 6,942 | 5,287 | 0,001 | 1,635 | 0,019 |
| | 70000 | 20,500 | 16,790 | 3,710 | 13,095 | 9,933 | 0,001 | 3,132 | 0,029 |
| | 90000 | 40,305 | 27,149 | 13,156 | 19,041 | 14,683 | 0,001 | 4,316 | 0,041 |

Z rezultatów przeprowadzonych badań wynika, że algorytm *TI-k-Neighborhood-Index* wyszukuje k sąsiedztwo szybciej niż *kNN-Index-Vp-Tree* wyznacza k sąsiadów. Wyjątek stanowi zbiór covtype, w którego przypadku oba algorytmy wykonują się w zbliżonym czasie. Z przebiegów wykresów na rys. 26 dla zbiorów różnych od covtype można wnioskować, że *TI-k-Neighborhood-Index* będzie wykonywał się tym szybciej niż algorytm *kNN-Index-Vp-Tree*, im większy będzie zbiór, na którym działają te algorytmy.



Rys. 26. Porównanie wydajności algorytmów TI-k-Neighborhood-Index i kNN-Index-Vp-Tree przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań $k=5$ sąsiadów i $k=5$ sąsiedztwa w przykładowych zbiorach danych dla 10% losowo wybranych punktów zbioru danych

6.5. Badania algorytmu DBSCAN

W następującym rozdziale przedstawiłem rezultaty badań odmian algorytmów gęstościowego grupowania *DBSCAN* stosujących nierówność trójkąta lub rzutowanie w celu oszacowania odległości między punktami. Każdy z podrozdziałów skupia się na jednej z odmian algorytmu lub porównuje uprzednio zbadane. Ostatni podrozdział porównuje wszystkie zbadane odmiany DBSCAN.

Algorytmy *TI-DBSCAN*, *TI-DBSCAN-REF* oraz *DBSCAN-PROJECTION* zakładają, że punkty grupowanego zbioru D , które zaklasyfikowano do pewnego klastra C przenoszone są z D do zbioru sklasyfikowanych punktów D' . W swoich implementacjach zbiór punktów, na którym wykonywane jest grupowanie, przechowuję w postaci wektora, ponieważ zapewnia on szybki sekwencyjny dostęp do danych i pozwala usprawnić przetwarzanie wstępne (ang. *preprocessing*). Wadą tego podejścia, w kontekście rozpatrywanych algorytmów, jest czasochłonność procesu usuwania punktu z wektora, które wykonywane jest tyle razy, ile punktów zawiera zbiór D . Dlatego na potrzeby sprawnej implementacji przenoszenia punktu z D do D' , utrzymuję dwa indeksy odpowiadające zbiorom D i D' będące listami iteratorów na elementy wektora punktów, na których wykonywane jest grupowanie.

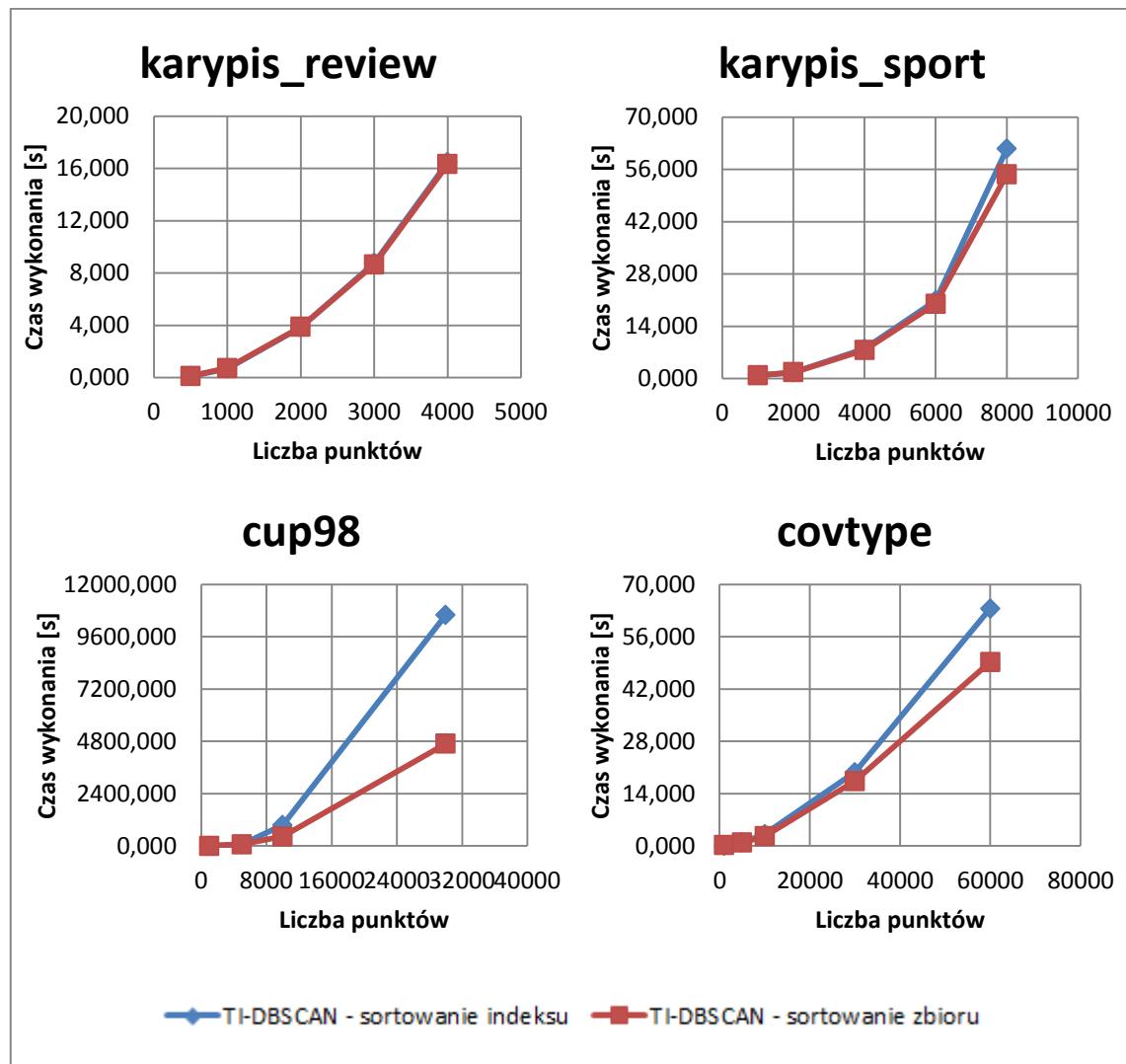
6.5.1. Badania algorytmu TI-DBSCAN

W kolejnych podrozdziałach w pierwszej kolejności skupiłem się na badaniu czasu wykonania *TI-DBSCAN* w zależności od implementacji algorytmu oraz w zależności od implementacji punktu. Następnie, w oparciu o uzyskane rezultaty eksperymentów, testowałem wydajność algorytmu w zależności od wartości promieniu sąsiedztwa Eps . W ostatnim kroku badałem wpływ wyboru punktu referencyjnego na czas wykonania algorytmu.

6.5.1.1. Implementacja algorytmu

Na przykładowych zbiorach danych przetestowałem dwie wersje implementacji algorytmu *TI-DBSCAN* korzystające ze struktury indeksu w celu dostępu do danych. Pierwsza implementacja najpierw sortowała bezpośrednio zbiór danych, a następnie budowała na nim indeks. W dalszej części pracy wersję tą będę nazywał *implementacją z sortowaniem zbioru*. Druga implementacja w pierwszej kolejności na nieposortowanym zbiorze budowała indeks,

który następnie sortowała. Wersji tej nadałem nazwę *implementacja z sortowaniem indeksu*. W tab. 19 zamieściłem czasy wykonania algorytmów wraz ze składającymi się na nie krokami. Rys. 27 prezentuje wykresy czasu wykonania badanych implementacji w funkcji liczby punktów.



Rys. 27. Porównanie wydajności implementacji algorytmu TI-DBSCAN w zależności od użytej metody sortowania przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania grupowań z parametrami MinPts=5 oraz Eps=0,05*przekątna danego zbioru

Tab. 19. Porównanie wydajności implementacji algorytmu TI-DBSCAN w zależności od użytej metody sortowania przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania grupowań z parametrami MinPts=5 oraz Eps=0,05*przekątna danego zbioru

Zgodnie ze szczegółowymi danymi z tab. 19 w implementacji z sortowaniem indeksu sortowanie wykonuje się do dwóch rzędów wielkości szybciej niż w implementacji z sortowaniem zbioru. Różnica w czasach sortowania między obiema implementacjami została wyjaśniona w rozdziale 6.2.1.1.

Jednakże bezpośrednie sortowanie zbioru danych nie jest wolniejsze od sortowania indeksu bardziej niż różnica czasu wykonania grupowania wynikająca z dostępu do danych. Dostęp przez posortowany indeks jest wolniejszy niż bezpośredni dostęp do posortowanego zbioru, ponieważ przeglądanie kolejnych elementów powoduje „skakanie” po pamięci. W kolejnych rozważaniach będę odnosił się do algorytmu *TI-DBSCAN* jako do implementacji z bezpośrednim dostępem do zbioru danych.

6.5.1.2. Implementacja struktury punktu

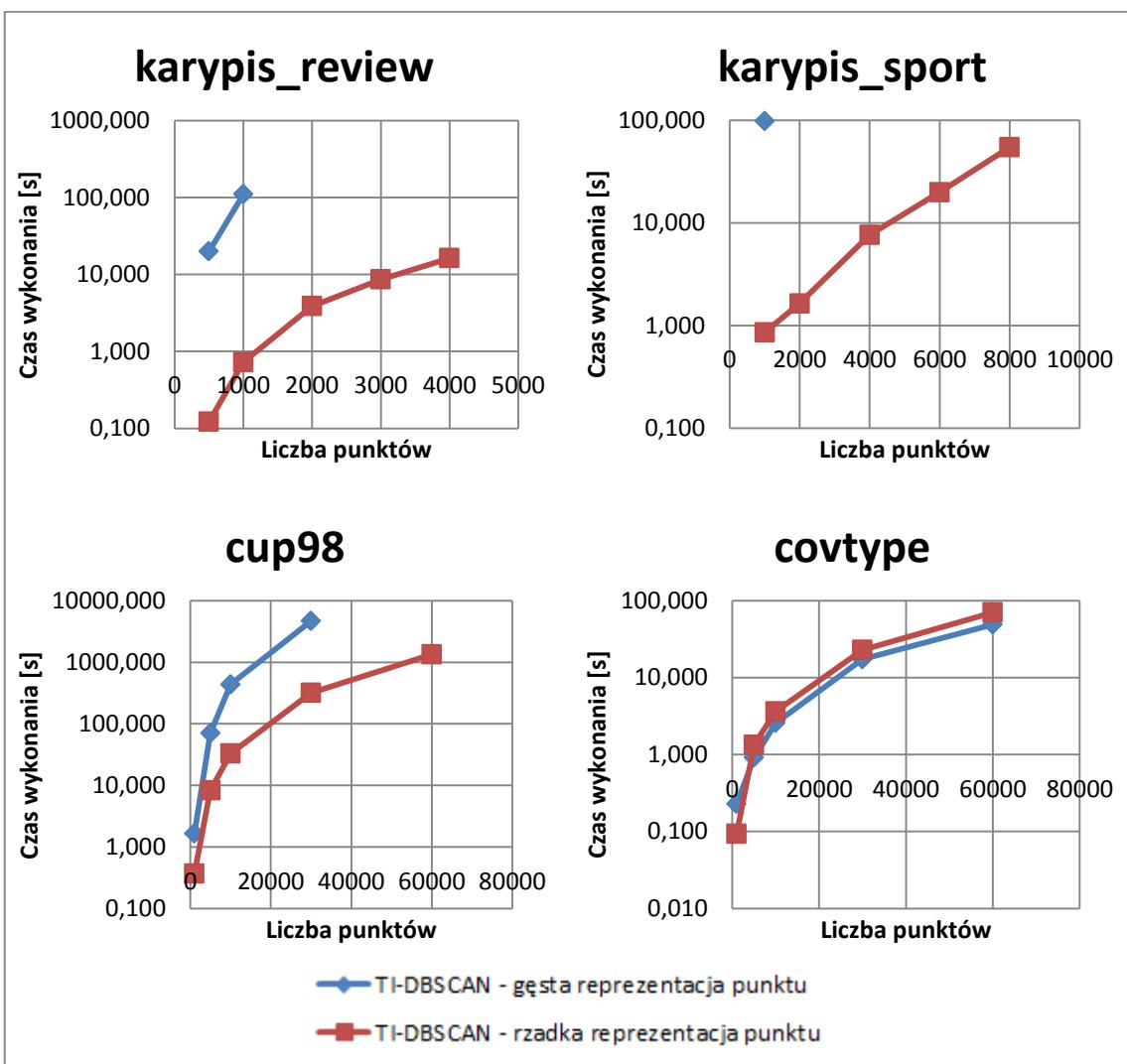
Podobnie jak w przypadku algorytmu *TI-k-Neighborhood-Index* i *kNN-Index-Vp-Tree* w niniejszym rozdziale zbadałem wydajność *TI-DBSCAN* w zależności od implementacji struktury punktu. W tab. 20 umieściłem czasy wykonania algorytmu wraz ze składającymi się na niego krokami. Na rys. 28 zaprezentowałem wykresy czasów wykonania algorytmu *TI-DBSCAN* z użyciem zarówno gęstej jak i rzadkiej implementacji punktu w funkcji liczby punktów.

Przyglądając się wynikom eksperymentów z tab. 20 warto zwrócić uwagę na anomalię zbioru covtype, w przypadku którego sprawniej wykonywały się uruchomienia wykorzystujące rzadką reprezentację punktu. Jednakże różnica w czasie wykonania algorytmu dla tego zbioru jest na tyle mała, że w świetle pozostałych wyników w dalszych rozważaniach nie będę brał pod uwagę rezultatów zebranych dla tego zbioru. Podobnie jak w przypadku badań przeprowadzonych w rozdziałach 6.2.1.2. i 6.3.2., z powodu wyczerpania pamięci RAM nie udało się uzyskać rezultatów badań dla implementacji gęstego punktu na tekstowych zbiorach danych gdy liczba punktów przewyższała 1000.

Wnioski z uruchomień algorytmu dla obu implementacji punktu nie odbiegają od tych uzyskanych w rozdziale 6.2.1.2. Dlatego w dalszej części pracy badania algorytmu *TI-DBSCAN* na danych tekstowych będą wykonywane z użyciem implementacji punktu rzadkiego, a działające z danymi gęstymi o niewielkim wymiarze będą wykonywane z zastosowaniem implementacji punktu gęstego.

Tab. 20. Porównanie wydajności algorytmu TI-DBSCAN w zależności od implementacji punktu przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania grupowań z parametrami MinPts=5 oraz Eps=0,05*przekątna danego zbioru

| zbior | I. p. | TI-DBSCAN gęsta reprezentacja punktu | | | | TI-DBSCAN rzadka reprezentacja punktu | | | |
|----------------|-------|--------------------------------------|----------|-------|-------|---------------------------------------|----------|-------|-------|
| | | EPS=0,05 | | | | EPS=0,05 | | | |
| | | wyk. | grup. | obl. | sort. | wyk. | grup | obl. | sort |
| | | alg. | | odl. | | alg. | | odl. | |
| karypis_sport | 1000 | 98,476 | 95,095 | 0,235 | 3,146 | 0,856 | 0,836 | 0,003 | 0,017 |
| | 2000 | - | - | - | - | 1,639 | 1,592 | 0,004 | 0,044 |
| | 4000 | - | - | - | - | 7,643 | 7,519 | 0,009 | 0,114 |
| | 6000 | - | - | - | - | 19,960 | 19,581 | 0,011 | 0,369 |
| | 8000 | - | - | - | - | 54,679 | 54,193 | 0,019 | 0,466 |
| | 500 | 19,938 | 18,494 | 0,115 | 1,329 | 0,122 | 0,104 | 0,002 | 0,016 |
| karypis_review | 1000 | 110,293 | 106,999 | 0,234 | 3,060 | 0,727 | 0,679 | 0,003 | 0,045 |
| | 2000 | - | - | - | - | 3,877 | 3,802 | 0,005 | 0,070 |
| | 3000 | - | - | - | - | 8,655 | 8,492 | 0,007 | 0,156 |
| | 4000 | - | - | - | - | 16,326 | 16,093 | 0,010 | 0,223 |
| | 1000 | 0,229 | 0,016 | 0,011 | 0,203 | 0,093 | 0,031 | 0,000 | 0,062 |
| | 5000 | 0,916 | 0,780 | 0,010 | 0,125 | 1,345 | 1,033 | 0,119 | 0,193 |
| covtype | 10000 | 2,579 | 2,298 | 0,011 | 0,270 | 3,645 | 3,447 | 0,073 | 0,125 |
| | 30000 | 17,368 | 15,985 | 0,021 | 1,362 | 22,942 | 22,490 | 0,031 | 0,422 |
| | 60000 | 49,256 | 46,744 | 0,047 | 2,465 | 70,475 | 69,493 | 0,052 | 0,931 |
| | 1000 | 1,654 | 1,475 | 0,003 | 0,175 | 0,369 | 0,344 | 0,001 | 0,023 |
| | 5000 | 70,735 | 60,992 | 0,115 | 9,629 | 8,310 | 8,231 | 0,007 | 0,072 |
| | 10000 | 437,990 | 430,565 | 0,419 | 7,006 | 33,075 | 32,904 | 0,014 | 0,156 |
| cup98 | 30000 | 4696,271 | 4694,960 | 0,023 | 1,288 | 319,363 | 318,786 | 0,047 | 0,530 |
| | 60000 | - | - | - | - | 1345,572 | 1344,310 | 0,085 | 1,177 |



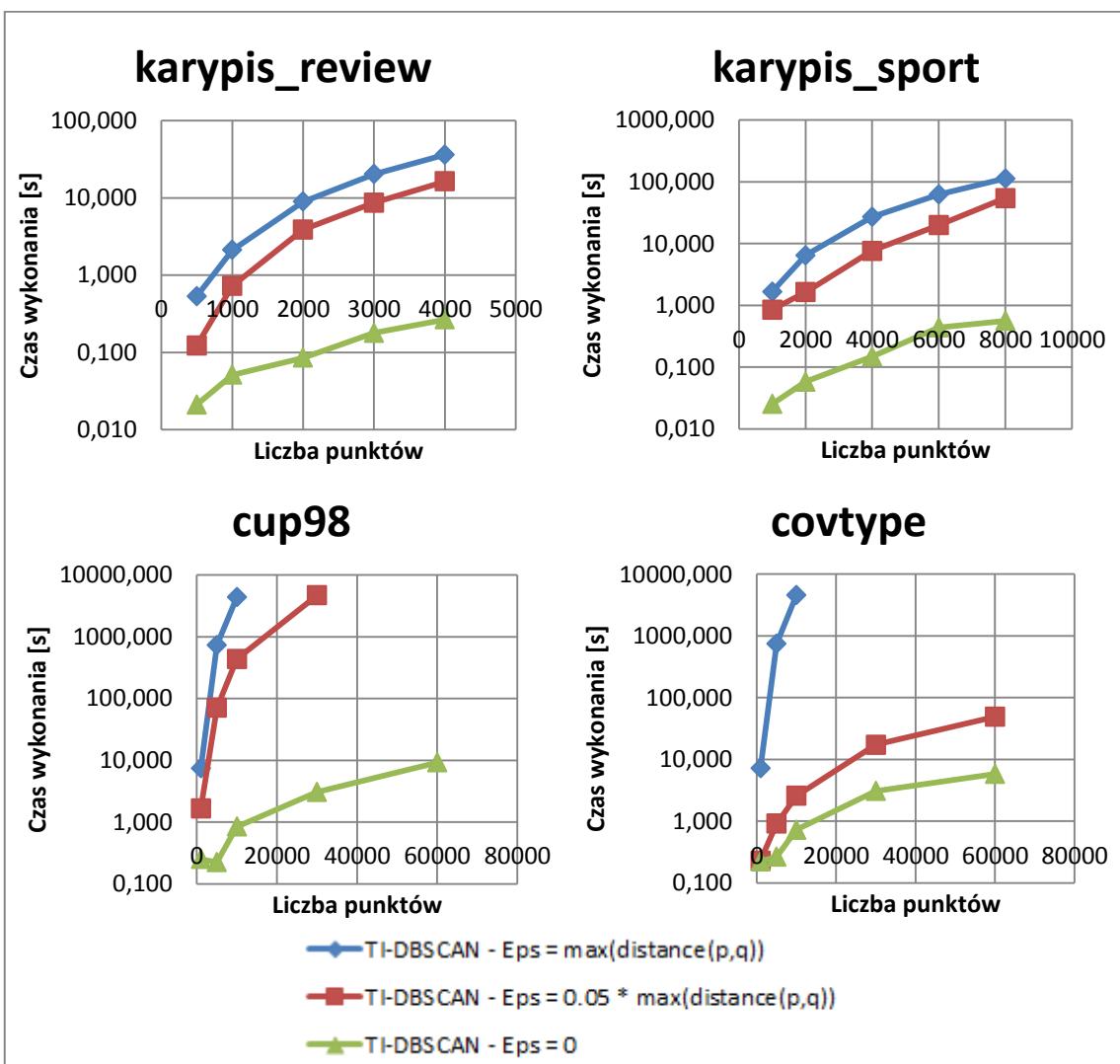
Rys. 28. Porównanie wydajności algorytmu TI-DBSCAN w zależności od implementacji punktu przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania grupowań z parametrami MinPts=5 oraz Eps=0,05*przekątna danego zbioru

6.5.1.3. Wybór parametru Eps

W celu stosownego przeprowadzenia dalszych eksperymentów z udziałem algorytmu TI-DBSCAN i jego odmian zbadałem wpływ dobrania parametru Eps na wydajność algorytmu. W tab. 21 zamieściłem czasy wykonania uruchomień kolejnych kroków algorytmu *TI-DBSCAN*, o punkcie referencyjnym równym [max], dla wartości parametrów MinPts=5 i Eps równe 0, 1/20 przekątnej dziedziny i przekątnej dziedziny. Na rys. 29 zamieściłem wykresy czasów wykonania algorytmu w funkcji liczby punktów.

Tab. 21. Porównanie wydajności algorytmu TI-DBSCAN w zależności od wartości parametru Eps przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania grupowań z parametrami MinPts=5 oraz Eps=[1; 0,05; 0]*przekątna danego zbioru

| zbior | l. p. | TI-DBSCAN EPS=MAX | | | | TI-DBSCAN EPS=0,05*MAX | | | | TI-DBSCAN EPS=0 | | | |
|----------------|-------|-------------------|----------|-------|-------|------------------------|----------|-------|-------|-----------------|-------|-------|-------|
| | | wyk. | grup. | obl. | sort. | wyk. | grup. | obl. | sort. | wyk. | grup. | obl. | sort. |
| | | alg. | | odl. | | alg. | | odl. | | alg. | | odl. | |
| karypis_sport | 1000 | 1,670 | 1,643 | 0,006 | 0,021 | 0,856 | 0,836 | 0,003 | 0,017 | 0,025 | 0,004 | 0,004 | 0,018 |
| | 2000 | 6,494 | 6,447 | 0,003 | 0,044 | 1,639 | 1,592 | 0,004 | 0,044 | 0,058 | 0,009 | 0,003 | 0,046 |
| | 4000 | 27,190 | 27,067 | 0,008 | 0,115 | 7,643 | 7,519 | 0,009 | 0,114 | 0,148 | 0,024 | 0,008 | 0,116 |
| | 6000 | 62,667 | 62,286 | 0,011 | 0,369 | 19,960 | 19,581 | 0,011 | 0,369 | 0,439 | 0,045 | 0,011 | 0,383 |
| | 8000 | 113,602 | 113,117 | 0,019 | 0,466 | 54,679 | 54,193 | 0,019 | 0,466 | 0,564 | 0,072 | 0,019 | 0,473 |
| | 500 | 0,532 | 0,515 | 0,001 | 0,016 | 0,122 | 0,104 | 0,002 | 0,016 | 0,021 | 0,002 | 0,002 | 0,017 |
| karypis_review | 1000 | 2,117 | 2,071 | 0,003 | 0,043 | 0,727 | 0,679 | 0,003 | 0,045 | 0,051 | 0,004 | 0,002 | 0,045 |
| | 2000 | 8,957 | 8,881 | 0,005 | 0,071 | 3,877 | 3,802 | 0,005 | 0,070 | 0,085 | 0,009 | 0,006 | 0,070 |
| | 3000 | 20,266 | 20,105 | 0,007 | 0,154 | 8,655 | 8,492 | 0,007 | 0,156 | 0,178 | 0,017 | 0,008 | 0,154 |
| | 4000 | 36,297 | 36,063 | 0,010 | 0,224 | 16,326 | 16,093 | 0,010 | 0,223 | 0,266 | 0,027 | 0,010 | 0,229 |
| | 1000 | 7,255 | 6,662 | 0,025 | 0,568 | 0,229 | 0,016 | 0,011 | 0,203 | 0,224 | 0,010 | 0,010 | 0,203 |
| | 5000 | 748,398 | 748,234 | 0,004 | 0,160 | 0,916 | 0,780 | 0,010 | 0,125 | 0,266 | 0,125 | 0,005 | 0,135 |
| covtype | 10000 | 4611,077 | 4610,710 | 0,007 | 0,360 | 2,579 | 2,298 | 0,011 | 0,270 | 0,728 | 0,458 | 0,010 | 0,260 |
| | 30000 | - | - | - | - | 17,368 | 15,985 | 0,021 | 1,362 | 3,077 | 1,695 | 0,027 | 1,355 |
| | 60000 | - | - | - | - | 49,256 | 46,744 | 0,047 | 2,465 | 5,819 | 3,344 | 0,042 | 2,433 |
| | 1000 | 7,432 | 6,826 | 0,026 | 0,581 | 1,654 | 1,475 | 0,003 | 0,175 | 0,250 | 0,016 | 0,012 | 0,223 |
| | 5000 | 733,815 | 733,654 | 0,004 | 0,158 | 70,735 | 60,992 | 0,115 | 9,629 | 0,225 | 0,129 | 0,008 | 0,088 |
| | 10000 | 4378,321 | 4378,110 | 0,007 | 0,204 | 437,990 | 430,565 | 0,419 | 7,006 | 0,842 | 0,530 | 0,000 | 0,312 |
| cup98 | 30000 | - | - | - | - | 4696,271 | 4694,960 | 0,023 | 1,288 | 3,068 | 2,044 | 0,016 | 1,009 |
| | 60000 | - | - | - | - | - | - | - | - | 9,214 | 6,453 | 0,046 | 2,715 |



Rys. 29. Porównanie wydajności algorytmu TI-DBSCAN w zależności od wartości parametru Eps przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania grupowań z parametrami MinPts=5 oraz Eps=[1; 0,05; 0]*przekątna danego zbioru

Na podstawie uzyskanych rezultatów zaobserwowałem, że dla Eps równego przekątnej dziedziny algorytm TI-DBSCAN wykonuje się naj wolniej, ponieważ wszystkie punkty dziedziny należą do otoczenia epsilonowego dowolnego punktu. W takim przypadku nierówność trójkąta nie przyspiesza procesu wyznaczania otoczenia epsilonowego, które musi zostać obliczone dla każdego punktu. Najbardziej selektywny przypadek następuje, gdy Eps jest równe 0. Z uwagi na najwyższą możliwą selektywność, nierówność trójkąta wyklucza z przestrzeni potencjalnych sąsiadów dużą liczbę punktów znakomicie przyczyniając się do zmniejszenia czasu wyznaczania otoczenia danego punktu. Dlatego dla danego MinPts

i dowolnego Eps większego od Eps_0=0 i mniejszego od przekątnej dziedziny Eps_max czas wykonania grupowania danego zbioru nie będzie wolniejszy od czasu grupowania dla Eps_max i nie będzie szybszy od czasu wykonania grupowania dla Eps_0. W zależności od charakterystyki zbioru czas ten może być bliższy rezultatom osiąganym dla Eps_max lub Eps_0. Przykładami takich sytuacji są eksperymenty wykonane dla Eps równego 1/20 przekątnej dziedziny. Dla większości eksperymentów czas wykonania algorytmu dla Eps = 0,05*Eps_max był mniejszy lecz bliższy temu dla Eps_max. Wyjątek stanowi zbiór covtype, dla którego czasy wykonania eksperymentów z Eps = 0,05*Eps_max są znacznie szybsze niż dla Eps_max. W dalszej części pracy badania algorytmu *TI-DBSCAN* będą dla Eps = 0,05*Eps_max.

6.5.1.4. Wybór punktu referencyjnego

W celu znalezienia stosownego punktu referencyjnego dla algorytmu *TI-DBSCAN* przeprowadziłem badania z różnymi ich przykładami. Podobnie jak w przypadku wyboru odpowiedniego punktu referencyjnego dla algorytmu *TI-k-Neighborhood-Index* skupiłem się na punktach takich jak:

- punkt maksymalny [max];
- punkt minimalny [min];
- punkt [max_min];
- punkt losowy [rand];

W tab. 22 i tab. 23 zamieściłem czasy uruchomień algorytmu *TI-DBSCAN* wraz z trwaniem składających się na niego kroków. Na rys. 30 znajdują się wykresy czasu wykonania algorytmu w funkcji liczby punktów.

Analiza wyników pozwala zauważyc, że dla większości przykładowych zbiorów minimalny punkt referencyjny najbardziej przyspiesza wykonanie algorytmu *TI-DBSCAN*. Wyjątek stanowi zbiór covtype, dla którego najlepiej sprawdza się punkt maksymalny. Rezultaty uzyskane dla tego zbioru z wykorzystaniem punktów referencyjnych [min] i [max_min] odbiegają charakterem od pozostałych wyników, dla których wraz ze wzrostem liczby punktów czas wykonania algorytmu jest dłuższy. Anomalia następuje dla liczby punktów równej 30000, dla której *TI-DBSCAN* wykonuje się szybciej niż dla 15000 punktów. Zaburzenie to wynika z charakteru zbioru oraz wybranych punktów referencyjnych. Z uwagi na specyficzną charakterystykę zbioru covtype, w dalszych rozważaniach na temat wyboru punktu

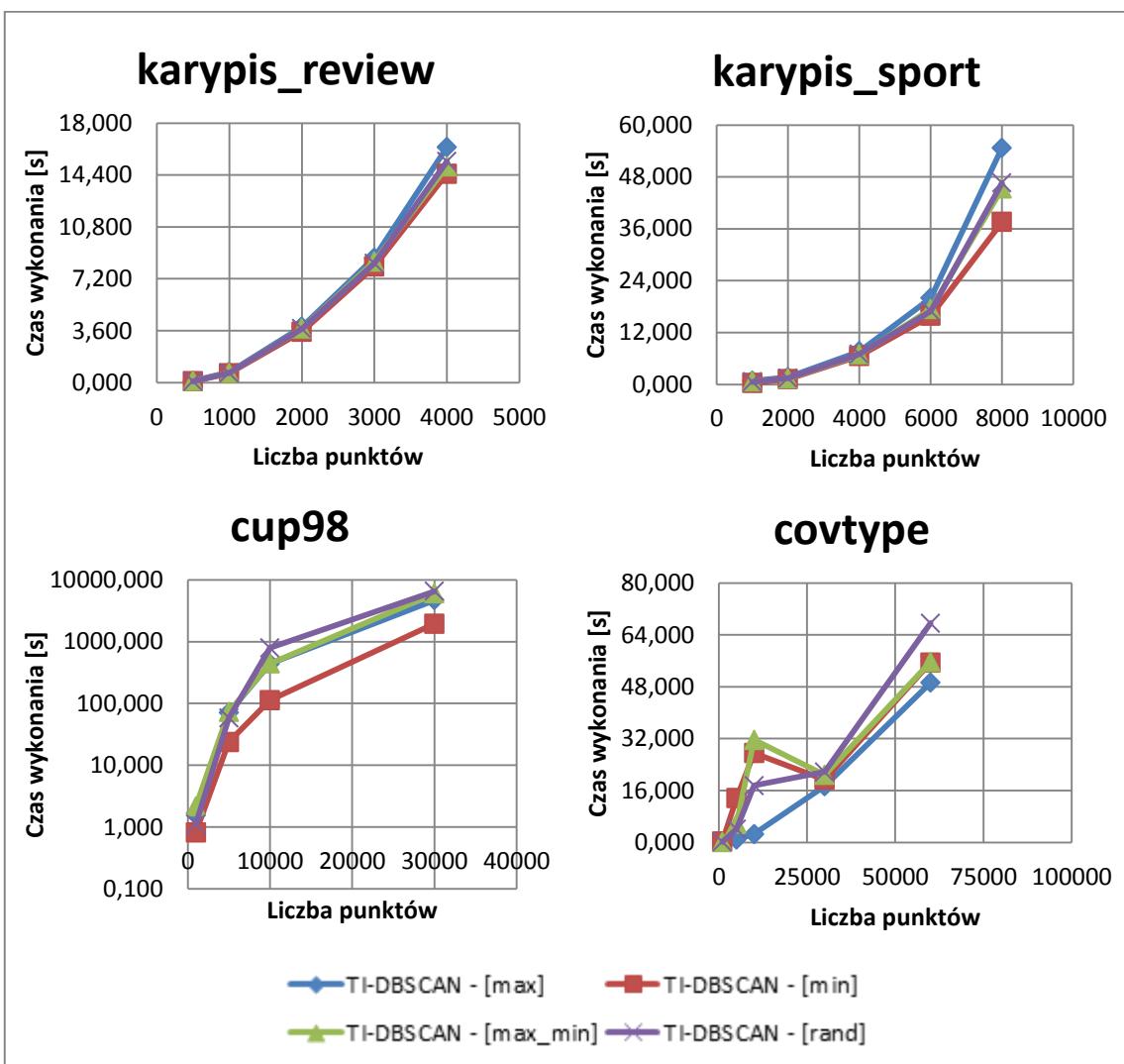
referencyjnego dla algorytmu TI-DBSCAN nie będę brał pod uwagę rezultatów uzyskanych dla tego zbioru.

Tab. 22. Porównanie wydajności algorytmu TI-DBSCAN w zależności od wybranego punktu referencyjnego [max] lub [min] przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania grupowań z parametrami MinPts=5 oraz Eps= 0,05*przekątna danego zbioru

| zbior | l. p. | TI-DBSCAN [max] | | | | TI-DBSCAN [min] | | | |
|----------------|-------|-----------------|----------|-------|-------|-----------------|----------|-------|-------|
| | | wyk. | grup. | obl. | sort. | wyk. | grup. | obl. | sort. |
| | | alg. | | odl. | | alg. | | odl. | |
| karypis_sport | 1000 | 0,856 | 0,836 | 0,003 | 0,017 | 0,303 | 0,281 | 0,002 | 0,018 |
| | 2000 | 1,639 | 1,592 | 0,004 | 0,044 | 1,224 | 1,184 | 0,003 | 0,038 |
| | 4000 | 7,643 | 7,519 | 0,009 | 0,114 | 6,564 | 6,429 | 0,007 | 0,129 |
| | 6000 | 19,960 | 19,581 | 0,011 | 0,369 | 15,836 | 15,489 | 0,010 | 0,317 |
| | 8000 | 54,679 | 54,193 | 0,019 | 0,466 | 37,548 | 36,914 | 0,013 | 0,623 |
| karypis_review | 500 | 0,122 | 0,104 | 0,002 | 0,016 | 0,091 | 0,075 | 0,001 | 0,015 |
| | 1000 | 0,727 | 0,679 | 0,003 | 0,045 | 0,645 | 0,597 | 0,002 | 0,046 |
| | 2000 | 3,877 | 3,802 | 0,005 | 0,070 | 3,531 | 3,451 | 0,004 | 0,077 |
| | 3000 | 8,655 | 8,492 | 0,007 | 0,156 | 8,057 | 7,878 | 0,006 | 0,166 |
| | 4000 | 16,326 | 16,093 | 0,010 | 0,223 | 14,487 | 14,243 | 0,008 | 0,240 |
| covtype | 1000 | 0,229 | 0,016 | 0,011 | 0,203 | 0,278 | 0,020 | 0,012 | 0,246 |
| | 5000 | 0,916 | 0,780 | 0,010 | 0,125 | 13,703 | 8,422 | 0,165 | 5,116 |
| | 10000 | 2,579 | 2,298 | 0,011 | 0,270 | 27,569 | 19,455 | 0,521 | 7,631 |
| | 30000 | 17,368 | 15,985 | 0,021 | 1,362 | 19,376 | 18,491 | 0,022 | 0,863 |
| | 60000 | 49,256 | 46,744 | 0,047 | 2,465 | 55,477 | 52,209 | 0,044 | 2,799 |
| cup98 | 1000 | 1,654 | 1,475 | 0,003 | 0,175 | 0,821 | 0,473 | 0,016 | 0,332 |
| | 5000 | 70,735 | 60,992 | 0,115 | 9,629 | 23,716 | 23,626 | 0,004 | 0,086 |
| | 10000 | 437,990 | 430,565 | 0,419 | 7,006 | 113,029 | 112,669 | 0,007 | 0,353 |
| | 30000 | 4696,271 | 4694,960 | 0,023 | 1,288 | 1945,815 | 1944,790 | 0,031 | 0,994 |
| | 60000 | - | - | - | - | - | - | - | - |

Tab. 23. Porównanie wydajności algorytmu TI-DBSCAN w zależności od wybranego punktu referencyjnego [max_min] lub [rand] przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania grupowań z parametrami MinPts=5 oraz Eps=0,05*przekątna danego zbioru

| zbior | I. p. | TI-DBSCAN [max_min] | | | | TI-DBSCAN [rand] | | | |
|----------------|-------|---------------------|----------|-------|--------|------------------|----------|-------|--------|
| | | wyk. | grup. | obl. | sort. | wyk. | grup. | obl. | sort. |
| | | alg. | | odl. | | alg. | | odl. | |
| karypis_sport | 1000 | 0,655 | 0,619 | 0,000 | 0,031 | 0,603 | 0,572 | 0,000 | 0,031 |
| | 2000 | 1,347 | 1,263 | 0,000 | 0,083 | 1,425 | 1,352 | 0,005 | 0,068 |
| | 4000 | 6,900 | 6,760 | 0,000 | 0,135 | 6,989 | 6,827 | 0,010 | 0,151 |
| | 6000 | 17,379 | 16,983 | 0,011 | 0,385 | 16,947 | 16,432 | 0,016 | 0,416 |
| | 8000 | 45,240 | 44,736 | 0,015 | 0,489 | 46,561 | 46,046 | 0,016 | 0,504 |
| | 500 | 0,109 | 0,094 | 0,000 | 0,016 | 0,094 | 0,083 | 0,000 | 0,016 |
| karypis_review | 1000 | 0,671 | 0,624 | 0,000 | 0,047 | 0,676 | 0,639 | 0,000 | 0,032 |
| | 2000 | 3,744 | 3,635 | 0,000 | 0,109 | 3,739 | 3,609 | 0,000 | 0,130 |
| | 3000 | 8,393 | 8,221 | 0,010 | 0,156 | 8,273 | 8,122 | 0,005 | 0,151 |
| | 4000 | 14,981 | 14,742 | 0,005 | 0,234 | 15,335 | 15,039 | 0,005 | 0,234 |
| | 1000 | 0,114 | 0,026 | 0,000 | 0,083 | 0,198 | 0,031 | 0,000 | 0,167 |
| | 5000 | 5,829 | 5,205 | 0,089 | 0,541 | 4,150 | 0,931 | 0,067 | 3,141 |
| covtype | 10000 | 31,632 | 24,368 | 0,374 | 6,895 | 17,566 | 4,098 | 0,280 | 13,942 |
| | 30000 | 20,810 | 19,952 | 0,026 | 0,910 | 21,746 | 20,920 | 0,016 | 0,842 |
| | 60000 | 55,671 | 53,071 | 0,047 | 2,345 | 67,590 | 64,610 | 0,047 | 2,174 |
| | 1000 | 2,152 | 2,017 | 0,000 | 0,130 | 1,113 | 0,962 | 0,016 | 0,135 |
| | 5000 | 72,805 | 71,063 | 0,078 | 1,674 | 57,747 | 54,283 | 0,078 | 3,271 |
| | 10000 | 440,898 | 430,020 | 0,343 | 10,514 | 791,618 | 783,194 | 0,265 | 8,008 |
| cup98 | 30000 | 6006,420 | 6005,800 | 0,026 | 0,889 | 6504,790 | 6500,170 | 0,026 | 1,305 |
| | 60000 | - | - | - | - | - | - | - | - |



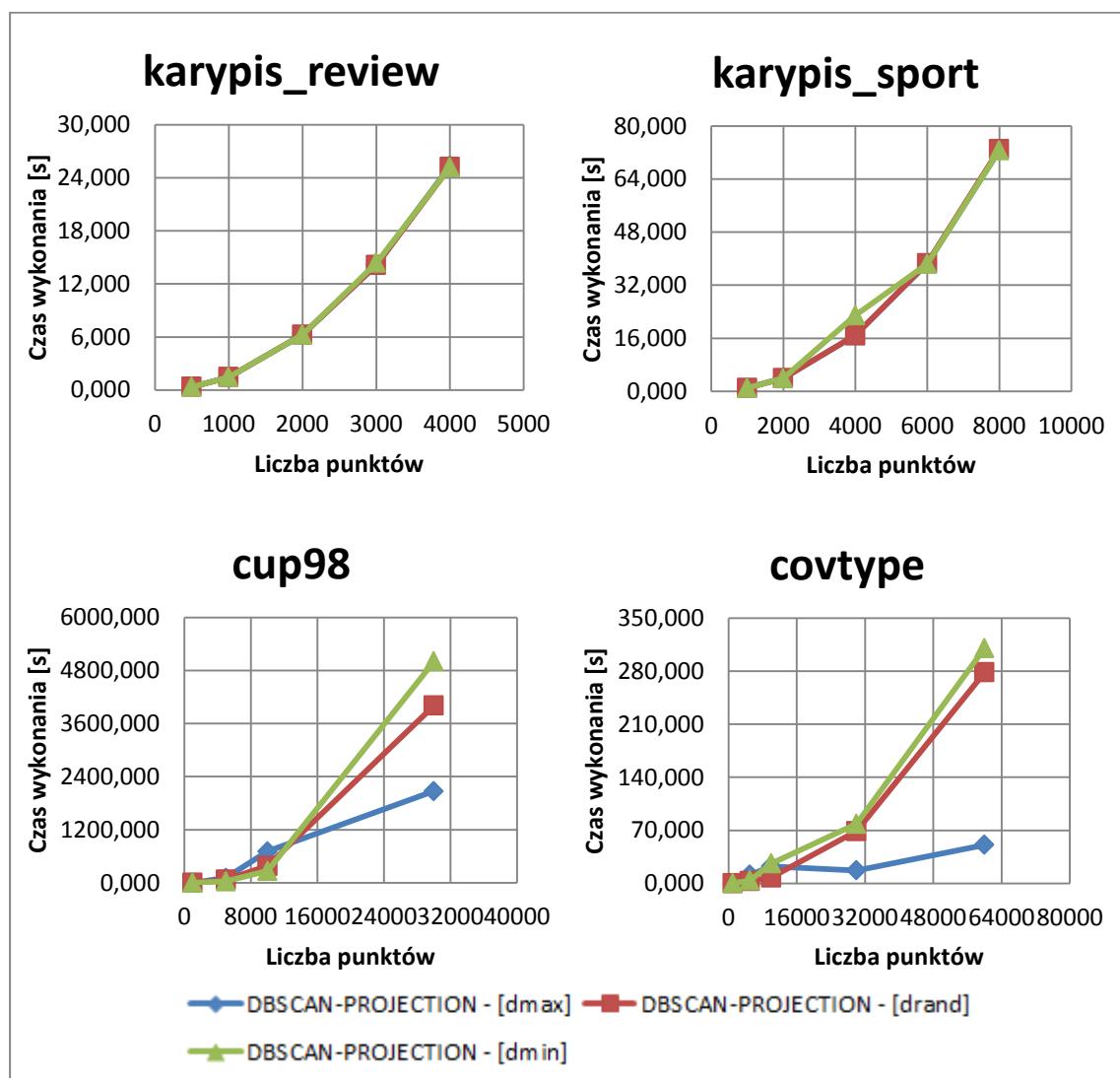
Rys. 30. Porównanie wydajności algorytmu TI-DBSCAN w zależności od wybranego punktu referencyjnego przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania grupowań z parametrami MinPts=5 oraz Eps= 0,05*przekątna danego zbioru

Biorąc pod uwagę uzyskane rezultaty, w dalszej części pracy jako wyniki czasowe algorytmu *TI-DBSCAN* będą prezentowane wartości otrzymane przy zastosowaniu punktu minimalnego jako punktu referencyjnego.

6.5.2. Badania algorytmu DBSCAN-PROJECTION

Podobnie jak w przypadku algorytmu *k-Neighborhood-Index-Projection*, opisanego w rozdziale 6.2.2., badania algorytmu *DBSCAN-PROJECTION* przeprowadziłem w kontekście rzutowania na [dmax], [dmin] oraz [drand];

Czasy uruchomień algorytmu *DBSCAN-PROJECTION* wraz z trwaniem składających się na niego kroków zamieściłem w tab. 24. Na rys. 31 znajdują się wykresy czasu wykonania algorytmu w funkcji liczby punktów.



Rys. 31. Porównanie wydajności algorytmu DBSCAN-PROJECTION w zależności od wybranego wymiaru rzutowania przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania grupowań z parametrami MinPts=5 oraz Eps= 0,05*przekątna danego zbioru

Tab. 24. Porównanie wydajności algorytmu DBSCAN-PROJECTION w zależności od wybranego wymiaru rzutowania przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania grupowań z parametrami MinPts=5 oraz Eps= 0,05*przekątna danego zbioru

| zbior | l. p. | DBSCAN-PROJECTION [dmax] | | | | DBSCAN-PROJECTION [drand] | | | | DBSCAN-PROJECTION [dmin] | | | |
|----------------|-------|--------------------------|----------|-------|--------|---------------------------|----------|-------|-------|--------------------------|----------|-------|-------|
| | | wyk. | grup. | obl. | sort. | wyk. | grup. | obl. | sort. | wyk. | grup. | obl. | sort. |
| | | alg. | | rz. | | alg. | | rz. | | alg. | | rz. | |
| karypis_sport | 1000 | 1,040 | 1,039 | 0,001 | 0,000 | 1,038 | 1,038 | 0,000 | 0,000 | 1,040 | 1,040 | 0,000 | 0,000 |
| | 2000 | 3,981 | 3,980 | 0,001 | 0,000 | 3,982 | 3,981 | 0,001 | 0,000 | 3,978 | 3,978 | 0,000 | 0,000 |
| | 4000 | 16,925 | 16,922 | 0,002 | 0,000 | 16,897 | 16,894 | 0,002 | 0,000 | 22,916 | 22,916 | 0,000 | 0,000 |
| | 6000 | 38,521 | 38,517 | 0,004 | 0,000 | 38,664 | 38,661 | 0,004 | 0,000 | 38,449 | 38,449 | 0,000 | 0,000 |
| | 8000 | 72,745 | 72,739 | 0,005 | 0,000 | 73,079 | 73,074 | 0,005 | 0,000 | 72,681 | 72,675 | 0,000 | 0,000 |
| | 500 | 0,366 | 0,365 | 0,000 | 0,000 | 0,365 | 0,365 | 0,000 | 0,000 | 0,364 | 0,364 | 0,000 | 0,000 |
| karypis_review | 1000 | 1,466 | 1,465 | 0,000 | 0,000 | 1,466 | 1,466 | 0,001 | 0,000 | 1,466 | 1,466 | 0,000 | 0,000 |
| | 2000 | 6,272 | 6,271 | 0,001 | 0,000 | 6,270 | 6,267 | 0,002 | 0,000 | 6,261 | 6,261 | 0,000 | 0,000 |
| | 3000 | 14,155 | 14,153 | 0,002 | 0,000 | 14,116 | 14,112 | 0,003 | 0,000 | 14,373 | 14,373 | 0,000 | 0,000 |
| | 4000 | 25,314 | 25,311 | 0,002 | 0,000 | 25,219 | 25,217 | 0,002 | 0,000 | 25,184 | 25,184 | 0,000 | 0,000 |
| | 1000 | 0,188 | 0,030 | 0,007 | 0,151 | 0,073 | 0,070 | 0,005 | 0,000 | 0,156 | 0,062 | 0,000 | 0,089 |
| | 5000 | 11,375 | 5,568 | 0,086 | 5,721 | 2,911 | 2,526 | 0,065 | 0,271 | 3,811 | 3,593 | 0,036 | 0,192 |
| covtype | 10000 | 22,420 | 9,860 | 0,366 | 12,194 | 7,747 | 6,724 | 0,224 | 0,800 | 26,213 | 18,881 | 0,166 | 7,165 |
| | 30000 | 17,144 | 16,294 | 0,020 | 0,981 | 68,816 | 68,671 | 0,020 | 0,097 | 78,474 | 78,011 | 0,021 | 0,468 |
| | 60000 | 50,843 | 48,323 | 0,039 | 2,195 | 278,390 | 273,248 | 0,039 | 0,501 | 310,243 | 309,162 | 0,036 | 1,050 |
| | 1000 | 0,897 | 0,824 | 0,006 | 0,067 | 0,857 | 0,797 | 0,005 | 0,052 | 0,572 | 0,525 | 0,000 | 0,047 |
| | 5000 | 109,947 | 106,240 | 0,120 | 5,125 | 71,474 | 70,102 | 0,074 | 1,343 | 32,053 | 31,674 | 0,042 | 0,333 |
| | 10000 | 708,618 | 708,037 | 0,306 | 3,543 | 381,858 | 378,088 | 0,243 | 2,577 | 268,757 | 265,715 | 0,203 | 2,834 |
| cup98 | 30000 | 2072,830 | 2071,820 | 0,020 | 0,988 | 4010,440 | 4010,220 | 0,019 | 0,268 | 5006,550 | 5006,350 | 0,026 | 0,203 |
| | 60000 | - | - | - | - | - | - | - | - | - | - | - | - |

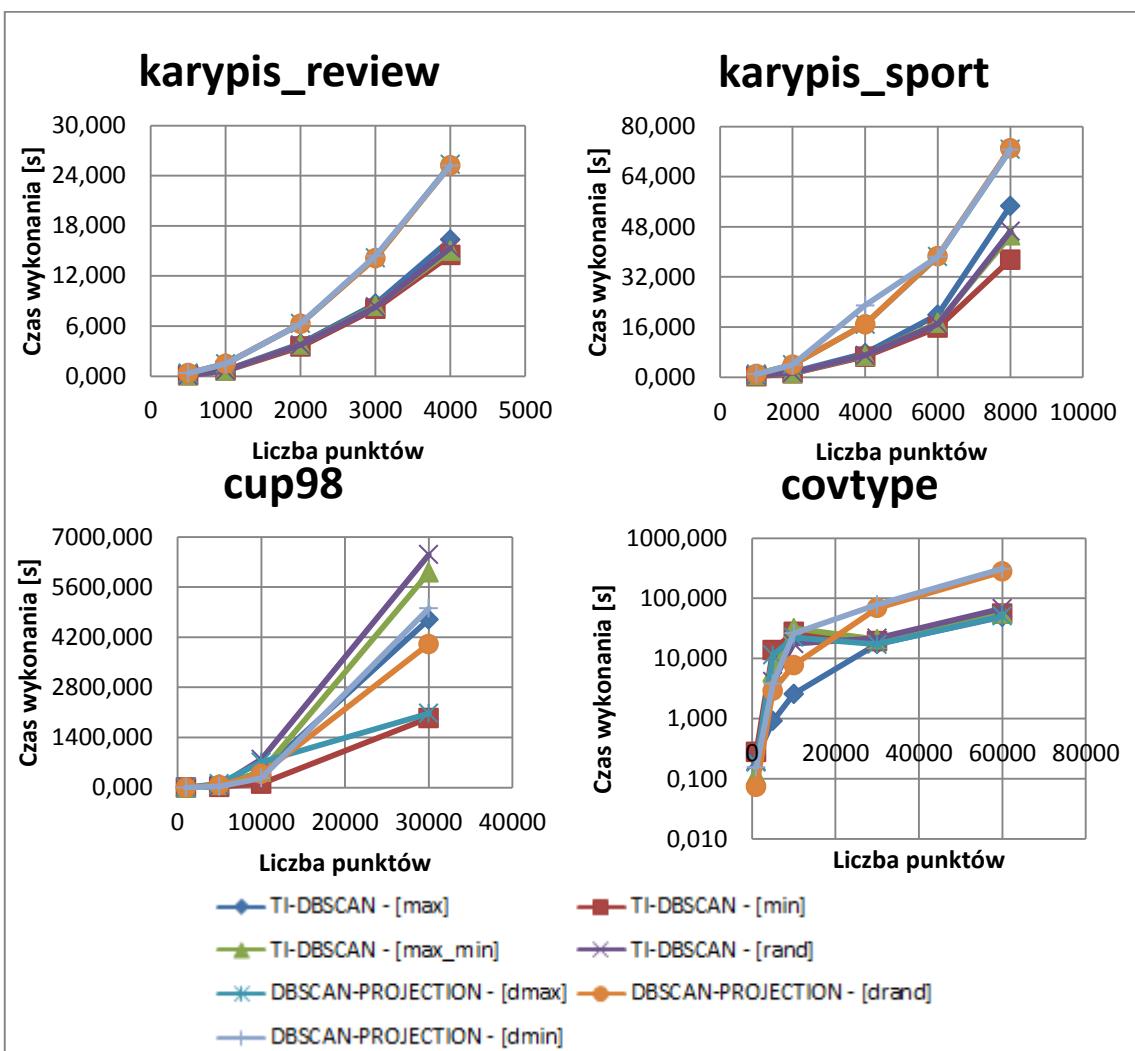
Różnice w czasach wykonania algorytmu uruchamianego dla różnych strategii rzutowania na zbiorach tekstowych są na tyle niewielkie, że nie pozwalają na wyciągnięcie wniosków na temat użyteczności danej metody. Rezultaty eksperymentów wykonanych na gęstych zbiorach danych świadczą, że rzutowanie na wymiar o największej dziedzinie pozwala na kilkukrotne przyspieszenie grupowania w porównaniu do pozostałych strategii. Liczność dziedziny wymiaru ma kluczowy wpływ na sprawność algorytmu *DBSCAN-PROJECTION*. Wpływ ten wyjaśniałem już w rozdziale 6.2.2.

Ponieważ w algorytmie *DBSCAN-PROJECTION* rzutowanie tym mocniej wspiera selektywność wyznaczania potencjalnych sąsiadów im liczniejsza jest dziedzina danego wymiaru, dlatego w dalszej części pracy jako wyniki czasowe algorytmu *DBSCAN-PROJECTION* będą prezentowane rezultaty osiągnięte przy zastosowaniu rzutu na wymiar o największej dziedzinie.

6.5.3. Porównanie algorytmów DBSCAN-PROJECTION i TI-DBSCAN

W celu porównania podejścia korzystającego z nierówności trójkąta z podejściem stosującym rzutowanie na wymiar, na rys. 32 zamieściłem wykresy czasów wykonania algorytmów *TI-DBSCAN* i *DBSCAN-PROJECTION* w funkcji liczby punktów we wszystkich badanych przypadkach zastosowania punktu referencyjnego i strategii rzutowania na wymiar. Dane zamieszczone na wykresach odpowiadają tym zgromadzonym w tabelach tab. 22, tab. 23 i tab. 24.

Z rezultatów eksperymentów wynika, że dla danych tekstowych algorytm wykorzystujący rzutowanie wykonuje się blisko dwukrotnie wolniej niż *TI-DBSCAN*. W przypadku gęstych zbiorów sytuacja nie jest już taka oczywista. Dla covtype rzutowanie na losowy wymiar i [dmin] sprawia, że grupowanie wykonuje się sześciokrotnie wolniej niż w pozostałych przypadkach. Co ciekawe, te same metody rzutowania zastosowane dla grupowania zbioru cup98 dają nie gorsze rezultaty niż *TI-DBSCAN*. Spośród wszystkich metod rzutowania, [dmax] najlepiej przyspiesza grupowanie, które w przypadku największych liczącości zbiorów gęstych uzyskuje rezultaty porównywalne z najlepszymi rezultatami algorytmu *TI-DBSCAN*. Jednakże, należy zwrócić uwagę, że na niekorzyść rzutowania w porównaniu z zastosowaniem nierówności trójkąta przemawia zależność tej metody od liczącości dziedzin wymiarów.



Rys. 32. Porównanie wydajności algorytmu DBSCAN-PROJECTION i TI-DBSCAN zależności od wybranego wymiaru rzutowania i punktu referencyjnego przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania grupowań z parametrami MinPts=5 oraz Eps=0,05*przekątna danego zbioru

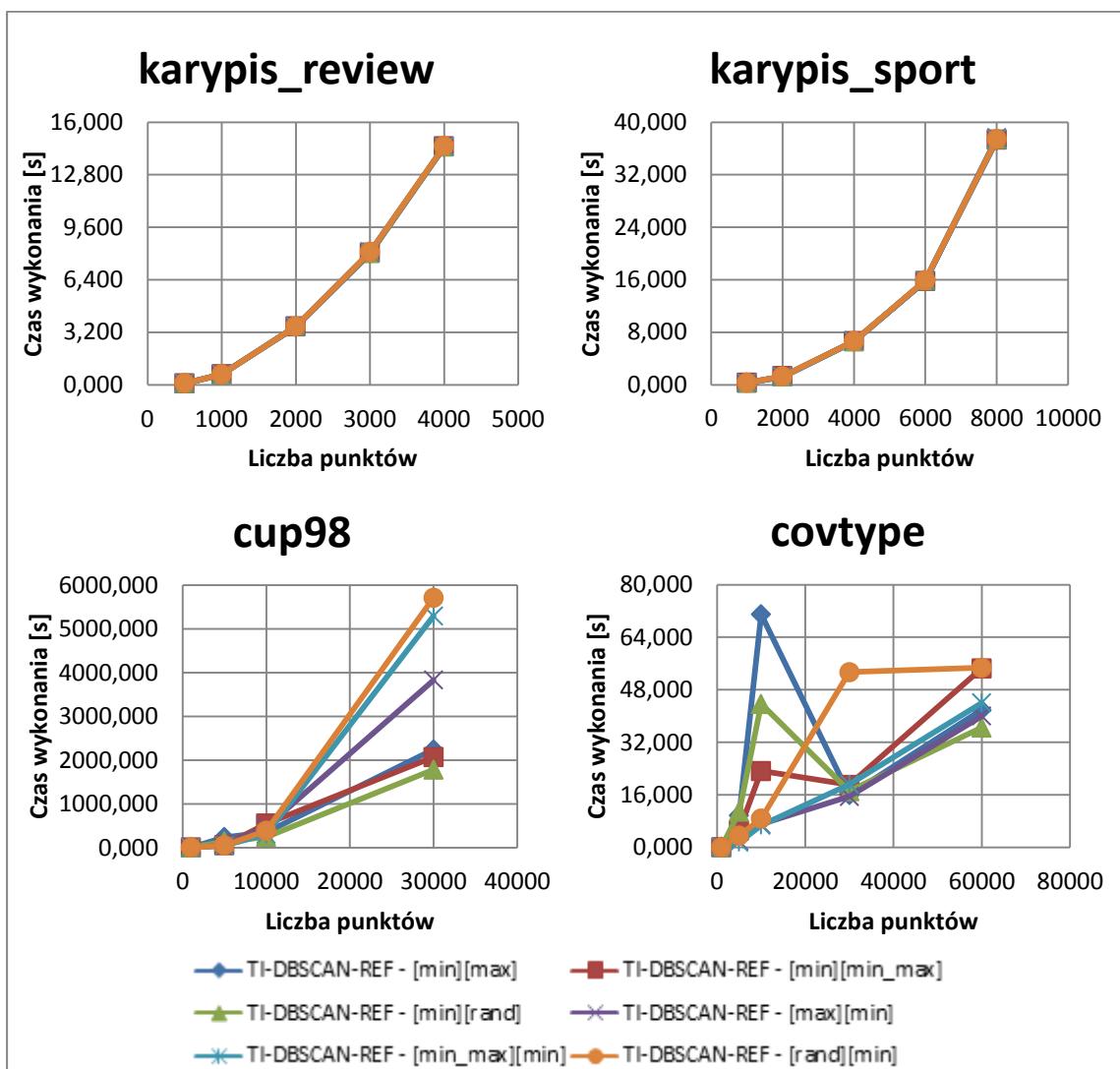
6.5.4. Badania algorytmu TI-DBSCAN-REF – wybór dwóch punktów referencyjnych

W celu wyznaczenia odpowiednich punktów referencyjnych dla algorytmu *TI-DBSCAN-REF* przeprowadziłem eksperymenty testując różne ich pary. W tabelach tab. 25 i tab. 26 zamieściłem czasy uruchomień algorytmu *TI-DBSCAN-REF* wraz z trwaniem składających się na niego kroków. Na rys. 33 znajdują się wykresy czasu wykonania algorytmu w funkcji liczby punktów.

Tab. 25. Porównanie wydajności algorytmu TI-DBSCAN-REF w zależności od wybranej pary punktów referencyjnych przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania grupowań z parametrami MinPts=5 oraz Eps= 0,05*przekątna danego zbioru

Tab. 26. Porównanie wydajności algorytmu TI-DBSCAN-REF w zależności od wybranej pary punktów referencyjnych przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania grupowań z parametrami MinPts=5 oraz Eps= 0,05*przekątna danego zbioru

| zbior | l. p. | TI-DBSCAN-REF [max][min] | | | | TI-DBSCAN-REF [min_max][min] | | | | TI-DBSCAN-REF [rand][min] | | | |
|----------------|-------|--------------------------|----------|-------|--------|------------------------------|----------|-------|-------|---------------------------|----------|-------|--------|
| | | wyk. | grup. | obl. | sort. | wyk. | grup. | obl. | sort. | wyk. | grup. | obl. | sort. |
| | | alg. | | odl. | | alg. | | odl. | | alg. | | odl. | |
| karypis_sport | 1000 | 0,302 | 0,276 | 0,000 | 0,026 | 0,301 | 0,281 | 0,000 | 0,021 | 0,302 | 0,281 | 0,000 | 0,021 |
| | 2000 | 1,238 | 1,155 | 0,015 | 0,068 | 1,238 | 1,170 | 0,015 | 0,052 | 1,253 | 1,186 | 0,011 | 0,057 |
| | 4000 | 6,666 | 6,422 | 0,031 | 0,213 | 6,656 | 6,422 | 0,037 | 0,197 | 6,635 | 6,438 | 0,021 | 0,176 |
| | 6000 | 15,787 | 15,402 | 0,052 | 0,332 | 15,834 | 15,387 | 0,073 | 0,374 | 15,923 | 15,429 | 0,115 | 0,379 |
| | 8000 | 37,580 | 36,977 | 0,031 | 0,572 | 37,471 | 36,889 | 0,031 | 0,552 | 37,404 | 36,899 | 0,031 | 0,473 |
| | 500 | 0,078 | 0,068 | 0,000 | 0,010 | 0,093 | 0,078 | 0,005 | 0,010 | 0,088 | 0,073 | 0,000 | 0,015 |
| karypis_review | 1000 | 0,629 | 0,593 | 0,005 | 0,031 | 0,629 | 0,598 | 0,000 | 0,031 | 0,629 | 0,598 | 0,005 | 0,026 |
| | 2000 | 3,567 | 3,443 | 0,015 | 0,109 | 3,552 | 3,432 | 0,005 | 0,115 | 3,552 | 3,442 | 0,010 | 0,099 |
| | 3000 | 8,060 | 7,841 | 0,016 | 0,203 | 8,097 | 7,847 | 0,016 | 0,234 | 8,091 | 7,842 | 0,015 | 0,234 |
| | 4000 | 14,519 | 14,243 | 0,016 | 0,260 | 14,513 | 14,227 | 0,021 | 0,265 | 14,535 | 14,233 | 0,016 | 0,286 |
| | 1000 | 0,037 | 0,016 | 0,005 | 0,016 | 0,130 | 0,021 | 0,000 | 0,109 | 0,042 | 0,021 | 0,000 | 0,021 |
| | 5000 | 1,950 | 0,733 | 0,041 | 1,175 | 1,519 | 0,785 | 0,037 | 0,696 | 3,577 | 2,226 | 0,031 | 1,320 |
| covtype | 10000 | 7,036 | 2,106 | 0,182 | 4,748 | 6,692 | 2,881 | 0,156 | 3,655 | 8,700 | 5,658 | 0,125 | 2,917 |
| | 30000 | 15,480 | 14,331 | 0,042 | 1,108 | 19,162 | 17,852 | 0,042 | 1,269 | 53,341 | 17,861 | 1,357 | 34,123 |
| | 60000 | 39,967 | 37,726 | 0,078 | 2,163 | 44,023 | 42,416 | 0,078 | 1,529 | 54,683 | 52,354 | 0,078 | 2,251 |
| | 1000 | 1,258 | 1,097 | 0,005 | 0,156 | 0,842 | 0,723 | 0,000 | 0,120 | 0,692 | 0,676 | 0,000 | 0,016 |
| | 5000 | 44,741 | 43,082 | 0,047 | 1,612 | 43,899 | 43,389 | 0,031 | 0,479 | 49,868 | 48,064 | 0,031 | 1,773 |
| | 10000 | 475,359 | 462,401 | 0,192 | 12,766 | 305,474 | 298,319 | 0,161 | 6,994 | 384,427 | 378,800 | 0,141 | 5,486 |
| cup98 | 30000 | 3827,157 | 3825,930 | 0,047 | 1,180 | 5291,611 | 5290,800 | 0,047 | 0,764 | 5706,921 | 5706,250 | 0,047 | 0,624 |
| | 60000 | - | - | - | - | - | - | - | - | - | - | - | - |



Rys. 33. Porównanie wydajności algorytmu TI-DBSCAN-REF w zależności od wybranej pary punktów referencyjnych przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania grupowań z parametrami MinPts=5 oraz Eps= 0,05*przekątna danego zbioru

Na podstawie rezultatów uzyskanych dla danych tekstowych niemożliwym jest wskazanie kombinacji punktów referencyjnych najbardziej przyspieszającej grupowanie. W przypadku zbioru covtype dla pierwszego punktu referencyjnego [min] występują anomalie analogiczne do zauważonych podczas badań doboru jednego punktu referencyjnego dla algorytmu TI-DBSCAN. Z tych samych powodów co opisane w rozdziale 6.5.1.4., czyli z uwagi na specyficzną charakterystykę zbioru covtype, w dalszych rozważaniach na temat wyboru punktu referencyjnego dla algorytmu TI-DBSCAN nie będę brał pod uwagę rezultatów uzyskanych dla tego zbioru.

Wynikami pozwalającymi na ocenę przyspieszenia grupowania przez pary punktów referencyjnych są rezultaty uzyskane dla zbioru cup98. Spośród nich wykonanie grupowania najbardziej przyspiesza te pary punktów referencyjnych, w których pierwszym punktem jest punkt minimalny co spójne jest z wnioskami uzyskanymi w rozdziale 6.5.1.4. Wśród trzech najszybszych par grupowanie najbardziej usprawniane jest przez parę [min][rand].

Z uwagi na uzyskane rezultaty w dalszej części pracy jako wyniki czasowe algorytmu *TI-DBSCAN-REF* będą prezentowane wyniki osiągnięte przy zastosowaniu pary punktów referencyjnych [min][rand].

6.5.5. Porównanie implementacji odmian algorytmu DBSCAN

W tab. 27 i tab. 28 zamieściłem rezultaty badań odmian algorytmu *DBSCAN*. Na rys. 34 zaprezentowałem wykresy czasów wykonania odmian algorytmu DBSCAN w funkcji czasu. Wyniki *TI-DBSCAN* zostały zebrane dla punktu referencyjnego [min], *DBSCAN-PROJECTION* dla [dmax], natomiast rezultaty *TI-DBSCAN-REF* dla punktów referencyjnych [max][rand]. Algorytm *DBSCAN-POINTS_ELIMINATION* stanowi modyfikację algorytmu *DBSCAN* stosującą eliminację sklasyfikowanych punktów. Podejście to zostało zaproponowane w artykule [9].

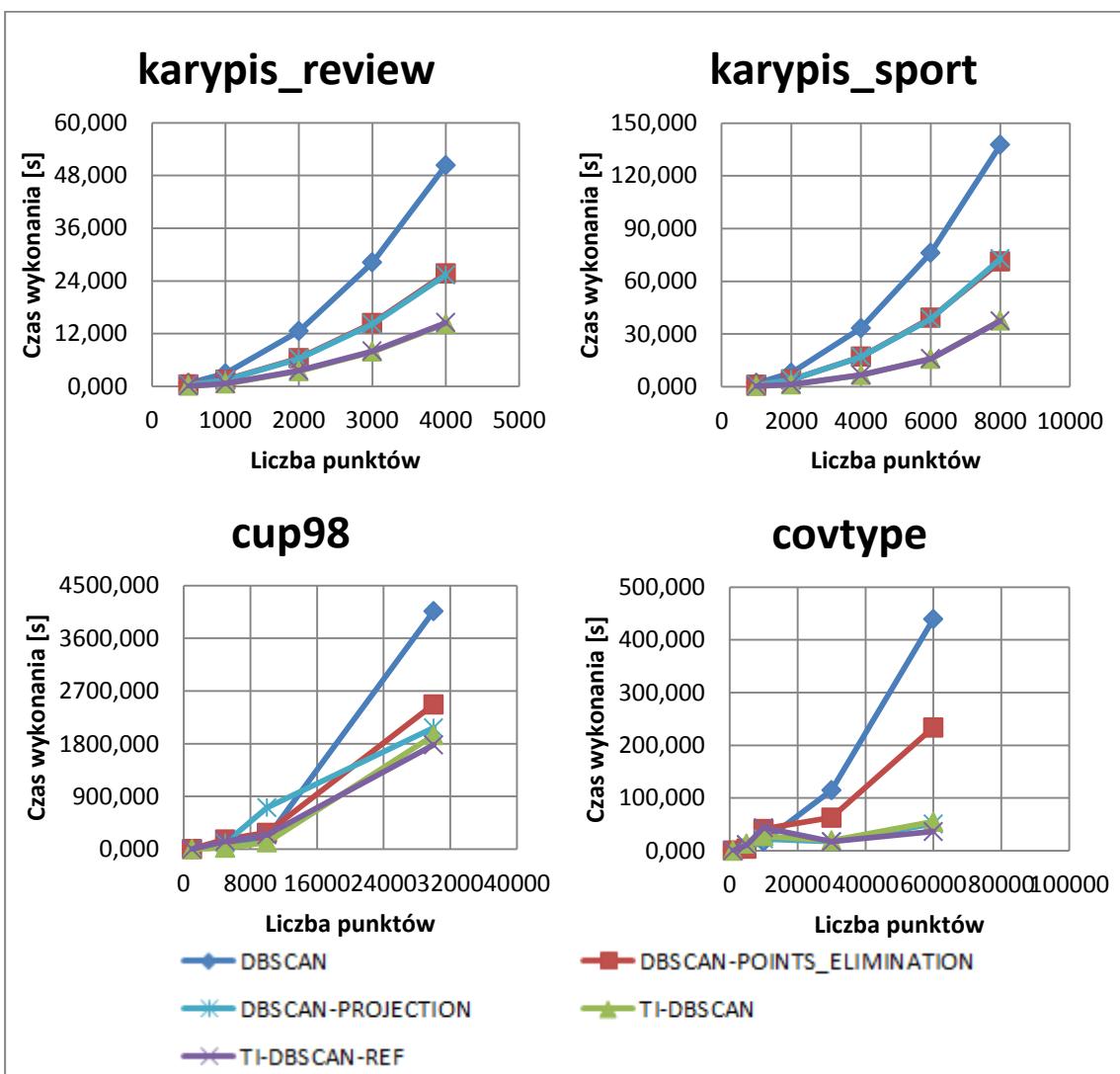
Rezultaty badań jednoznacznie potwierdzają wzrost wydajności *TI-DBSCAN*, *TI-DBSCAN-REF* oraz *DBSCAN-PROJECTION* w stosunku do algorytmów nie korzystających z nierówności trójkąta, tj. *DBSCAN* oraz *DBSCAN-POINTS_ELIMINATION*. Zastosowanie nierówności trójkąta pozwala uzyskiwać wyniki w o dwa rzędy krótszym czasie. Z uwagi na rzadki charakter, wzrost ten jest mniej widoczny dla danych tekstowych niż dla pozostałych zbiorów. Z otrzymanych rezultatów eksperymentów wynika, że zwiększenie liczby punktów referencyjnych z jednego do dwóch nie przynosi znaczącej poprawy sprawności grupowania. Biorąc pod uwagę tą obserwację wnioskuję, że dalsze zwiększenie liczby punktów referencyjnych może spowolnić wykonanie algorytmu, ponieważ koszt obsługi wielu punktów referencyjnych może przewyższyć zysk z ich zastosowania. Pozyskane wyniki wskazują, że zastosowanie rzutowania nie przyspiesza wykonania algorytmu bardziej niż wykorzystanie nierówności trójkąta.

Rezultaty badań wskazują również na krótszy czas grupowania zbioru algorytmem *DBSCAN-POINTS_ELIMINATION* niż *DBSCAN*. Użycie eliminacji punktów w algorytmie DBSCAN pozwala na co najmniej dwukrotnie szybsze uzyskanie rezultatów.

Tab. 27. Porównanie wydajności algorytmów TI-DBSCAN, TI-DBSCAN-REF i DBSCAN-PROJECTION przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania grupowań z parametrami MinPts=5 oraz Eps= 0,05*przekątna danego zbioru.

Tab. 28. Porównanie wydajności odmian algorytmów DBSCAN i DBSCAN-POINTS_ELIMINATION przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Tabela zawiera czasy wykonania grupowań z parametrami MinPts=5 oraz Eps= 0,05*przekątna danego zbioru

| zbior | l. p. | DBSCAN | | | | DBSCAN-POINTS_ELIMINATION | | | |
|----------------|-------|----------|----------|------|-------|---------------------------|----------|-------|--------|
| | | wyk. | grup. | obl. | sort. | wyk. | grup. | obl. | sort. |
| | | alg. | | odl. | | alg. | | odl. | |
| karypis_sport | 1000 | 2,059 | 2,059 | N/A | N/A | 1,045 | 1,030 | 0,000 | 0,015 |
| | 2000 | 7,899 | 7,899 | N/A | N/A | 3,973 | 3,931 | 0,000 | 0,042 |
| | 4000 | 33,207 | 33,207 | N/A | N/A | 16,952 | 16,817 | 0,000 | 0,136 |
| | 6000 | 76,035 | 76,035 | N/A | N/A | 39,057 | 38,719 | 0,016 | 0,322 |
| | 8000 | 137,519 | 137,519 | N/A | N/A | 71,131 | 70,481 | 0,016 | 0,634 |
| | 500 | 0,728 | 0,728 | N/A | N/A | 0,374 | 0,359 | 0,000 | 0,015 |
| karypis_review | 1000 | 2,917 | 2,917 | N/A | N/A | 1,498 | 1,456 | 0,000 | 0,042 |
| | 2000 | 12,548 | 12,548 | N/A | N/A | 6,375 | 6,297 | 0,000 | 0,078 |
| | 3000 | 28,163 | 28,163 | N/A | N/A | 14,362 | 14,186 | 0,010 | 0,166 |
| | 4000 | 50,294 | 50,294 | N/A | N/A | 25,683 | 25,439 | 0,005 | 0,239 |
| | 1000 | 0,140 | 0,140 | N/A | N/A | 0,229 | 0,083 | 0,005 | 0,140 |
| | 5000 | 3,906 | 3,906 | N/A | N/A | 3,614 | 2,382 | 0,104 | 1,128 |
| covtype | 10000 | 17,290 | 17,290 | N/A | N/A | 41,138 | 23,104 | 0,312 | 17,722 |
| | 30000 | 114,681 | 114,681 | N/A | N/A | 62,743 | 61,568 | 0,026 | 1,149 |
| | 60000 | 439,157 | 439,157 | N/A | N/A | 233,766 | 231,889 | 0,047 | 1,830 |
| | 1000 | 0,894 | 0,894 | N/A | N/A | 2,751 | 2,533 | 0,015 | 0,203 |
| | 5000 | 59,233 | 59,233 | N/A | N/A | 166,395 | 165,548 | 0,093 | 0,754 |
| | 10000 | 188,048 | 188,048 | N/A | N/A | 279,630 | 253,838 | 0,401 | 25,391 |
| cup98 | 30000 | 4064,900 | 4064,900 | N/A | N/A | 2471,351 | 2470,290 | 0,026 | 1,035 |
| | 60000 | - | - | N/A | N/A | - | - | - | - |



Rys. 34. Porównanie wydajności algorytmów DBSCAN, DBSCAN-POINTS_ELIMINATION, DBSCAN-PROJECTION, TI-DBSCAN, TI-DBSCAN-REF przy zastosowaniu odległości euklidesowej jako miary podobieństwa. Wykresy zawierają czasy wykonania grupowań z parametrami MinPts=5 oraz Eps= 0,05*przekątna danego zbioru

6.6. Badania algorytmu k-Neighborhood-Index – miara kosinusowa

W celu porównania wyszukiwania k sąsiedztwa w zależności od przyjętej miary podobieństwa, w niniejszym rozdziale powtórzyłem badania z rozdziału 6.2. dla miary kosinusowej jako miary podobieństwa.

6.6.1. Badania algorytmu TI- k-Neighborhood-Index

W kolejnych podrozdziałach wykonałem testy algorytmu *TI-k-Neighborhood-Index* korzystającego z miary kosinusowej jako miary podobieństwa. Podobnie jak w rozdziale 6.2.1. najpierw skupiłem się na badaniu czasu wykonania algorytmu w zależności od sposobu jego implementacji oraz w zależności od implementacji punktu. Następnie w oparciu o uzyskane rezultaty eksperymentów zbadałem wpływ wyboru punktu referencyjnego na wydajność *TI-k-Neighborhood-Index*.

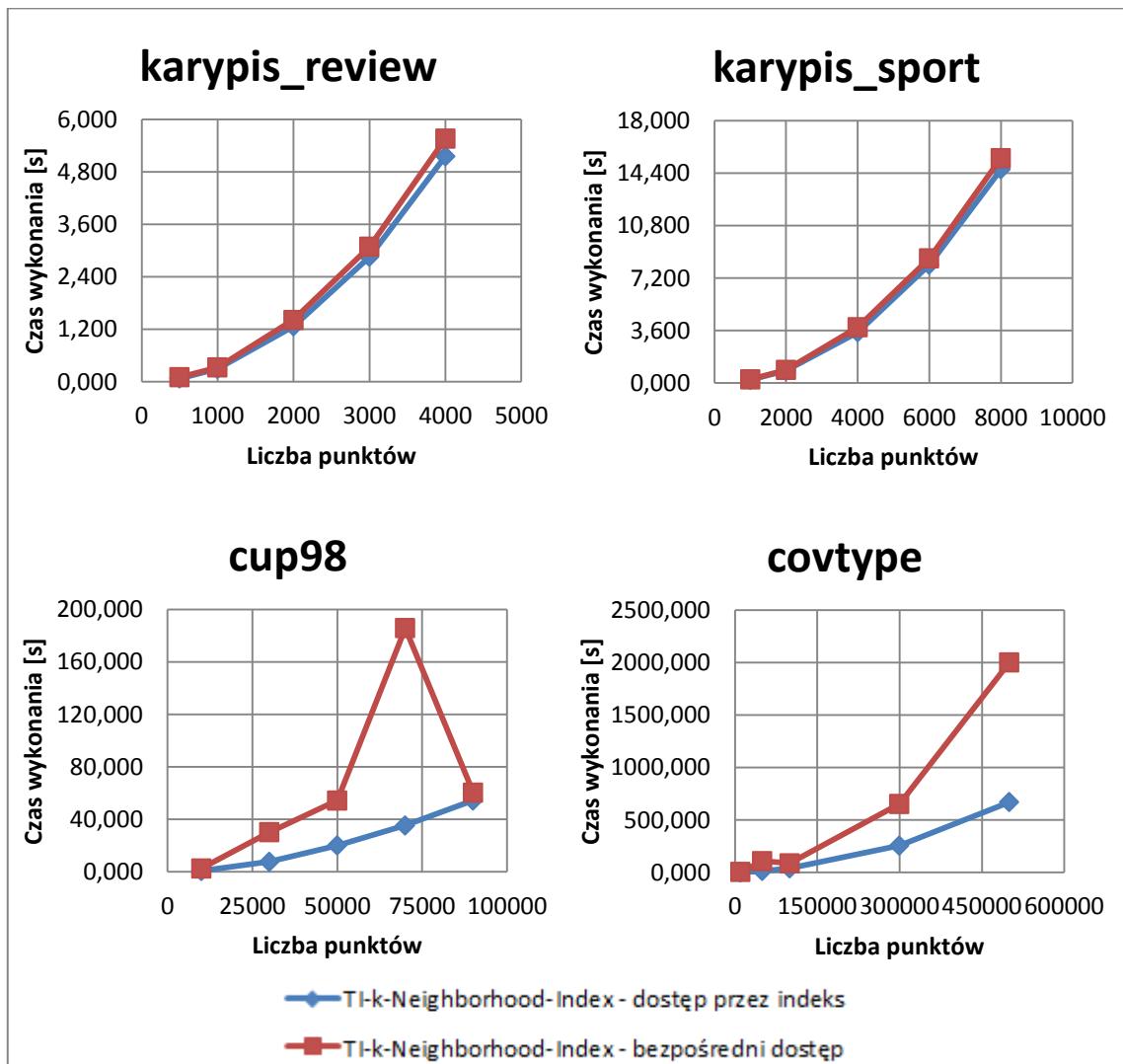
6.6.1.1. Implementacja algorytmu

Na przykładowych zbiorach danych przeprowadziłem badania dwóch implementacji algorytmu *TI-k-Neighborhood-Index* analogicznych do tych opisanych w rozdziale 6.2.1.1. Czasy wykonania eksperymentów wraz z trwaniem kroków, które się na nie składają zamieściłem w rab. 29. Na rys. 35 zaprezentowałem wykresy czasów trwania badanych implementacji algorytmu w funkcji liczby punktów.

Uzyskane wyniki , w swoim charakterze, podobne są do tych otrzymanych w rozdziale 6.2.1.1. Implementacja korzystająca z dostępu do danych przez indeks wykonuje się szybciej niż implementacja z bezpośrednim dostępem do danych ze względu na różnice w czasie sortowania wykorzystanej w danej implementacji struktury. Największa różnica w wydajności algorytmu wystąpiła przy 50000 punktów zbioru covtype. Dla tego przypadku implementacja z bezpośredniim dostępem do danych wykonuje się blisko 3 razy wolniej niż implementacja korzystająca z indeksu. Przyczyna tego zjawiska została przeze mnie opisana w rozdziale 6.2.1.1. Z uwagi na otrzymane rezultaty, w dalszych rozważaniach będę się odnosił do algorytmu *TI-k-Neighborhood-Index* wykorzystującego miarę kosinusową w charakterze miary podobieństwa jako do implementacji korzystającej z indeksu iteratorów.

Tab. 29. Porównanie wydajności TI-k-Neighborhood-Index w zależności od metody dostępu do danych przy zastosowaniu miary kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach dla 50% losowo wybranych punktów zbioru danych

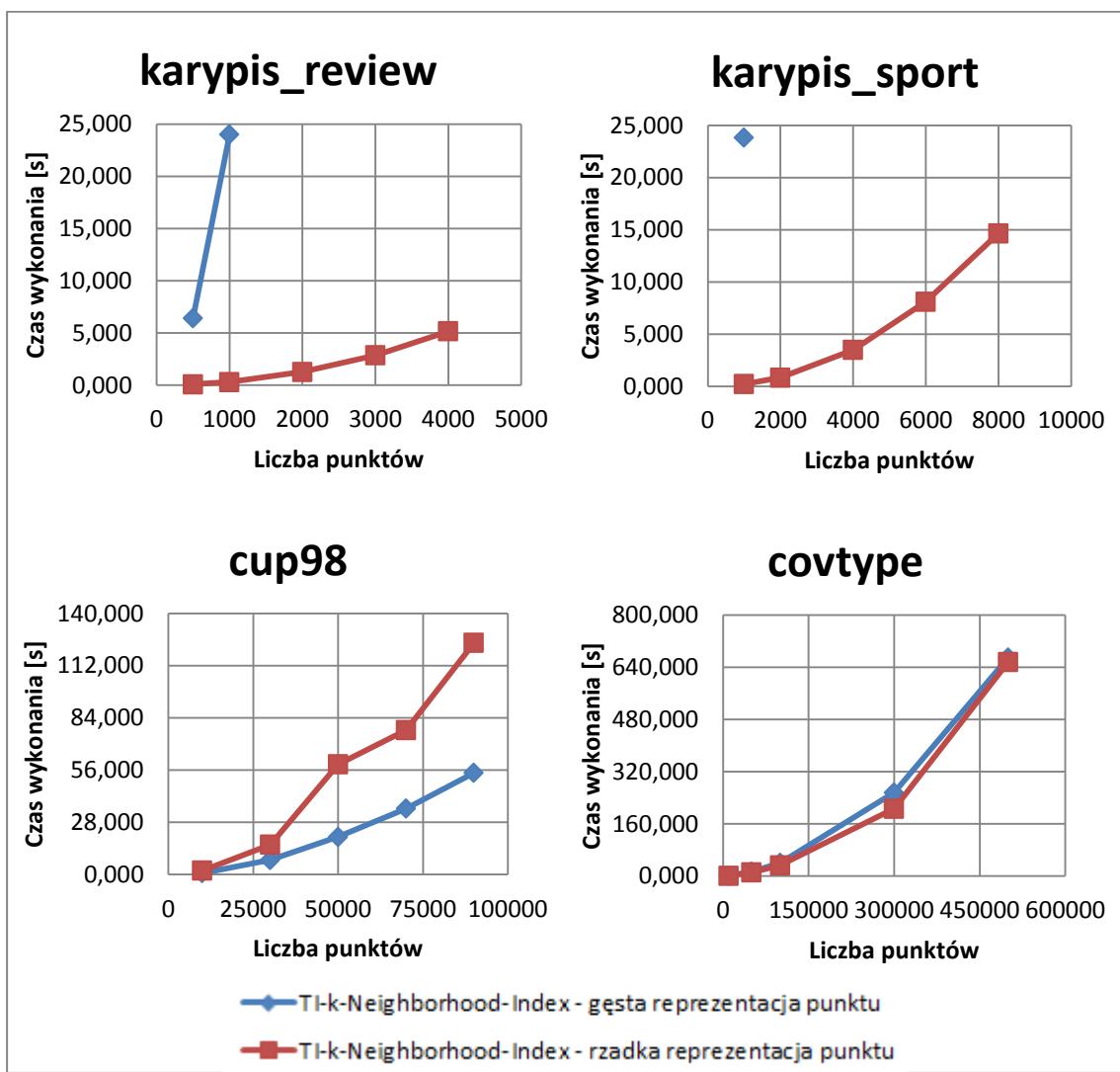
| zbior | l. p. | TI-k-Neighborhood-Index dostęp przez indeks | | | | | | TI-k-Neighborhood-Index bezpośredni dostęp | | | | | |
|----------------|--------|---|---------|-------|--------|-------|-------|--|---------|--------|----------|-------|--|
| | | wyk. | wysz. | bud. | obl. | sort. | norm. | wyk. | wysz. | bud. | obl. | sort. | |
| | | alg. | | ind. | odl. | | | alg. | | ind. | odl. | | |
| karypis_sport | 1000 | 0,218 | 0,213 | 0,000 | 0,005 | 0,000 | 0,000 | 0,249 | 0,218 | 0,000 | 0,031 | 0,000 | |
| | 2000 | 0,842 | 0,826 | 0,000 | 0,005 | 0,000 | 0,010 | 0,905 | 0,811 | 0,000 | 0,078 | 0,016 | |
| | 4000 | 3,484 | 3,453 | 0,000 | 0,016 | 0,000 | 0,015 | 3,827 | 3,443 | 0,015 | 0,354 | 0,015 | |
| | 6000 | 8,117 | 8,086 | 0,000 | 0,016 | 0,000 | 0,016 | 8,549 | 7,836 | 0,016 | 0,681 | 0,016 | |
| | 8000 | 14,654 | 14,617 | 0,000 | 0,021 | 0,000 | 0,016 | 15,428 | 14,134 | 0,026 | 1,253 | 0,015 | |
| | 500 | 0,078 | 0,078 | 0,000 | 0,000 | 0,000 | 0,000 | 0,104 | 0,078 | 0,000 | 0,016 | 0,010 | |
| karypis_review | 1000 | 0,296 | 0,291 | 0,000 | 0,000 | 0,000 | 0,005 | 0,328 | 0,291 | 0,000 | 0,031 | 0,005 | |
| | 2000 | 1,269 | 1,253 | 0,000 | 0,000 | 0,000 | 0,016 | 1,414 | 1,264 | 0,010 | 0,130 | 0,010 | |
| | 3000 | 2,855 | 2,829 | 0,000 | 0,010 | 0,000 | 0,016 | 3,088 | 2,823 | 0,010 | 0,239 | 0,016 | |
| | 4000 | 5,158 | 5,127 | 0,000 | 0,016 | 0,000 | 0,015 | 5,559 | 5,086 | 0,011 | 0,447 | 0,015 | |
| | 10000 | 0,764 | 0,489 | 0,000 | 0,270 | 0,000 | 0,005 | 4,607 | 0,421 | 0,078 | 4,092 | 0,015 | |
| | 50000 | 13,135 | 10,930 | 0,000 | 2,168 | 0,015 | 0,021 | 108,992 | 7,498 | 2,543 | 98,925 | 0,026 | |
| covtype | 100000 | 39,854 | 34,980 | 0,000 | 4,795 | 0,032 | 0,047 | 87,729 | 20,343 | 4,820 | 62,520 | 0,047 | |
| | 300000 | 255,592 | 239,279 | 0,000 | 16,021 | 0,141 | 0,151 | 653,350 | 110,719 | 15,397 | 527,078 | 0,156 | |
| | 500000 | 669,564 | 641,848 | 0,005 | 27,196 | 0,265 | 0,250 | 2000,451 | 271,123 | 25,058 | 1704,020 | 0,250 | |
| | 10000 | 0,733 | 0,562 | 0,000 | 0,171 | 0,000 | 0,000 | 2,683 | 0,395 | 0,109 | 2,179 | 0,000 | |
| | 30000 | 7,738 | 6,682 | 0,000 | 1,029 | 0,011 | 0,016 | 30,279 | 3,177 | 0,712 | 26,374 | 0,015 | |
| | 50000 | 20,161 | 17,706 | 0,000 | 2,418 | 0,016 | 0,021 | 54,382 | 8,388 | 1,680 | 44,288 | 0,026 | |
| cup98 | 70000 | 35,350 | 32,464 | 0,000 | 2,824 | 0,031 | 0,031 | 185,776 | 14,680 | 3,084 | 167,981 | 0,031 | |
| | 90000 | 54,470 | 50,144 | 0,000 | 4,238 | 0,042 | 0,046 | 60,414 | 22,651 | 4,504 | 33,212 | 0,047 | |



Rys. 35. Porównanie wydajności implementacji algorytmu TI-k-Neighborhood-Index w zależności od zastosowanej metody dostępu do danych przy zastosowaniu miary kosinusowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań $k=5$ sąsiedztwa w przykładowych zbiorach danych dla 50% losowo wybranych punktów zbioru danych

6.6.1.2. Implementacja struktury punktów

Dla miary kosinusowej jako miary podobieństwa powtórzyłem również badania z rozdziału 6.2.1.2. Uzyskane wyniki czasowe uruchomienia algorytmów wraz ze składającymi się na nie krokami zamieściłem w tab. 30 Na rys. 36 zaprezentowałem wykresy czasu wykonania algorytmów w funkcji liczby punktów.



Rys. 36. Porównanie wydajności algorytmu TI-kNeighborhood-Index w zależności od implementacji punktu przy zastosowaniu miary kosinusowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań k=5 sąsiadztwa w przykładowych zbiorach danych dla 50% losowo wybranych punktów zbioru danych

Otrzymane rezultaty są analogiczne do tych uzyskanych w rozdziale 6.2.1.2. Stąd, w dalszej części pracy badania algorytmów *TI-k-Neighborhood-Index* działających na danych tekstowych i korzystających z miary kosinusowej w charakterze miary podobieństwa będą wykonywane z użyciem implementacji punktu rzadkiego.

Tab. 30. Porównanie wydajności TI-kNeighborhood-Index w zależności od implementacji punktu przy zastosowaniu miary kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 50% losowo wybranych punktów zbioru

| zbior | l. p. | TI-k-Neighborhood-Index gęsta reprezentacja punktu | | | | | | TI-k-Neighborhood-Index rzadka reprezentacja punktu | | | | | |
|----------------|--------|--|---------|-------|--------|-------|-------|---|---------|-------|--------|-------|-------|
| | | wyk. | | wysz. | bud. | obl. | sort. | norm. | wyk. | | wysz. | bud. | obl. |
| | | alg. | ind. | odl. | odl. | | | | alg. | ind. | odl. | odl. | |
| karypis_sport | 1000 | 23,832 | 22,449 | 0,000 | 0,255 | 0,000 | 1,128 | 0,218 | 0,213 | 0,000 | 0,005 | 0,000 | 0,000 |
| | 2000 | - | - | - | - | - | - | 0,842 | 0,826 | 0,000 | 0,005 | 0,000 | 0,010 |
| | 4000 | - | - | - | - | - | - | 3,484 | 3,453 | 0,000 | 0,016 | 0,000 | 0,015 |
| | 6000 | - | - | - | - | - | - | 8,117 | 8,086 | 0,000 | 0,016 | 0,000 | 0,016 |
| | 8000 | - | - | - | - | - | - | 14,654 | 14,617 | 0,000 | 0,021 | 0,000 | 0,016 |
| karypis_review | 500 | 6,393 | 5,693 | 0,000 | 0,129 | 0,000 | 0,570 | 0,078 | 0,078 | 0,000 | 0,000 | 0,000 | 0,000 |
| | 1000 | 24,015 | 22,623 | 0,000 | 0,253 | 0,000 | 1,138 | 0,296 | 0,291 | 0,000 | 0,000 | 0,000 | 0,005 |
| | 2000 | - | - | - | - | - | - | 1,269 | 1,253 | 0,000 | 0,000 | 0,000 | 0,016 |
| | 3000 | - | - | - | - | - | - | 2,855 | 2,829 | 0,000 | 0,010 | 0,000 | 0,016 |
| | 4000 | - | - | - | - | - | - | 5,158 | 5,127 | 0,000 | 0,016 | 0,000 | 0,015 |
| covtype | 10000 | 0,764 | 0,489 | 0,000 | 0,270 | 0,000 | 0,005 | 0,613 | 0,535 | 0,000 | 0,073 | 0,003 | 0,002 |
| | 50000 | 13,135 | 10,930 | 0,000 | 2,168 | 0,015 | 0,021 | 11,080 | 9,742 | 0,001 | 1,309 | 0,016 | 0,012 |
| | 100000 | 39,854 | 34,980 | 0,000 | 4,795 | 0,032 | 0,047 | 32,876 | 29,815 | 0,001 | 3,000 | 0,037 | 0,023 |
| | 300000 | 255,592 | 239,279 | 0,000 | 16,021 | 0,141 | 0,151 | 206,123 | 195,487 | 0,003 | 10,417 | 0,146 | 0,070 |
| | 500000 | 669,564 | 641,848 | 0,005 | 27,196 | 0,265 | 0,250 | 655,649 | 638,020 | 0,005 | 17,229 | 0,278 | 0,117 |
| cup98 | 10000 | 0,733 | 0,562 | 0,000 | 0,171 | 0,000 | 0,000 | 2,097 | 1,984 | 0,000 | 0,101 | 0,003 | 0,009 |
| | 30000 | 7,738 | 6,682 | 0,000 | 1,029 | 0,011 | 0,016 | 16,001 | 15,314 | 0,000 | 0,649 | 0,011 | 0,026 |
| | 50000 | 20,161 | 17,706 | 0,000 | 2,418 | 0,016 | 0,021 | 59,108 | 57,789 | 0,001 | 1,253 | 0,022 | 0,043 |
| | 70000 | 35,350 | 32,464 | 0,000 | 2,824 | 0,031 | 0,031 | 77,367 | 75,760 | 0,001 | 1,512 | 0,033 | 0,061 |
| | 90000 | 54,470 | 50,144 | 0,000 | 4,238 | 0,042 | 0,046 | 124,205 | 121,071 | 0,001 | 3,009 | 0,046 | 0,078 |

6.6.1.3. Wybór punktu referencyjnego

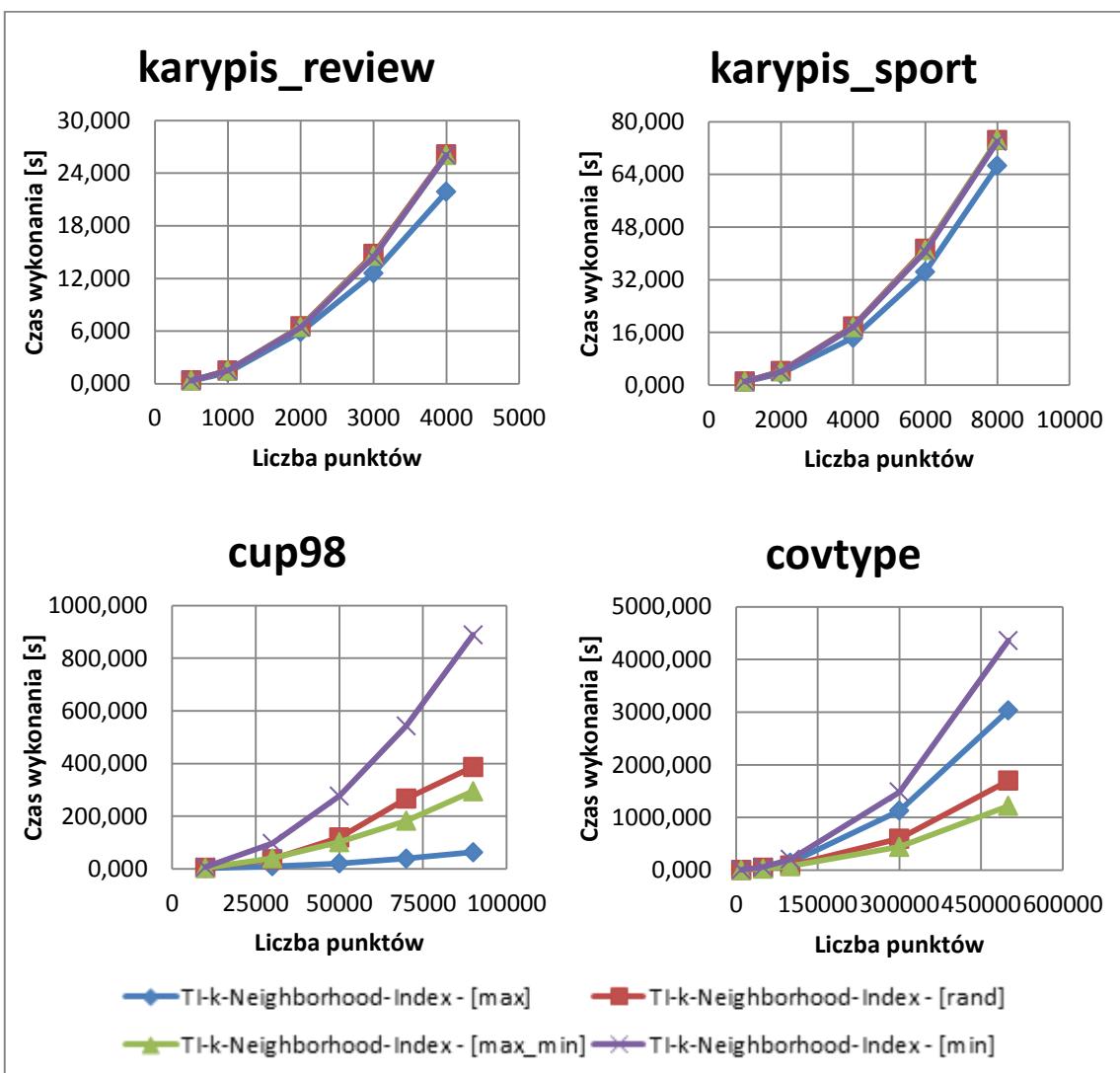
W poszukiwaniu właściwego punktu referencyjnego dla algorytmu *TI-k-Neighborhood-Index* przy użyciu miary kosinusowej powtórzyłem badania z rozdziału 6.2.1.3. Podobnie jak poprzednio eksperymenty przeprowadziłem z wykorzystaniem punktów [max], [min], [rand] oraz [max_min]. Czasy uruchomień algorytmów wraz z trwaniem składających się nań krokami zamieściłem w tab. 31 i tab. 32. Na rys. 37 zaprezentowałem wykresy czasów wykonania algorytmów w funkcji liczby punktów.

Uzyskane rezultaty mają odmienny charakter od tych otrzymanych w rozdziale 6.2.1.3., w przypadku których badałem wpływ doboru punktu referencyjnego na algorytm *TI-k-Neighborhood-Index* korzystający z odległości euklidesowej jako miary podobieństwa. Zasadniczą i cenną różnicą jest możliwość jednoznacznego wskazania najskuteczniejszego punktu referencyjnego zarówno dla danych tekstowych jak i pozostałych zbiorów.

Analiza otrzymanych wyników pozwala wskazać [max] jako punkt najbardziej przyspieszający wyszukanie k sąsiadów dla większości zbiorów danych. W przypadku cup98 dla punktu [max] rezultaty osiągane są o rząd wielkości szybciej niż dla punktu [min]. Wyjątek stanowi zbiór covtype, dla którego [max] okazał się jednym z najsłabiej przyspieszających wyszukanie k sąsiadów punktem referencyjnym. Podobnie jak w poprzednich rozdziałach, ze względu na niereprezentatywny charakter zbioru covtype rezultaty uzyskane dla tego zbioru zostaną pominięte w dalszych rozważaniach.

Wyniki otrzymane dla wszystkich przykładowych zbiorów danych jednoznacznie potwierdzają, że punktem referencyjnym najgorzej wpływającym na wydajność algorytmu jest punkt [min]. Obserwacja ta wynika stąd, że punkt ten leży blisko środka hipersfery, na powierzchni której leżą punkty zbioru po normalizacji, dlatego [min] w bardzo małym stopniu redukuje liczbę potencjalnych sąsiadów danego zapytania.

W porównaniu do wyników otrzymanych w rozdziale 6.2.1.3. punkt referencyjny [max] sprawdza się najlepiej na tle innych badanych punktów gdy miara kosinusowa stosowana jest jako miara podobieństwa. W dalszej części pracy jako wyniki czasowe algorytmu *TI-k-Neighborhood-Index* z zastosowaniem miary kosinusowej jako miary podobieństwa będą prezentowane rezultaty osiągnięte przy zastosowaniu punktu maksymalnego jako punktu referencyjnego.



Rys. 37. Porównanie wydajności algorytmu TI-k-Neighborhood-Index w zależności od wybranego punktu referencyjnego przy zastosowaniu miary kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań $k=5$ sąsiedztwa w przykładowych zbiorach danych dla 50% losowo wybranych punktów zbioru danych

Tab. 31. Porównanie wydajności TI-k-Neighborhood-Index w zależności od punktu referencyjnego przy zastosowaniu miary kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach dla 50% losowo wybranych punktów zbioru danych

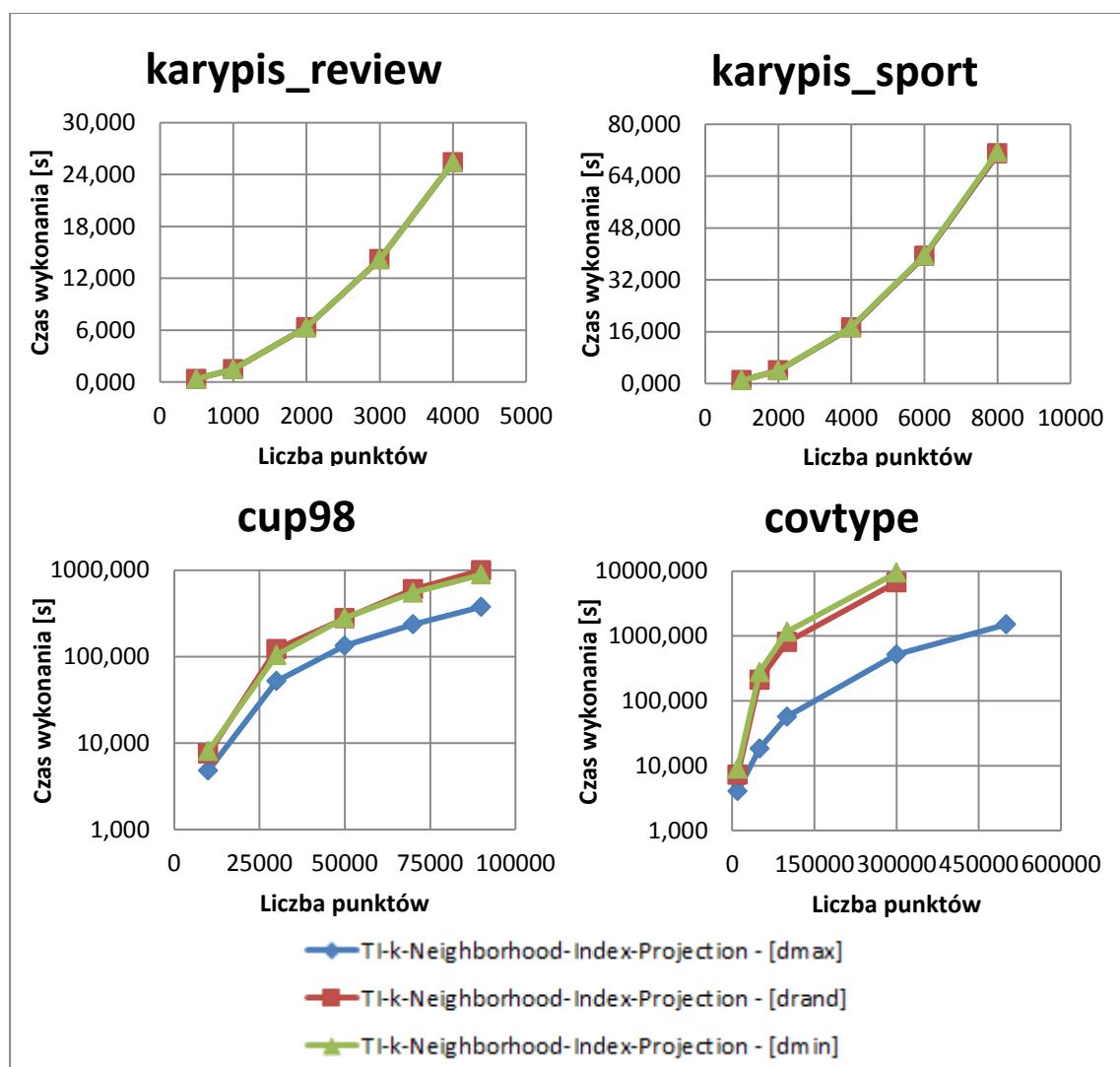
| zbior | l.p. | TI-k-Neighborhood-Index [max] | | | | | | TI-k-Neighborhood-Index [rand] | | | | | |
|----------------|--------|-------------------------------|----------|-------|--------|-------|-------|--------------------------------|----------|-------|--------|-------|-------|
| | | wyk. | wysz. | bud. | obl. | sort. | norm. | wyk. | wysz. | bud. | obl. | sort. | norm. |
| | | alg. | | ind. | odl. | | | alg. | | ind. | odl. | | |
| karypis_sport | 1000 | 1,034 | 1,027 | 0,000 | 0,005 | 0,000 | 0,002 | 1,077 | 1,070 | 0,000 | 0,005 | 0,000 | 0,002 |
| | 2000 | 3,519 | 3,510 | 0,000 | 0,005 | 0,001 | 0,004 | 4,247 | 4,237 | 0,000 | 0,005 | 0,001 | 0,004 |
| | 4000 | 14,359 | 14,337 | 0,000 | 0,012 | 0,001 | 0,009 | 17,819 | 17,797 | 0,000 | 0,012 | 0,001 | 0,009 |
| | 6000 | 34,451 | 34,418 | 0,000 | 0,018 | 0,001 | 0,013 | 41,372 | 41,339 | 0,000 | 0,018 | 0,002 | 0,013 |
| | 8000 | 66,656 | 66,607 | 0,000 | 0,029 | 0,003 | 0,017 | 74,430 | 74,381 | 0,000 | 0,029 | 0,003 | 0,017 |
| | 500 | 0,337 | 0,333 | 0,000 | 0,003 | 0,000 | 0,002 | 0,374 | 0,370 | 0,000 | 0,002 | 0,000 | 0,002 |
| karypis_review | 1000 | 1,293 | 1,286 | 0,000 | 0,004 | 0,000 | 0,003 | 1,493 | 1,486 | 0,000 | 0,004 | 0,000 | 0,003 |
| | 2000 | 5,894 | 5,878 | 0,000 | 0,008 | 0,001 | 0,007 | 6,513 | 6,498 | 0,000 | 0,008 | 0,000 | 0,007 |
| | 3000 | 12,591 | 12,569 | 0,000 | 0,011 | 0,001 | 0,010 | 14,758 | 14,736 | 0,000 | 0,011 | 0,001 | 0,010 |
| | 4000 | 21,877 | 21,847 | 0,000 | 0,016 | 0,001 | 0,013 | 26,117 | 26,088 | 0,000 | 0,015 | 0,001 | 0,013 |
| | 10000 | 2,913 | 2,553 | 0,000 | 0,322 | 0,003 | 0,005 | 2,752 | 2,539 | 0,000 | 0,205 | 0,003 | 0,005 |
| | 50000 | 37,549 | 34,851 | 0,001 | 2,934 | 0,016 | 0,025 | 44,024 | 41,471 | 0,001 | 2,511 | 0,016 | 0,025 |
| covtype | 100000 | 141,164 | 134,889 | 0,001 | 6,105 | 0,038 | 0,050 | 88,524 | 82,892 | 0,001 | 5,545 | 0,036 | 0,050 |
| | 300000 | 1128,400 | 1106,590 | 0,003 | 21,494 | 0,147 | 0,150 | 599,154 | 578,544 | 0,003 | 20,319 | 0,138 | 0,150 |
| | 500000 | 3034,090 | 2997,940 | 0,003 | 35,633 | 0,275 | 0,250 | 1700,227 | 1664,180 | 0,005 | 35,520 | 0,271 | 0,251 |
| | 10000 | 2,868 | 2,414 | 0,000 | 0,447 | 0,003 | 0,005 | 3,723 | 3,496 | 0,000 | 0,220 | 0,003 | 0,005 |
| | 30000 | 9,568 | 8,799 | 0,000 | 0,744 | 0,010 | 0,015 | 36,900 | 35,574 | 0,001 | 1,299 | 0,011 | 0,015 |
| | 50000 | 20,456 | 18,402 | 0,001 | 2,009 | 0,019 | 0,025 | 119,318 | 117,493 | 0,001 | 1,780 | 0,019 | 0,025 |
| cup98 | 70000 | 39,183 | 35,322 | 0,001 | 3,796 | 0,029 | 0,035 | 266,641 | 264,152 | 0,001 | 2,425 | 0,029 | 0,035 |
| | 90000 | 63,047 | 58,520 | 0,001 | 4,440 | 0,039 | 0,046 | 386,949 | 382,602 | 0,001 | 4,260 | 0,040 | 0,046 |

Tab. 32. Porównanie wydajności TI-k-Neighborhood-Index w zależności od punktu referencyjnego przy zastosowaniu miary kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach dla 50% losowo wybranych punktów zbioru danych

| zbior | l.p. | TI-k-Neighborhood-Index [max_min] | | | | | | TI-k-Neighborhood-Index [min] | | | | | |
|----------------|--------|-----------------------------------|----------|-------|--------|-------|-------|-------------------------------|----------|-------|--------|-------|-------|
| | | wyk. | wysz. | bud. | obl. | sort. | norm. | wyk. | wysz. | bud. | obl. | sort. | norm. |
| | | alg. | | ind. | odl. | | | alg. | | ind. | odl. | | |
| karypis_sport | 1000 | 1,065 | 1,059 | 0,000 | 0,003 | 0,000 | 0,002 | 1,053 | 1,050 | 0,000 | 0,003 | 0,000 | 0,002 |
| | 2000 | 4,164 | 4,155 | 0,000 | 0,005 | 0,000 | 0,004 | 4,071 | 4,067 | 0,000 | 0,005 | 0,000 | 0,004 |
| | 4000 | 17,606 | 17,586 | 0,000 | 0,011 | 0,001 | 0,008 | 17,608 | 17,598 | 0,000 | 0,011 | 0,000 | 0,008 |
| | 6000 | 41,030 | 41,000 | 0,000 | 0,017 | 0,001 | 0,012 | 40,575 | 40,558 | 0,000 | 0,017 | 0,001 | 0,013 |
| | 8000 | 74,523 | 74,479 | 0,000 | 0,024 | 0,003 | 0,018 | 74,051 | 74,029 | 0,000 | 0,020 | 0,001 | 0,018 |
| | 500 | 0,372 | 0,369 | 0,000 | 0,002 | 0,000 | 0,001 | 0,372 | 0,370 | 0,000 | 0,002 | 0,000 | 0,002 |
| karypis_review | 1000 | 1,493 | 1,486 | 0,000 | 0,004 | 0,000 | 0,003 | 1,494 | 1,491 | 0,000 | 0,003 | 0,000 | 0,003 |
| | 2000 | 6,469 | 6,455 | 0,000 | 0,007 | 0,000 | 0,007 | 6,422 | 6,415 | 0,000 | 0,007 | 0,000 | 0,006 |
| | 3000 | 14,541 | 14,520 | 0,000 | 0,010 | 0,001 | 0,010 | 14,428 | 14,418 | 0,000 | 0,010 | 0,000 | 0,010 |
| | 4000 | 26,044 | 26,016 | 0,000 | 0,014 | 0,001 | 0,013 | 26,047 | 26,034 | 0,000 | 0,013 | 0,000 | 0,013 |
| | 10000 | 3,770 | 3,400 | 0,000 | 0,362 | 0,003 | 0,005 | 6,024 | 5,695 | 0,000 | 0,326 | 0,003 | 0,005 |
| | 50000 | 26,748 | 23,722 | 0,001 | 2,985 | 0,015 | 0,025 | 59,520 | 56,663 | 0,001 | 2,832 | 0,015 | 0,025 |
| covtype | 100000 | 75,949 | 70,091 | 0,001 | 5,772 | 0,035 | 0,050 | 206,582 | 200,670 | 0,001 | 6,186 | 0,036 | 0,050 |
| | 300000 | 441,928 | 421,168 | 0,002 | 20,476 | 0,131 | 0,151 | 1484,120 | 1462,950 | 0,002 | 21,028 | 0,134 | 0,149 |
| | 500000 | 1219,184 | 1182,240 | 0,005 | 36,434 | 0,254 | 0,250 | 4360,080 | 4323,260 | 0,005 | 36,495 | 0,251 | 0,250 |
| | 10000 | 3,569 | 3,401 | 0,000 | 0,160 | 0,003 | 0,005 | 6,776 | 6,534 | 0,000 | 0,268 | 0,001 | 0,005 |
| | 30000 | 40,471 | 39,387 | 0,001 | 1,059 | 0,010 | 0,015 | 96,608 | 94,900 | 0,001 | 1,766 | 0,001 | 0,015 |
| | 50000 | 101,757 | 98,766 | 0,001 | 2,946 | 0,019 | 0,025 | 276,935 | 273,325 | 0,001 | 3,477 | 0,002 | 0,025 |
| cup98 | 70000 | 182,582 | 180,763 | 0,001 | 1,754 | 0,028 | 0,035 | 542,703 | 538,977 | 0,001 | 3,751 | 0,004 | 0,035 |
| | 90000 | 294,857 | 291,390 | 0,001 | 3,380 | 0,040 | 0,046 | 888,676 | 883,159 | 0,001 | 5,560 | 0,004 | 0,046 |

6.6.2. Badania algorytmu k-Neighborhood-Index-Projection – miara kosinusowa

W badaniach algorytmu *k-Neighborhood-Index-Projection*, tak jak w rozdziale 6.2.2, skupiłem się na trzech rodzajach rzutowania [dmax] [dmin] oraz [drand]. W tab. 33 i tab. 34 zamieściłem czasy uruchomień algorytmu *k-Neighborhood-Index-Projection* wraz z trwaniem składających się na niego kroków. Na rys. 38 znajdują się wykresy czasu wykonania algorytmu w funkcji liczby punktów.



Rys. 38. Porównanie wydajności algorytmu TI-k-Neighborhood-Index-Projection w zależności od wybranej metody rzutowania przy zastosowaniu miara kosinusowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań $k=5$ sąsiadztwa w przykładowych zbiorach danych dla 50% losowo wybranych punktów zbioru danych

Tab. 33. Porównanie wydajności TI-k-Neighborhood-Index-Projection w zależności od metody rzutowania przy zastosowaniu miary kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach dla 50% losowo wybranych punktów zbioru danych

| zbior | l.p. | TI-k-Neighborhood-Index-Projection [dmax] | | | | | | TI-k-Neighborhood-Index-Projection [drand] | | | | | |
|----------------|--------|---|----------|-------|--------|-------|-------|--|----------|-------|--------|-------|-------|
| | | wyk. | wysz. | bud. | obl. | sort. | norm. | wyk. | wysz. | bud. | obl. | sort. | norm. |
| | | alg. | | ind. | odl. | | | alg. | | ind. | odl. | | |
| karypis_sport | 1000 | 1,047 | 1,044 | 0,000 | 0,001 | 0,000 | 0,002 | 1,051 | 1,048 | 0,000 | 0,001 | 0,000 | 0,002 |
| | 2000 | 4,038 | 4,032 | 0,000 | 0,002 | 0,000 | 0,004 | 4,029 | 4,023 | 0,000 | 0,002 | 0,000 | 0,004 |
| | 4000 | 17,224 | 17,211 | 0,000 | 0,005 | 0,000 | 0,008 | 17,277 | 17,265 | 0,000 | 0,004 | 0,000 | 0,008 |
| | 6000 | 39,275 | 39,254 | 0,000 | 0,008 | 0,000 | 0,013 | 39,356 | 39,336 | 0,000 | 0,007 | 0,000 | 0,013 |
| | 8000 | 70,763 | 70,736 | 0,001 | 0,008 | 0,000 | 0,017 | 70,928 | 70,903 | 0,000 | 0,008 | 0,000 | 0,017 |
| | 500 | 0,371 | 0,369 | 0,000 | 0,001 | 0,000 | 0,001 | 0,372 | 0,370 | 0,000 | 0,001 | 0,000 | 0,001 |
| karypis_review | 1000 | 1,482 | 1,478 | 0,000 | 0,001 | 0,000 | 0,003 | 1,484 | 1,480 | 0,000 | 0,001 | 0,000 | 0,003 |
| | 2000 | 6,357 | 6,349 | 0,000 | 0,002 | 0,000 | 0,006 | 6,334 | 6,325 | 0,000 | 0,002 | 0,000 | 0,006 |
| | 3000 | 14,186 | 14,173 | 0,000 | 0,003 | 0,000 | 0,010 | 14,201 | 14,189 | 0,000 | 0,003 | 0,000 | 0,009 |
| | 4000 | 25,388 | 25,371 | 0,000 | 0,003 | 0,000 | 0,013 | 25,424 | 25,407 | 0,000 | 0,004 | 0,000 | 0,013 |
| | 10000 | 4,032 | 3,698 | 0,000 | 0,326 | 0,003 | 0,005 | 7,255 | 7,088 | 0,000 | 0,162 | 0,000 | 0,005 |
| | 50000 | 18,396 | 15,797 | 0,001 | 2,558 | 0,015 | 0,025 | 210,542 | 207,572 | 0,001 | 2,938 | 0,006 | 0,025 |
| covtype | 100000 | 57,044 | 50,757 | 0,002 | 6,201 | 0,034 | 0,050 | 802,338 | 796,325 | 0,001 | 5,959 | 0,003 | 0,050 |
| | 300000 | 517,584 | 496,146 | 0,004 | 21,144 | 0,141 | 0,150 | 6717,360 | 6696,000 | 0,002 | 21,203 | 0,009 | 0,146 |
| | 500000 | 1506,937 | 1470,180 | 0,004 | 36,233 | 0,270 | 0,250 | | | | | | |
| | 10000 | 4,819 | 4,571 | 0,000 | 0,239 | 0,003 | 0,005 | 7,545 | 7,196 | 0,000 | 0,342 | 0,003 | 0,005 |
| | 30000 | 52,023 | 50,588 | 0,001 | 1,409 | 0,010 | 0,015 | 121,720 | 120,225 | 0,001 | 1,470 | 0,010 | 0,015 |
| | 50000 | 133,558 | 130,449 | 0,001 | 3,064 | 0,019 | 0,025 | 276,007 | 273,229 | 0,001 | 2,732 | 0,019 | 0,026 |
| cup98 | 70000 | 235,906 | 232,374 | 0,001 | 3,468 | 0,029 | 0,035 | 598,666 | 594,648 | 0,001 | 3,954 | 0,029 | 0,035 |
| | 90000 | 375,325 | 370,919 | 0,001 | 4,320 | 0,040 | 0,045 | 993,199 | 988,330 | 0,001 | 4,783 | 0,040 | 0,046 |

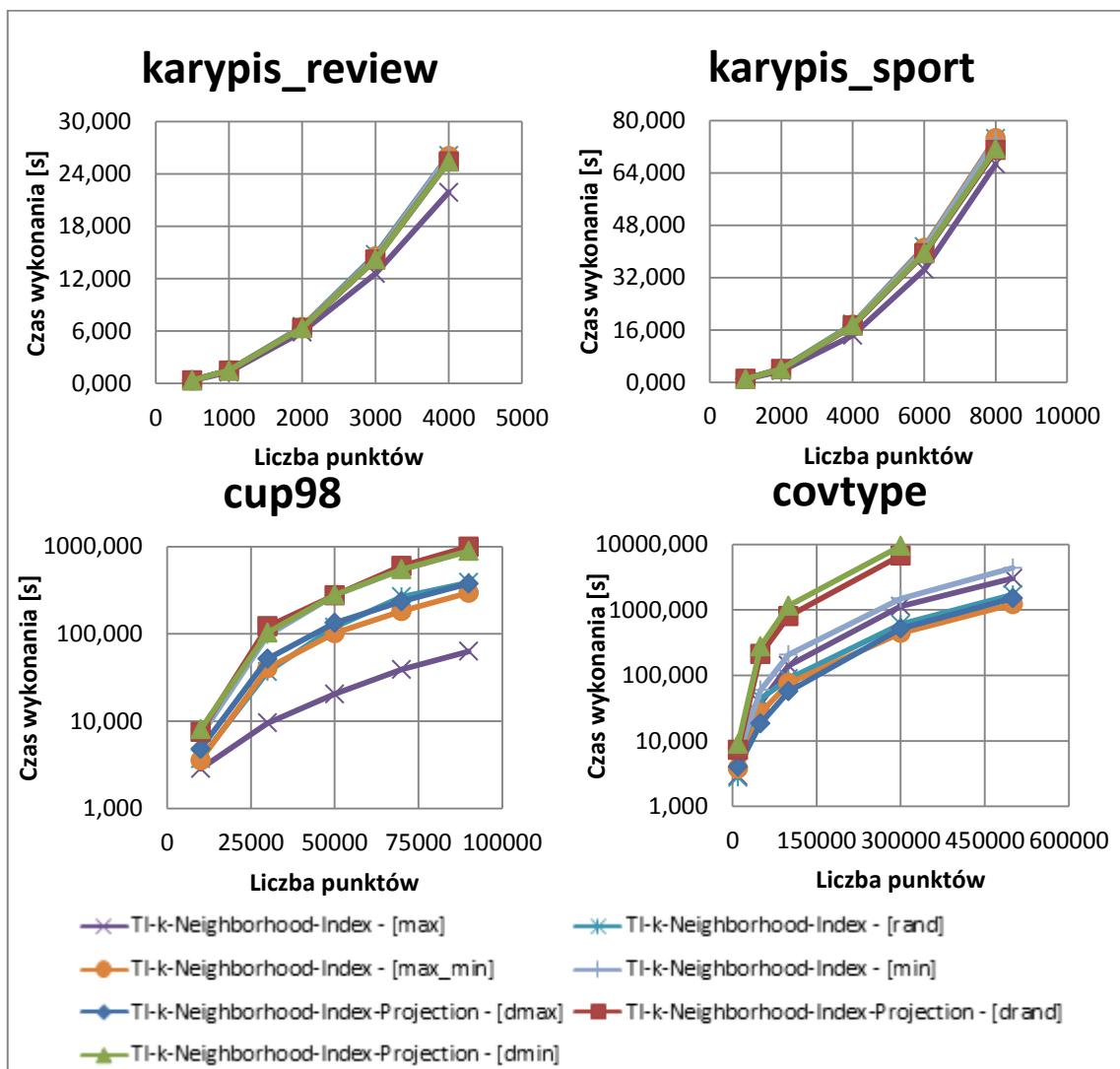
Tab. 34. Wyniki czasowe wykonania algorytmu TI-k-Neighborhood-Index-Projection dla metody rzutowania [dmin] przy zastosowaniu miary kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 50% losowo wybranych punktów zbioru danych

| zbior | l.p. | TI-k-Neighborhood-Index-Projection [dmin] | | | | | |
|----------------|--------|---|----------|-------|--------|-------|-------|
| | | wyk. | wysz. | bud. | obl. | sort. | norm. |
| | | alg. | | ind. | odl. | | |
| karypis_sport | 1000 | 1,054 | 1,051 | 0,000 | 0,001 | 0,000 | 0,002 |
| | 2000 | 4,059 | 4,054 | 0,000 | 0,001 | 0,000 | 0,004 |
| | 4000 | 17,436 | 17,424 | 0,000 | 0,004 | 0,000 | 0,009 |
| | 6000 | 39,509 | 39,490 | 0,000 | 0,006 | 0,000 | 0,013 |
| | 8000 | 71,359 | 71,335 | 0,000 | 0,007 | 0,000 | 0,017 |
| | 500 | 0,371 | 0,370 | 0,000 | 0,000 | 0,000 | 0,001 |
| karypis_review | 1000 | 1,490 | 1,486 | 0,000 | 0,001 | 0,000 | 0,003 |
| | 2000 | 6,350 | 6,342 | 0,000 | 0,002 | 0,000 | 0,006 |
| | 3000 | 14,210 | 14,198 | 0,000 | 0,003 | 0,000 | 0,009 |
| | 4000 | 25,409 | 25,392 | 0,000 | 0,004 | 0,000 | 0,013 |
| | 10000 | 9,005 | 8,745 | 0,000 | 0,253 | 0,002 | 0,005 |
| | 50000 | 276,036 | 273,289 | 0,001 | 2,706 | 0,015 | 0,025 |
| covtype | 100000 | 1154,916 | 1148,930 | 0,002 | 5,897 | 0,037 | 0,050 |
| | 300000 | 9440,579 | 9420,160 | 0,003 | 20,120 | 0,147 | 0,150 |
| | 500000 | - | - | - | - | - | - |
| | 10000 | 8,084 | 7,826 | 0,000 | 0,252 | 0,001 | 0,005 |
| | 30000 | 103,714 | 102,178 | 0,001 | 1,516 | 0,004 | 0,015 |
| | 50000 | 279,633 | 276,193 | 0,001 | 3,408 | 0,007 | 0,025 |
| cup98 | 70000 | 548,499 | 545,526 | 0,001 | 2,927 | 0,010 | 0,035 |
| | 90000 | 889,131 | 884,535 | 0,001 | 4,537 | 0,013 | 0,045 |

Uzyskane rezultaty mają podobny charakter do otrzymanych w rozdziale 6.2.2. Ich analiza prowadzi do wniosku, że liczność dziedziny wymiaru ma kluczowy wpływ na sprawność algorytmu *k-Neighborhood-Index-Projection*. Wyjaśnienie genezy tego wpływu zostało zamieszczone we wspomnianym wyżej rozdziale 6.2.2. W dalszej części pracy jako wyniki czasowe algorytmu *k-Neighborhood-Index-Projection* z zastosowaniem miary kosinusowej jako miary podobieństwa będą prezentowane rezultaty osiągnięte przy zastosowaniu rzutu na wymiar [dmax].

6.6.3. Porównanie algorytmów k-Neighborhood-Index-Projection z TI-k-Neighborhood-Index – miara kosinusowa

Na rys. 39 zamieściłem wykresy czasów wykonania algorytmów *k-Neighborhood-Index-Projection* i *TI-k-Neighborhood-Index*, wykorzystujące miarę kosinusową jako miarę podobieństwa, w funkcji liczby punktów we wszystkich badanych przypadkach zastosowania punktu referencyjnego i strategii rzutowania na wymiar. Dane zamieszczone na wykresach odpowiadają tym zgromadzonym w tabelach tab. 31, tab. 32, tab. 33 i tab. 34.



Rys. 39. Porównanie wydajności algorytmu *k-Neighborhood-Index-Projection* i *TI-k-Neighborhood-Index* w zależności od wybranego wymiaru rzutowania i punktu referencyjnego przy zastosowaniu miary kosinusowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań $k=5$ sąsiedztwa w przykładowych zbiorach danych dla 50% losowo wybranych punktów zbioru danych

Z otrzymanych rezultatów eksperymentów wynika, że dla danych tekstowych algorytm wykorzystujący rzutowanie wykonuje się nieznacznie wolniej niż *TI-k-Neighborhood-Index*. W przypadku gęstych zbiorów wyszukiwanie sąsiedztwa z zastosowaniem rzutowania na losowy wymiar i [dmin] wykonuje się kilka razy wolniej niż w pozostałych rozpatrywanych przypadkach. Warto zwrócić uwagę, że dla zbioru cup98 sprawność *k-neighborhood-Index-Projection* z rzutowaniem na [dmax] jest porównywalna ze sprawnością *TI-k-Neighborhood-Index* dla punktów referencyjnych różnych od [max]. Na niekorzyść rzutowania w porównaniu do punktu referencyjnego przemawia zależność tej metody od liczności dziedzin wymiarów.

6.6.4. Badania algorytmu TI-k-Neighborhood-Index-Ref – wybór dwóch punktów referencyjnych – miara kosinusowa

W celu wyznaczenia odpowiednich punktów referencyjnych dla algorytmu *TI-k-Neighborhood-Index* z zastosowaniem miary kosinusowej jako miary podobieństwa przeprowadziłem eksperymenty badające następujące pary punktów referencyjnych:

- [max][min];
- [max][max_min];
- [max][rand];
- [min][max];
- [max_min][max];
- [rand][max].

W tabelach tab. 35, tab. 36 i tab. 37 zamieściłem czasy uruchomień algorytmu *TI-k-Neighborhood-Index -Ref* wraz z trwaniem składających się na niego kroków. Na rys. 40 zaprezentowałem wykresy czasu wykonania algorytmu w funkcji liczby punktów.

Podobnie jak dla wyników uzyskanych w rozdziale 6.2.4. z otrzymanych rezultatów przeprowadzonych eksperymentów trudno jednoznacznie wskazać, która kombinacja punktów referencyjnych najbardziej przyspiesza wyznaczenie k sąsiedztwa. W większości przypadków eksperymenty z punktem maksymalnym jako pierwszym punktem referencyjnym wykonują się szybciej niż dla punktu losowego [rand] czy innego punktu skrajnego [min]. W przypadku zbiorów gęstych najgorsze rezultaty otrzymano gdy pierwszym punktem referencyjnym był [min], co zgodne jest z wnioskami, do których doszedłem w rozdziale 6.6.1.3.

Tab. 35. Porównanie wydajności TI-k-Neighborhood-Index-Ref dla par punktów [max][min] i [max][max_min] przy zastosowaniu miary kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach dla 50% losowo выбрanych punktów zbioru

| zbior | l.p. | TI-k-Neighborhood-Index-Ref [max][min] | | | | | | TI-k-Neighborhood-Index-Ref [max][max_min] | | | | | | TI-k-Neighborhood-Index-Ref [max][min] | | | | | | TI-k-Neighborhood-Index-Ref [max][max_min] | | | | | |
|----------------|--------|--|----------|-------|--------|-------|-------|--|---------|-------|--------|-------|-------|--|-------|-------|-------|-------|-------|--|-------|-------|-------|-------|------|
| | | wyk. | | wysz. | | bud. | | obl. | | sort. | | norm. | | wyk. | | wysz. | | bud. | | obl. | | sort. | | norm. | |
| | | alg. | ind. | alg. | ind. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. |
| karypis_sport | 1000 | 1,071 | 1,055 | 0,000 | 0,010 | 0,000 | 0,005 | 0,010 | 0,016 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | |
| | 2000 | 4,227 | 4,212 | 0,000 | 0,015 | 0,000 | 0,000 | 0,010 | 0,026 | 0,000 | 0,000 | 0,005 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | |
| | 4000 | 17,868 | 17,836 | 0,000 | 0,021 | 0,000 | 0,010 | 17,670 | 17,628 | 0,000 | 0,026 | 0,000 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | |
| | 6000 | 41,220 | 41,168 | 0,000 | 0,031 | 0,005 | 0,015 | 41,252 | 41,205 | 0,000 | 0,042 | 0,000 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | |
| | 8000 | 74,251 | 74,189 | 0,000 | 0,046 | 0,000 | 0,016 | 74,902 | 74,823 | 0,000 | 0,063 | 0,000 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | |
| | 500 | 0,375 | 0,375 | 0,000 | 0,000 | 0,000 | 0,000 | 0,375 | 0,375 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | |
| karypis_review | 1000 | 1,487 | 1,482 | 0,000 | 0,005 | 0,000 | 0,000 | 1,503 | 1,498 | 0,000 | 0,005 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 | |
| | 2000 | 6,568 | 6,542 | 0,000 | 0,016 | 0,000 | 0,010 | 6,510 | 6,489 | 0,000 | 0,015 | 0,000 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | |
| | 3000 | 14,570 | 14,545 | 0,000 | 0,015 | 0,000 | 0,010 | 14,571 | 14,544 | 0,000 | 0,021 | 0,000 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | |
| | 4000 | 26,010 | 25,964 | 0,000 | 0,031 | 0,000 | 0,015 | 26,058 | 26,011 | 0,000 | 0,031 | 0,000 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | |
| | 10000 | 4,233 | 3,889 | 0,000 | 0,344 | 0,000 | 0,000 | 2,147 | 1,955 | 0,000 | 0,187 | 0,000 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | |
| | 50000 | 19,865 | 16,796 | 0,000 | 3,032 | 0,016 | 0,021 | 22,812 | 19,438 | 0,000 | 3,339 | 0,015 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | |
| covtype | 100000 | 62,702 | 55,739 | 0,000 | 6,885 | 0,031 | 0,047 | 50,820 | 44,398 | 0,000 | 6,323 | 0,047 | 0,052 | 0,052 | 0,052 | 0,052 | 0,052 | 0,052 | 0,052 | 0,052 | 0,052 | 0,052 | 0,052 | 0,052 | |
| | 300000 | 419,734 | 397,234 | 0,005 | 22,194 | 0,146 | 0,156 | 279,969 | 257,452 | 0,000 | 22,220 | 0,151 | 0,146 | 0,146 | 0,146 | 0,146 | 0,146 | 0,146 | 0,146 | 0,146 | 0,146 | 0,146 | 0,146 | 0,146 | |
| | 500000 | 1196,521 | 1157,760 | 0,005 | 38,225 | 0,281 | 0,250 | 697,057 | 658,212 | 0,000 | 38,314 | 0,281 | 0,250 | 0,250 | 0,250 | 0,250 | 0,250 | 0,250 | 0,250 | 0,250 | 0,250 | 0,250 | 0,250 | 0,250 | |
| | 10000 | 4,103 | 3,833 | 0,000 | 0,270 | 0,000 | 0,000 | 3,192 | 2,984 | 0,000 | 0,197 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | |
| | 30000 | 36,248 | 34,476 | 0,000 | 1,747 | 0,010 | 0,015 | 32,480 | 31,013 | 0,000 | 1,435 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | 0,016 | |
| | 50000 | 91,239 | 87,532 | 0,000 | 3,666 | 0,021 | 0,020 | 82,982 | 79,456 | 0,000 | 3,489 | 0,016 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | 0,021 | |
| cup98 | 70000 | 159,714 | 155,975 | 0,000 | 3,671 | 0,031 | 0,037 | 147,280 | 142,470 | 0,000 | 4,742 | 0,031 | 0,036 | 0,036 | 0,036 | 0,036 | 0,036 | 0,036 | 0,036 | 0,036 | 0,036 | 0,036 | 0,036 | 0,036 | |
| | 90000 | 253,990 | 248,628 | 0,000 | 5,268 | 0,047 | 0,047 | 226,575 | 222,165 | 0,000 | 4,321 | 0,042 | 0,047 | 0,047 | 0,047 | 0,047 | 0,047 | 0,047 | 0,047 | 0,047 | 0,047 | 0,047 | 0,047 | 0,047 | |

Tab. 36. Porównanie wydajności TI-k-Neighborhood-Index-Ref dla par punktów [max][rand] i [min][max] przy zastosowaniu miary kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach dla 50% losowo wybranych punktów zbioru danych

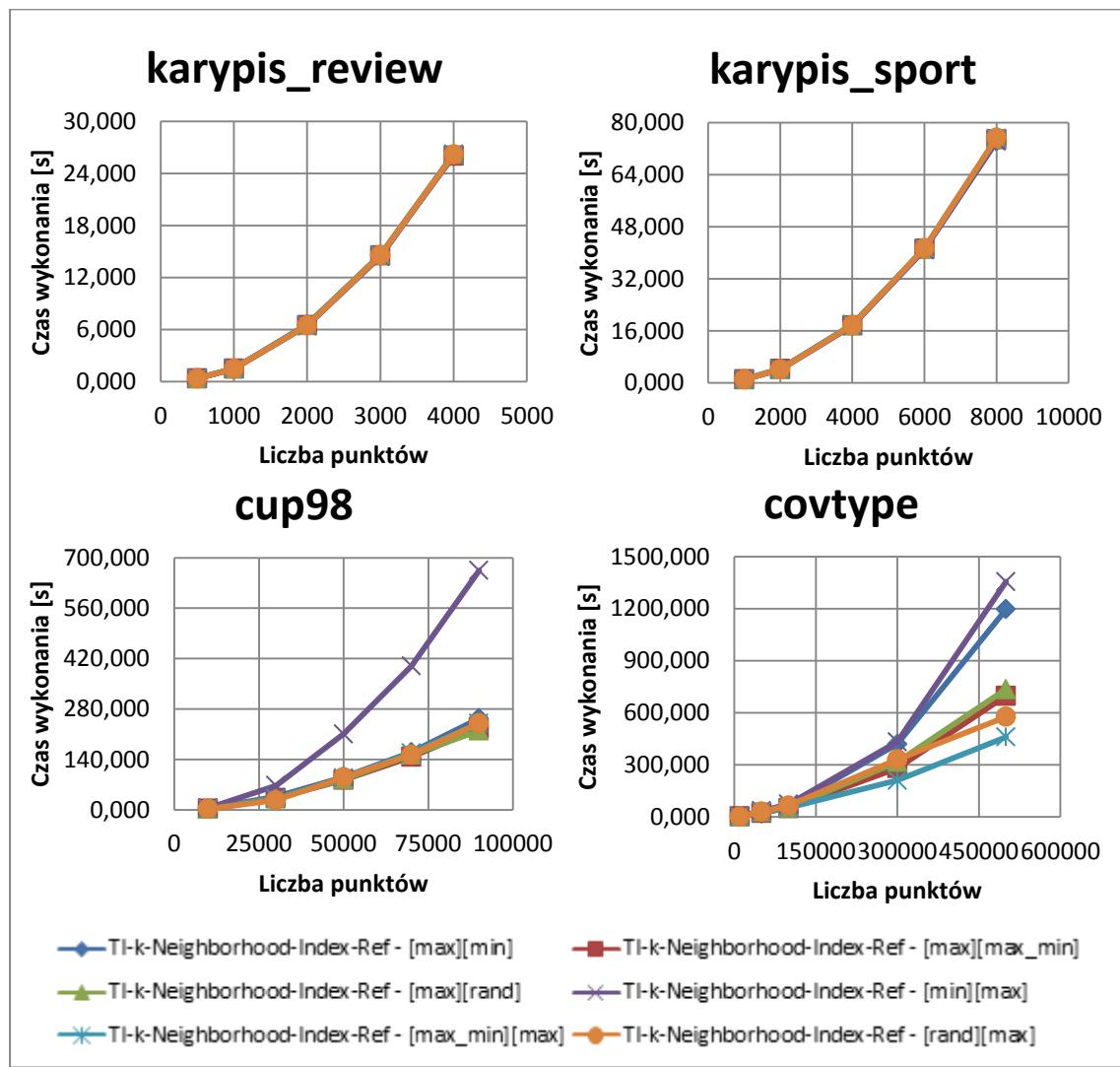
| zbior | l.p. | TI-k-Neighborhood-Index-Ref [max][rand] | | | | | | TI-k-Neighborhood-Index-Ref [min][max] | | | | | |
|----------------|--------|---|---------|-------|--------|-------|-------|--|----------|-------|--------|-------|-------|
| | | wyk. | wysz. | bud. | obl. | sort. | norm. | wyk. | wysz. | bud. | obl. | sort. | norm. |
| | | alg. | | ind. | odl. | | | alg. | | ind. | odl. | | |
| karypis_sport | 1000 | 1,077 | 1,061 | 0,000 | 0,016 | 0,000 | 0,000 | 1,070 | 1,060 | 0,000 | 0,007 | 0,000 | 0,002 |
| | 2000 | 4,228 | 4,207 | 0,000 | 0,011 | 0,005 | 0,005 | 4,094 | 4,079 | 0,000 | 0,010 | 0,001 | 0,004 |
| | 4000 | 17,685 | 17,654 | 0,000 | 0,031 | 0,000 | 0,000 | 17,550 | 17,516 | 0,000 | 0,023 | 0,002 | 0,009 |
| | 6000 | 41,283 | 41,252 | 0,000 | 0,016 | 0,000 | 0,016 | 40,867 | 40,814 | 0,000 | 0,038 | 0,002 | 0,013 |
| | 8000 | 75,069 | 74,989 | 0,000 | 0,061 | 0,002 | 0,017 | 74,315 | 74,242 | 0,000 | 0,054 | 0,003 | 0,017 |
| | 500 | 0,378 | 0,372 | 0,000 | 0,005 | 0,000 | 0,002 | 0,378 | 0,372 | 0,000 | 0,004 | 0,000 | 0,001 |
| karypis_review | 1000 | 1,505 | 1,494 | 0,000 | 0,008 | 0,000 | 0,003 | 1,502 | 1,492 | 0,000 | 0,007 | 0,000 | 0,003 |
| | 2000 | 6,511 | 6,487 | 0,000 | 0,017 | 0,001 | 0,006 | 6,422 | 6,400 | 0,000 | 0,015 | 0,000 | 0,006 |
| | 3000 | 14,640 | 14,607 | 0,000 | 0,023 | 0,001 | 0,009 | 14,417 | 14,383 | 0,000 | 0,023 | 0,001 | 0,010 |
| | 4000 | 26,247 | 26,197 | 0,000 | 0,036 | 0,001 | 0,013 | 26,215 | 26,169 | 0,000 | 0,033 | 0,001 | 0,013 |
| | 10000 | 2,039 | 1,890 | 0,000 | 0,142 | 0,003 | 0,005 | 2,276 | 2,135 | 0,000 | 0,133 | 0,003 | 0,005 |
| | 50000 | 29,925 | 26,694 | 0,001 | 3,189 | 0,016 | 0,025 | 34,667 | 31,438 | 0,001 | 3,187 | 0,016 | 0,025 |
| covtype | 100000 | 53,147 | 47,324 | 0,001 | 5,734 | 0,038 | 0,050 | 73,390 | 68,125 | 0,001 | 5,179 | 0,036 | 0,050 |
| | 300000 | 316,281 | 293,844 | 0,002 | 22,135 | 0,150 | 0,150 | 432,313 | 410,396 | 0,003 | 21,628 | 0,136 | 0,150 |
| | 500000 | 735,369 | 696,890 | 0,004 | 37,939 | 0,286 | 0,250 | 1354,993 | 1316,530 | 0,005 | 37,950 | 0,259 | 0,250 |
| | 10000 | 2,870 | 2,699 | 0,000 | 0,163 | 0,003 | 0,005 | 5,262 | 5,134 | 0,000 | 0,119 | 0,004 | 0,005 |
| | 30000 | 31,878 | 30,631 | 0,000 | 1,222 | 0,010 | 0,015 | 67,443 | 66,429 | 0,001 | 0,986 | 0,013 | 0,015 |
| | 50000 | 85,132 | 81,870 | 0,001 | 3,217 | 0,019 | 0,026 | 210,852 | 208,038 | 0,001 | 2,765 | 0,024 | 0,025 |
| cup98 | 70000 | 153,402 | 148,217 | 0,001 | 5,119 | 0,030 | 0,036 | 399,570 | 394,270 | 0,001 | 5,228 | 0,036 | 0,035 |
| | 90000 | 219,948 | 215,319 | 0,001 | 4,541 | 0,042 | 0,046 | 664,947 | 659,309 | 0,001 | 5,541 | 0,050 | 0,045 |

Tab. 37. Porównanie wydajności TI-k-Neighborhood-Index-Ref dla par punktów [max_min][max] i [rand][max] przy zastosowaniu miary kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach dla 50% losowo выбрanych punktów zbioru

| zbior | l.p. | TI-k-Neighborhood-Index-Ref [max_min][max] | | | | | | TI-k-Neighborhood-Index-Ref [rand][max] | | | | | | TI-k-Neighborhood-Index-Ref [rand][max] | | | | | | TI-k-Neighborhood-Index-Ref [rand][max] | | | | | |
|----------------|--------|--|---------|-------|--------|-------|-------|---|-------|-------|-------|-------|-------|---|---------|---------|-------|--------|-------|---|-------|-------|-------|-------|------|
| | | wyk. | | wysz. | | bud. | | obl. | | sort. | | norm. | | wyk. | | wysz. | | bud. | | obl. | | sort. | | norm. | |
| | | alg. | ind. | alg. | ind. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. | odl. |
| karypis_sport | 1000 | 1,079 | 1,068 | 0,000 | 0,009 | 0,000 | 0,002 | 0,002 | 0,002 | 0,002 | 0,002 | 0,002 | 0,002 | 0,002 | 1,075 | 1,063 | 0,000 | 0,010 | 0,000 | 0,002 | 0,002 | 0,002 | 0,002 | 0,002 | |
| | 2000 | 4,190 | 4,175 | 0,000 | 0,011 | 0,001 | 0,004 | 0,004 | 0,004 | 0,004 | 0,004 | 0,004 | 0,004 | 0,004 | 4,173 | 4,158 | 0,000 | 0,011 | 0,001 | 0,004 | 0,004 | 0,004 | 0,004 | 0,004 | |
| | 4000 | 17,961 | 17,925 | 0,000 | 0,026 | 0,001 | 0,008 | 0,008 | 0,008 | 0,008 | 0,008 | 0,008 | 0,008 | 0,008 | 17,761 | 17,726 | 0,000 | 0,026 | 0,001 | 0,008 | 0,008 | 0,008 | 0,008 | 0,008 | |
| | 6000 | 41,298 | 41,238 | 0,000 | 0,045 | 0,002 | 0,013 | 0,013 | 0,013 | 0,013 | 0,013 | 0,013 | 0,013 | 0,013 | 41,392 | 41,337 | 0,000 | 0,040 | 0,002 | 0,013 | 0,013 | 0,013 | 0,013 | 0,013 | |
| | 8000 | 75,206 | 75,132 | 0,000 | 0,055 | 0,002 | 0,017 | 0,017 | 0,017 | 0,017 | 0,017 | 0,017 | 0,017 | 0,017 | 75,321 | 75,238 | 0,000 | 0,062 | 0,003 | 0,017 | 0,017 | 0,017 | 0,017 | 0,017 | |
| | 500 | 0,380 | 0,373 | 0,000 | 0,005 | 0,000 | 0,002 | 0,002 | 0,002 | 0,002 | 0,002 | 0,002 | 0,002 | 0,002 | 0,378 | 0,372 | 0,000 | 0,005 | 0,000 | 0,001 | 0,001 | 0,001 | 0,001 | 0,001 | |
| karypis_review | 1000 | 1,504 | 1,493 | 0,000 | 0,008 | 0,000 | 0,003 | 0,003 | 0,003 | 0,003 | 0,003 | 0,003 | 0,003 | 0,003 | 1,503 | 1,492 | 0,000 | 0,008 | 0,000 | 0,003 | 0,003 | 0,003 | 0,003 | 0,003 | |
| | 2000 | 6,494 | 6,469 | 0,000 | 0,017 | 0,001 | 0,007 | 0,007 | 0,007 | 0,007 | 0,007 | 0,007 | 0,007 | 0,007 | 6,485 | 6,460 | 0,000 | 0,018 | 0,000 | 0,006 | 0,006 | 0,006 | 0,006 | 0,006 | |
| | 3000 | 14,578 | 14,546 | 0,000 | 0,022 | 0,001 | 0,009 | 0,009 | 0,009 | 0,009 | 0,009 | 0,009 | 0,009 | 0,009 | 14,578 | 14,544 | 0,000 | 0,024 | 0,001 | 0,009 | 0,009 | 0,009 | 0,009 | 0,009 | |
| | 4000 | 26,157 | 26,110 | 0,000 | 0,034 | 0,001 | 0,013 | 0,013 | 0,013 | 0,013 | 0,013 | 0,013 | 0,013 | 0,013 | 26,169 | 26,123 | 0,000 | 0,032 | 0,001 | 0,013 | 0,013 | 0,013 | 0,013 | 0,013 | |
| | 10000 | 1,297 | 1,183 | 0,000 | 0,106 | 0,003 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 1,336 | 1,237 | 0,000 | 0,091 | 0,003 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | |
| | 50000 | 28,046 | 25,405 | 0,001 | 2,600 | 0,015 | 0,025 | 0,025 | 0,025 | 0,025 | 0,025 | 0,025 | 0,025 | 0,025 | 29,401 | 26,667 | 0,001 | 2,692 | 0,016 | 0,025 | 0,025 | 0,025 | 0,025 | 0,025 | |
| covtype | 100000 | 52,639 | 47,016 | 0,001 | 5,536 | 0,035 | 0,050 | 0,050 | 0,050 | 0,050 | 0,050 | 0,050 | 0,050 | 0,050 | 64,901 | 59,692 | 0,001 | 5,122 | 0,036 | 0,050 | 0,050 | 0,050 | 0,050 | 0,050 | |
| | 300000 | 210,182 | 188,005 | 0,003 | 21,891 | 0,133 | 0,150 | 0,150 | 0,150 | 0,150 | 0,150 | 0,150 | 0,150 | 0,150 | 331,224 | 308,781 | 0,003 | 22,142 | 0,148 | 0,150 | 0,150 | 0,150 | 0,150 | 0,150 | |
| | 500000 | 459,830 | 421,472 | 0,005 | 37,843 | 0,260 | 0,250 | 0,250 | 0,250 | 0,250 | 0,250 | 0,250 | 0,250 | 0,250 | 576,918 | 538,730 | 0,007 | 37,649 | 0,283 | 0,250 | 0,250 | 0,250 | 0,250 | 0,250 | |
| | 10000 | 2,725 | 2,603 | 0,000 | 0,114 | 0,003 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | 2,544 | 2,446 | 0,000 | 0,090 | 0,003 | 0,005 | 0,005 | 0,005 | 0,005 | 0,005 | |
| | 30000 | 30,845 | 29,990 | 0,000 | 0,829 | 0,011 | 0,015 | 0,015 | 0,015 | 0,015 | 0,015 | 0,015 | 0,015 | 0,015 | 26,685 | 25,890 | 0,000 | 0,770 | 0,010 | 0,015 | 0,015 | 0,015 | 0,015 | 0,015 | |
| | 50000 | 88,627 | 85,711 | 0,001 | 2,871 | 0,019 | 0,025 | 0,025 | 0,025 | 0,025 | 0,025 | 0,025 | 0,025 | 0,025 | 90,060 | 87,428 | 0,001 | 2,587 | 0,019 | 0,025 | 0,025 | 0,025 | 0,025 | 0,025 | |
| cup98 | 70000 | 158,042 | 152,886 | 0,001 | 5,090 | 0,030 | 0,036 | 0,036 | 0,036 | 0,036 | 0,036 | 0,036 | 0,036 | 0,036 | 153,026 | 148,148 | 0,001 | 4,812 | 0,030 | 0,035 | 0,035 | 0,035 | 0,035 | 0,035 | |
| | 90000 | 240,591 | 234,696 | 0,001 | 5,807 | 0,041 | 0,046 | 0,046 | 0,046 | 0,046 | 0,046 | 0,046 | 0,046 | 0,046 | 240,332 | 233,848 | 0,001 | 6,393 | 0,044 | 0,046 | 0,046 | 0,046 | 0,046 | 0,046 | |

Różnice w czasach wykonania algorytmu dla poszczególnych zestawów punktów referencyjnych są na tyle niewielkie, że nie pozwalają jednoznacznie wskazać, która z badanych par punktów referencyjnych najbardziej przyspiesza wykonanie algorytmu. Tym co wyniki pozwalają stwierdzić jest wniosek, że jako pierwszy punkt referencyjny najlepiej wybrać [max].

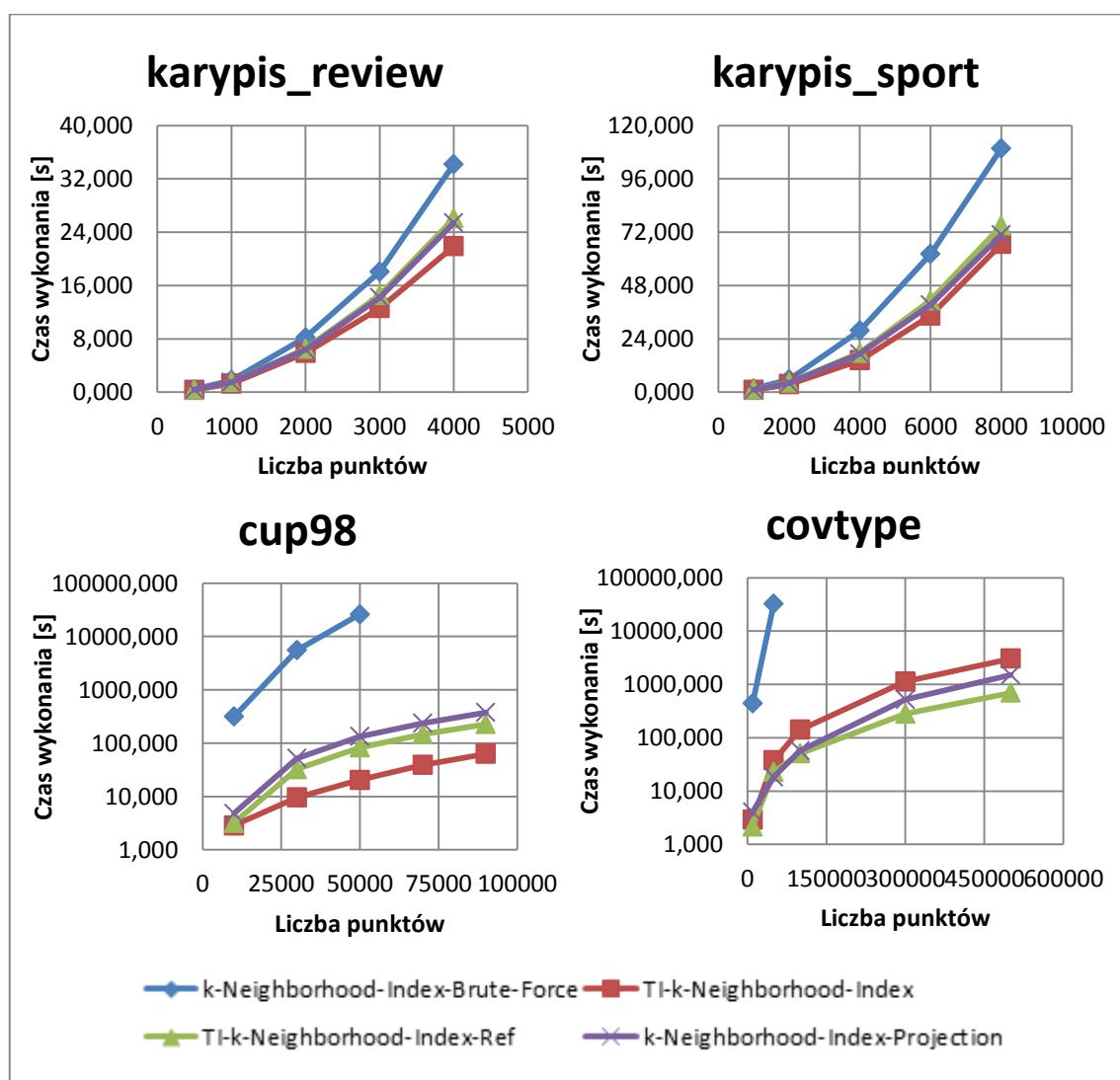
Biorąc pod uwagę wniosek z rozdziału 6.6.1.3., wskazujący że najgorszym punktem referencyjnym jest [min], w dalszej części pracy jako wyniki czasowe algorytmu *TI-k-Neighborhood-Index-Ref* będą prezentowane rezultaty osiągnięte dla punktów referencyjnych [max][max_min].



Rys. 40. Porównanie wydajności algorytmu *TI-k-Neighborhood-Index-Ref* w zależności od wybranej pary punktów referencyjnych przy zastosowaniu miary kosinusowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań $k=5$ sąsiedztwa w przykładowych zbiorach danych dla 50% losowo wybranych punktów zbioru danych

6.6.5. Porównanie implementacji odmian algorytmu k-Neighborhood-Index – miara kosinusowa

W tab. 38, tab. 39 i na rys. 41 zamieściłem rezultaty badań odmian algorytmu *k-Neighborhood-Index* z zastosowaniem miary kosinusowej jako miary podobieństwa. Zamieszczone wyniki algorytmu *TI-k-Neighborhood-Index* zostały zebrane dla punktu referencyjnego [max], *k-Neighborhood-Index-Projection* dla [dmax], natomiast rezultaty *TI-k-Neighborhood-Ref* dla punktów referencyjnych [max][rand]. Algorytm *k-Neighborhood-Index-Brute-Force* jest brutalną implementacją wyszukiwania k sąsiedztwa o złożoności kwadratowej.



Rys. 41. Porównanie wydajności odmian algorytmu k-Neighborhood- przy zastosowaniu miary kosinusowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 50% losowo wybranych punktów zbioru danych

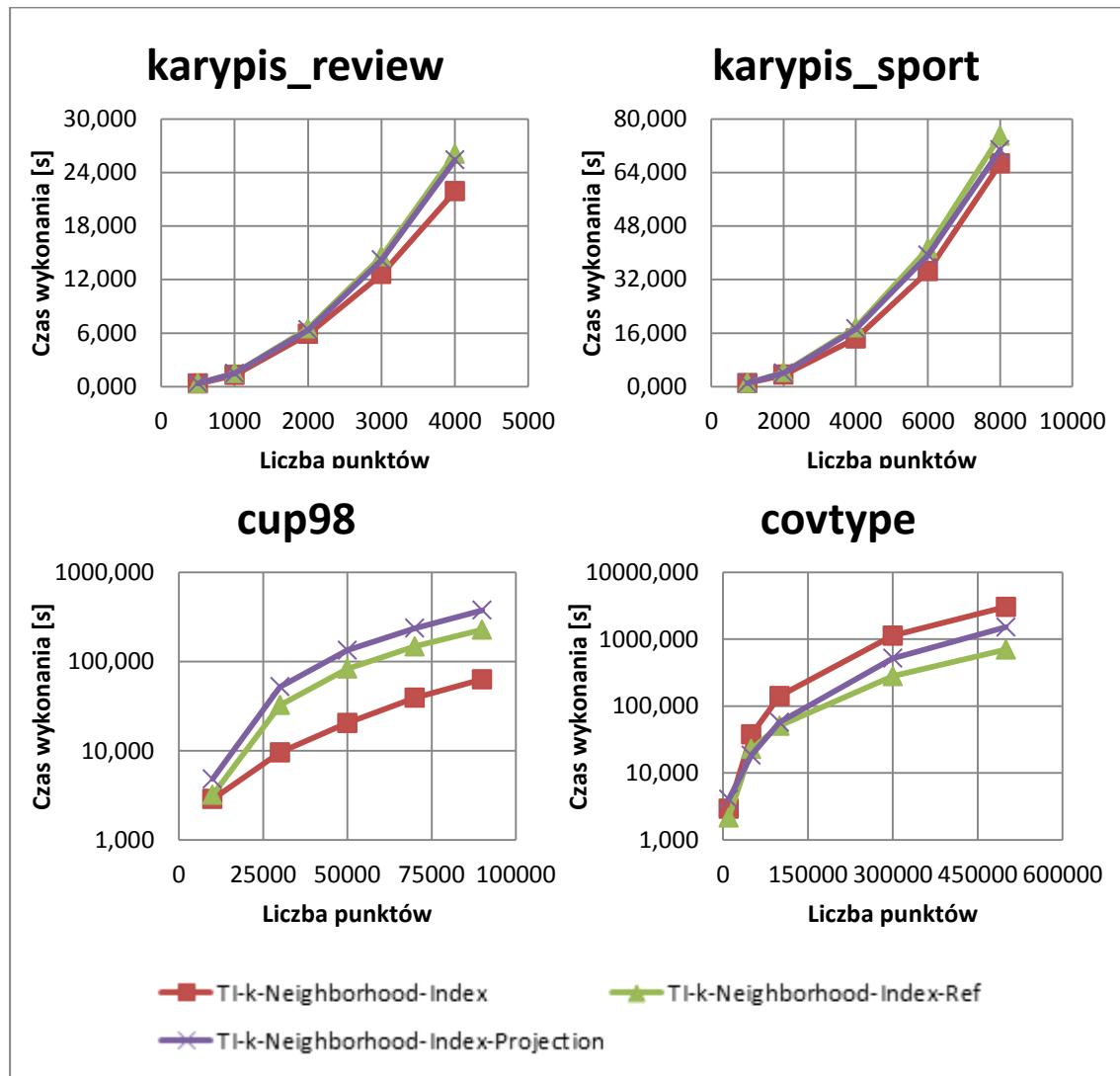
Tab. 38. Porównanie wydajności algorytmów k-Neighborhood-Index-Brute-Force i TI-k-Neighborhood-Index przy zastosowaniu miary kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach dla 50% losowo wybranych punktów zbioru danych

| zbior | l.p. | k-Neighborhood-Index-Brute-Force | | | | | | TI-k-Neighborhood-Index [max] | | | | | |
|----------------|--------|----------------------------------|-----------|-------|-------|-------|-------|-------------------------------|----------|-------|--------|-------|-------|
| | | wyk. | wysz. | bud. | obl. | sort. | norm. | wyk. | wysz. | bud. | obl. | sort. | norm. |
| | | alg. | | ind. | odl. | | | alg. | | ind. | odl. | | |
| karypis_sport | 1000 | 1,607 | 1,607 | 0,000 | 0,000 | 0,000 | 0,005 | 1,034 | 1,027 | 0,000 | 0,005 | 0,000 | 0,002 |
| | 2000 | 5,673 | 5,673 | 0,000 | 0,016 | 0,000 | 0,010 | 3,519 | 3,510 | 0,000 | 0,005 | 0,001 | 0,004 |
| | 4000 | 27,675 | 27,675 | 0,000 | 0,000 | 0,000 | 0,015 | 14,359 | 14,337 | 0,000 | 0,012 | 0,001 | 0,009 |
| | 6000 | 62,077 | 62,077 | 0,000 | 0,016 | 0,000 | 0,010 | 34,451 | 34,418 | 0,000 | 0,018 | 0,001 | 0,013 |
| | 8000 | 109,694 | 109,694 | 0,000 | 0,016 | 0,000 | 0,016 | 66,656 | 66,607 | 0,000 | 0,029 | 0,003 | 0,017 |
| karypis_review | 500 | 0,437 | 0,437 | 0,000 | 0,000 | 0,000 | 0,005 | 0,337 | 0,333 | 0,000 | 0,003 | 0,000 | 0,002 |
| | 1000 | 1,784 | 1,784 | 0,000 | 0,000 | 0,000 | 0,000 | 1,293 | 1,286 | 0,000 | 0,004 | 0,000 | 0,003 |
| | 2000 | 8,211 | 8,211 | 0,000 | 0,015 | 0,000 | 0,010 | 5,894 | 5,878 | 0,000 | 0,008 | 0,001 | 0,007 |
| | 3000 | 18,070 | 18,070 | 0,000 | 0,005 | 0,000 | 0,015 | 12,591 | 12,569 | 0,000 | 0,011 | 0,001 | 0,010 |
| | 4000 | 34,185 | 34,185 | 0,000 | 0,016 | 0,000 | 0,016 | 21,877 | 21,847 | 0,000 | 0,016 | 0,001 | 0,013 |
| covtype | 10000 | 437,508 | 437,508 | 0,000 | 0,281 | 0,000 | 0,005 | 2,913 | 2,553 | 0,000 | 0,322 | 0,003 | 0,005 |
| | 50000 | 32193,500 | 32193,500 | 0,000 | 1,700 | 0,000 | 0,031 | 37,549 | 34,851 | 0,001 | 2,934 | 0,016 | 0,025 |
| | 100000 | - | - | - | - | - | - | 141,164 | 134,889 | 0,001 | 6,105 | 0,038 | 0,050 |
| | 300000 | - | - | - | - | - | - | 1128,400 | 1106,590 | 0,003 | 21,494 | 0,147 | 0,150 |
| | 500000 | - | - | - | - | - | - | 3034,090 | 2997,940 | 0,003 | 35,633 | 0,275 | 0,250 |
| cup98 | 10000 | 315,781 | 315,781 | 0,000 | 0,305 | 0,000 | 0,005 | 2,868 | 2,414 | 0,000 | 0,447 | 0,003 | 0,005 |
| | 30000 | 5512,320 | 5512,320 | 0,001 | 1,329 | 0,001 | 0,015 | 9,568 | 8,799 | 0,000 | 0,744 | 0,010 | 0,015 |
| | 50000 | 26015,500 | 26015,500 | 0,000 | 3,229 | 0,000 | 0,016 | 20,456 | 18,402 | 0,001 | 2,009 | 0,019 | 0,025 |
| | 70000 | - | - | - | - | - | - | 39,183 | 35,322 | 0,001 | 3,796 | 0,029 | 0,035 |
| | 90000 | - | - | - | - | - | - | 63,047 | 58,520 | 0,001 | 4,440 | 0,039 | 0,046 |

Tab. 39. Porównanie wydajności TI-k-Neighborhood-Index-Projection i TI-k-Neighborhood-Index-Ref przy zastosowaniu miary kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach dla 50% losowo wybranych punktów zbioru danych

| zbior | l.p. | TI-k-Neighborhood-Index-Projection [dmax] | | | | | | TI-k-Neighborhood-Index-Ref [max][max_min] | | | | | |
|----------------|--------|---|----------|-------|--------|-------|-------|--|---------|-------|--------|-------|-------|
| | | wyk. | wysz. | bud. | obl. | sort. | norm. | wyk. | wysz. | bud. | obl. | sort. | norm. |
| | | alg. | | ind. | odl. | | | alg. | | ind. | odl. | | |
| karypis_sport | 1000 | 1,047 | 1,044 | 0,000 | 0,001 | 0,000 | 0,002 | 1,077 | 1,061 | 0,000 | 0,016 | 0,000 | 0,000 |
| | 2000 | 4,038 | 4,032 | 0,000 | 0,002 | 0,000 | 0,004 | 4,280 | 4,274 | 0,000 | 0,005 | 0,000 | 0,000 |
| | 4000 | 17,224 | 17,211 | 0,000 | 0,005 | 0,000 | 0,008 | 17,670 | 17,628 | 0,000 | 0,026 | 0,000 | 0,016 |
| | 6000 | 39,275 | 39,254 | 0,000 | 0,008 | 0,000 | 0,013 | 41,252 | 41,205 | 0,000 | 0,042 | 0,000 | 0,005 |
| | 8000 | 70,763 | 70,736 | 0,001 | 0,008 | 0,000 | 0,017 | 74,902 | 74,823 | 0,000 | 0,063 | 0,000 | 0,016 |
| | 500 | 0,371 | 0,369 | 0,000 | 0,001 | 0,000 | 0,001 | 0,375 | 0,375 | 0,000 | 0,000 | 0,000 | 0,000 |
| karypis_review | 1000 | 1,482 | 1,478 | 0,000 | 0,001 | 0,000 | 0,003 | 1,503 | 1,498 | 0,000 | 0,005 | 0,000 | 0,000 |
| | 2000 | 6,357 | 6,349 | 0,000 | 0,002 | 0,000 | 0,006 | 6,510 | 6,489 | 0,000 | 0,015 | 0,000 | 0,005 |
| | 3000 | 14,186 | 14,173 | 0,000 | 0,003 | 0,000 | 0,010 | 14,571 | 14,544 | 0,000 | 0,021 | 0,000 | 0,005 |
| | 4000 | 25,388 | 25,371 | 0,000 | 0,003 | 0,000 | 0,013 | 26,058 | 26,011 | 0,000 | 0,031 | 0,000 | 0,016 |
| | 10000 | 4,032 | 3,698 | 0,000 | 0,326 | 0,003 | 0,005 | 2,147 | 1,955 | 0,000 | 0,187 | 0,000 | 0,005 |
| | 50000 | 18,396 | 15,797 | 0,001 | 2,558 | 0,015 | 0,025 | 22,812 | 19,438 | 0,000 | 3,339 | 0,015 | 0,021 |
| covtype | 100000 | 57,044 | 50,757 | 0,002 | 6,201 | 0,034 | 0,050 | 50,820 | 44,398 | 0,000 | 6,323 | 0,047 | 0,052 |
| | 300000 | 517,584 | 496,146 | 0,004 | 21,144 | 0,141 | 0,150 | 279,969 | 257,452 | 0,000 | 22,220 | 0,151 | 0,146 |
| | 500000 | 1506,937 | 1470,180 | 0,004 | 36,233 | 0,270 | 0,250 | 697,057 | 658,212 | 0,000 | 38,314 | 0,281 | 0,250 |
| | 10000 | 4,819 | 4,571 | 0,000 | 0,239 | 0,003 | 0,005 | 3,192 | 2,984 | 0,000 | 0,197 | 0,005 | 0,005 |
| | 30000 | 52,023 | 50,588 | 0,001 | 1,409 | 0,010 | 0,015 | 32,480 | 31,013 | 0,000 | 1,435 | 0,016 | 0,016 |
| | 50000 | 133,558 | 130,449 | 0,001 | 3,064 | 0,019 | 0,025 | 82,982 | 79,456 | 0,000 | 3,489 | 0,016 | 0,021 |
| cup98 | 70000 | 235,906 | 232,374 | 0,001 | 3,468 | 0,029 | 0,035 | 147,280 | 142,470 | 0,000 | 4,742 | 0,031 | 0,036 |
| | 90000 | 375,325 | 370,919 | 0,001 | 4,320 | 0,040 | 0,045 | 226,575 | 222,165 | 0,000 | 4,321 | 0,042 | 0,047 |

Na rys. 42 przedstawiłem wykresy zaprezentowane na rys. 41 z pominięciem tych, które zostały uzyskane dla algorytmu *k-Neighborhood-Index-Brute-Force* dla uwidocznienia różnic między pozostałymi eksperymentami. Rezultaty wykazują, że zastosowanie rzutowania na wymiar nie przyspiesza wyszukiwania k sąsiedztwa bardziej niż wykorzystanie nierówności trójkąta.



Rys. 42. Porównanie wydajności algorytmów k-Neighborhood-Index-Projection, TI-k-Neighborhood-Index i TI-k-Neighborhood-Index-Ref przy zastosowaniu miary kosinusowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań k=5 sąsiedztwa w przykładowych zbiorach danych dla 50% losowo wybranych punktów zbioru danych

6.7. Badania algorytmu kNN-Index-Vp-Tree – miara kosinusowa

W bieżącym rozdziale zaprezentowałem wyniki badań implementacji algorytmu *kNN-Index-Vp-Tree* wykorzystującej miarę kosinusową jako miarę podobieństwa. W swoich eksperymentach skupiłem się na metodach przeszukiwania indeksu metrycznego w celu wyznaczenia k sąsiedztwa danego zapytania. Badałem również wpływ implementacji punktu na wydajność algorytmu.

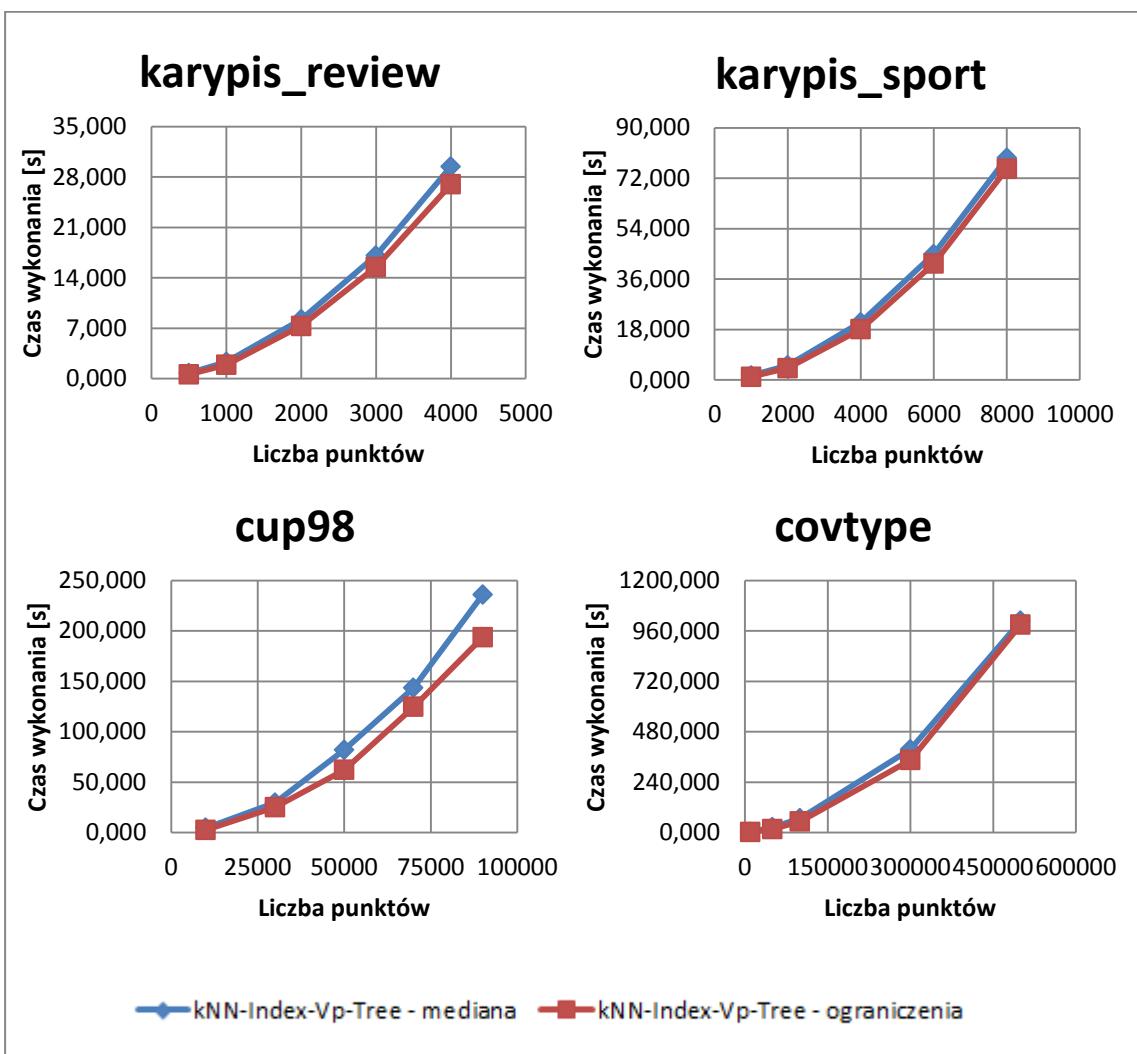
6.7.1. Implementacja algorytmu

Analogicznie do eksperymentów wykonanych w rozdziale 6.3.1., w którym jako miarę podobieństwa wykorzystałem odległość euklidesową, na przykładowych zbiorach danych zbadałem użyteczność metody mediany i metody ograniczeń w wyznaczaniu k sąsiadów w oparciu o indeks metryczny. W eksperymentach posłużyłem się miarą kosinusową jako miarą podobieństwa. Przeszukiwanie indeksu metrycznego metodą mediany i metodą ograniczeń opisałem w rozdziale 6.3.1., a w rozdziale 5.3.1. zamieściłem ich pseudokod.

W tab.40 umieściłem czasy uruchomień implementacji algorytmu *kNN-Index-Vp-Tree* dla różnych metod przeszukiwania indeksu metrycznego oraz czasy trwania kroków składających się na algorytm. Na rys. 43 zmieściłem wykresy czasów uruchomień implementacji algorytmu *kNN-Index-Vp-Tree* dla obu metod wyszukiwania k sąsiedztwa w funkcji liczby punktów.

Z rezultatów badań wynika, że *metoda ograniczeń* pozwala na szybsze wyznaczenie k sąsiedztwa niż *metoda mediany*. Największa różnica w czasach wykonania algorytmów występuje dla wyszukiwania k sąsiedztwa spośród 90000 punktów zbioru cup98. Dla tego szczególnego przypadku implementacja korzystająca z *metody ograniczeń* wykonuje się blisko 20% szybciej niż implementacja stosująca *metodę mediany*. Uzyskane wyniki mają podobny charakter do tych otrzymanych w rozdziale 6.3.1. Warto jednak zwrócić uwagę, że różnice w czasie wyznaczania k sąsiedztwa obu metodami są większe dla rezultatów otrzymanych, gdy miarą podobieństwa jest odległość euklidesowa.

Z uwagi na otrzymane rezultaty, w dalszej części rozważań będę się odnosił do algorytmu *kNN-Index-Vp-Tree* stosującego miarę kosinusową jako miarę podobieństwa jako do implementacji wykorzystującej metodę ograniczeń.



Rys. 43. Porównanie wydajności algorytmu kNN-Index-Vp-Tree w zależności od implementacji metody wyszukiwania w indeksie metrycznym przy zastosowaniu miary kosinusowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań $k=5$ sąsiadów w przykładowych zbiorach dla 50% losowo wybranych punktów zbioru danych

Tab. 40. Porównanie wydajności algorytmu kNN-Index-Vp-Tree w zależności od implementacji metody wyszukiwania w indeksie metrycznym przy zastosowaniu odległości kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiadów w przykładowych zbiorach dla 50% losowo wybranych punktów zbioru danych

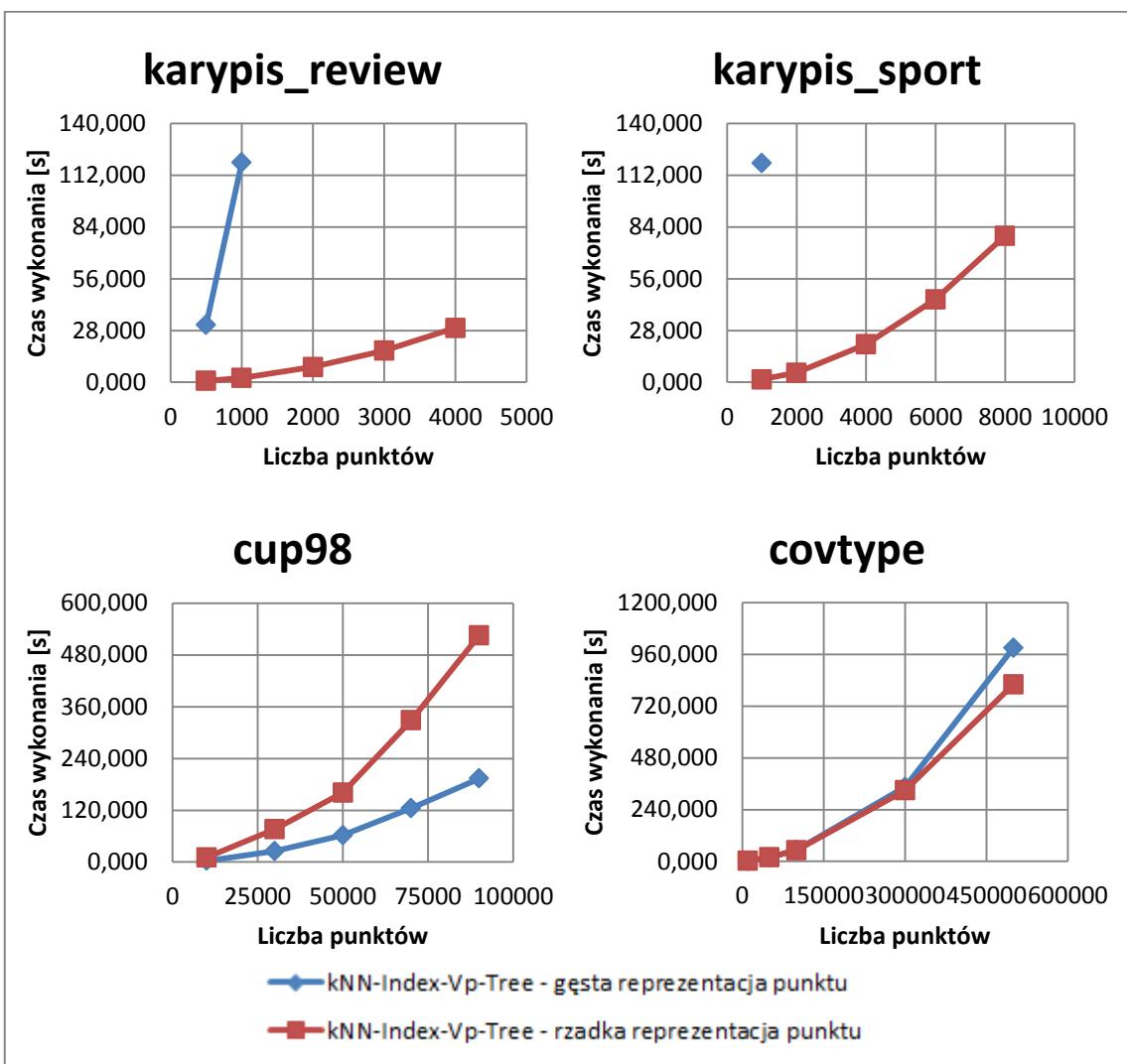
| zbior | l.p. | kNN-Index-Vp-Tree mediana | | | | kNN-Index-Vp-Tree ograniczenia | | | |
|----------------|--------|---------------------------|---------|---------|-------|--------------------------------|---------|---------|-------|
| | | wyk. | wysz. | bud. | norm. | wyk. | wysz. | bud. | norm. |
| | | alg. | | ind. | | alg. | | ind. | |
| karypis_sport | 1000 | 1,626 | 1,086 | 0,538 | 0,002 | 1,145 | 1,059 | 0,083 | 0,002 |
| | 2000 | 5,197 | 4,216 | 0,977 | 0,004 | 4,252 | 4,084 | 0,164 | 0,004 |
| | 4000 | 20,588 | 18,772 | 1,807 | 0,009 | 18,095 | 17,719 | 0,367 | 0,009 |
| | 6000 | 44,894 | 42,438 | 2,443 | 0,013 | 41,553 | 40,977 | 0,563 | 0,013 |
| | 8000 | 79,216 | 76,535 | 2,663 | 0,018 | 75,349 | 74,594 | 0,737 | 0,018 |
| karypis_review | 500 | 0,789 | 0,379 | 0,409 | 0,002 | 0,599 | 0,375 | 0,223 | 0,001 |
| | 1000 | 2,330 | 1,520 | 0,807 | 0,003 | 1,930 | 1,509 | 0,418 | 0,003 |
| | 2000 | 8,247 | 6,672 | 1,568 | 0,006 | 7,335 | 6,598 | 0,731 | 0,007 |
| | 3000 | 17,121 | 14,982 | 2,130 | 0,009 | 15,511 | 14,728 | 0,774 | 0,009 |
| | 4000 | 29,455 | 26,796 | 2,646 | 0,013 | 26,991 | 26,361 | 0,616 | 0,013 |
| covtype | 10000 | 2,981 | 1,815 | 1,160 | 0,006 | 2,135 | 1,682 | 0,447 | 0,006 |
| | 50000 | 25,051 | 18,805 | 6,213 | 0,033 | 16,985 | 14,367 | 2,585 | 0,033 |
| | 100000 | 67,692 | 50,994 | 16,631 | 0,067 | 53,101 | 33,074 | 19,969 | 0,059 |
| | 300000 | 395,891 | 336,354 | 59,345 | 0,192 | 345,252 | 248,231 | 96,841 | 0,181 |
| | 500000 | 1008,160 | 879,765 | 128,072 | 0,323 | 989,915 | 812,569 | 177,028 | 0,318 |
| cup98 | 10000 | 4,720 | 2,813 | 1,900 | 0,007 | 2,637 | 2,214 | 0,417 | 0,006 |
| | 30000 | 29,552 | 25,197 | 4,335 | 0,020 | 25,135 | 22,861 | 2,255 | 0,019 |
| | 50000 | 81,973 | 72,735 | 9,205 | 0,034 | 62,006 | 56,783 | 5,191 | 0,032 |
| | 70000 | 143,418 | 131,709 | 11,661 | 0,048 | 124,572 | 117,935 | 6,592 | 0,045 |
| | 90000 | 235,859 | 221,433 | 14,365 | 0,061 | 193,587 | 186,308 | 7,220 | 0,058 |

6.7.2. Implementacja struktury punktu

Analogicznie do badań przeprowadzonych w rozdziale 6.3.2. przetestowałem wydajność *kNN-Index-Vp-Tree* w zależności od implementacji punktu, gdy miarą podobieństwa jest miara kosinusowa. W tab. 41 oraz na rys. 44 zamieszczono wyniki uruchomień algorytmu *kNN-Index-Vp-Tree* z użyciem zarówno gęstej jak i rzadkiej implementacji punktu.

Tab. 41. Porównanie wydajności algorytmu kNN-Index-Vp-Tree w zależności od implementacji punktu przy zastosowaniu miary kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiadów w przykładowych zbiorach dla 50% losowo wybranych punktów zbioru danych

| zbior | l.p. | <i>kNN-Index-Vp-Tree</i> | | | | <i>kNN-Index-Vp-Tree</i> | | | |
|----------------|--------|----------------------------|---------|---------|-------|-----------------------------|---------|--------|-------|
| | | gęsta reprezentacja punktu | | | | rzadka reprezentacja punktu | | | |
| | | wyk. | wysz. | bud. | norm. | wyk. | wysz. | bud. | norm. |
| | | alg. | | ind. | | alg. | | ind. | |
| karypis_sport | 1000 | 118,578 | 112,775 | 5,779 | 1,139 | 1,626 | 1,086 | 0,538 | 0,002 |
| | 2000 | - | - | - | - | 5,197 | 4,216 | 0,977 | 0,004 |
| | 4000 | - | - | - | - | 20,588 | 18,772 | 1,807 | 0,009 |
| | 6000 | - | - | - | - | 44,894 | 42,438 | 2,443 | 0,013 |
| | 8000 | - | - | - | - | 79,216 | 76,535 | 2,663 | 0,018 |
| karypis_review | 500 | 31,088 | 28,282 | 2,821 | 0,570 | 0,789 | 0,379 | 0,409 | 0,002 |
| | 1000 | 118,821 | 113,089 | 5,797 | 1,136 | 2,330 | 1,520 | 0,807 | 0,003 |
| | 2000 | - | - | - | - | 8,247 | 6,672 | 1,568 | 0,006 |
| | 3000 | - | - | - | - | 17,121 | 14,982 | 2,130 | 0,009 |
| | 4000 | - | - | - | - | 29,455 | 26,796 | 2,646 | 0,013 |
| covtype | 10000 | 2,135 | 1,682 | 0,447 | 0,006 | 2,576 | 1,883 | 0,689 | 0,003 |
| | 50000 | 16,985 | 14,367 | 2,585 | 0,033 | 19,315 | 15,803 | 3,536 | 0,014 |
| | 100000 | 53,101 | 33,074 | 19,969 | 0,059 | 51,776 | 45,576 | 6,173 | 0,030 |
| | 300000 | 345,252 | 248,231 | 96,841 | 0,181 | 329,558 | 293,785 | 36,307 | 0,088 |
| | 500000 | 989,915 | 812,569 | 177,028 | 0,318 | 820,811 | 738,432 | 87,895 | 0,147 |
| cup98 | 10000 | 2,637 | 2,214 | 0,417 | 0,006 | 10,505 | 9,529 | 1,058 | 0,009 |
| | 30000 | 25,135 | 22,861 | 2,255 | 0,019 | 76,141 | 71,666 | 4,129 | 0,027 |
| | 50000 | 62,006 | 56,783 | 5,191 | 0,032 | 160,719 | 154,914 | 5,780 | 0,047 |
| | 70000 | 124,572 | 117,935 | 6,592 | 0,045 | 328,221 | 322,200 | 6,286 | 0,066 |
| | 90000 | 193,587 | 186,308 | 7,220 | 0,058 | 525,102 | 517,221 | 7,548 | 0,086 |

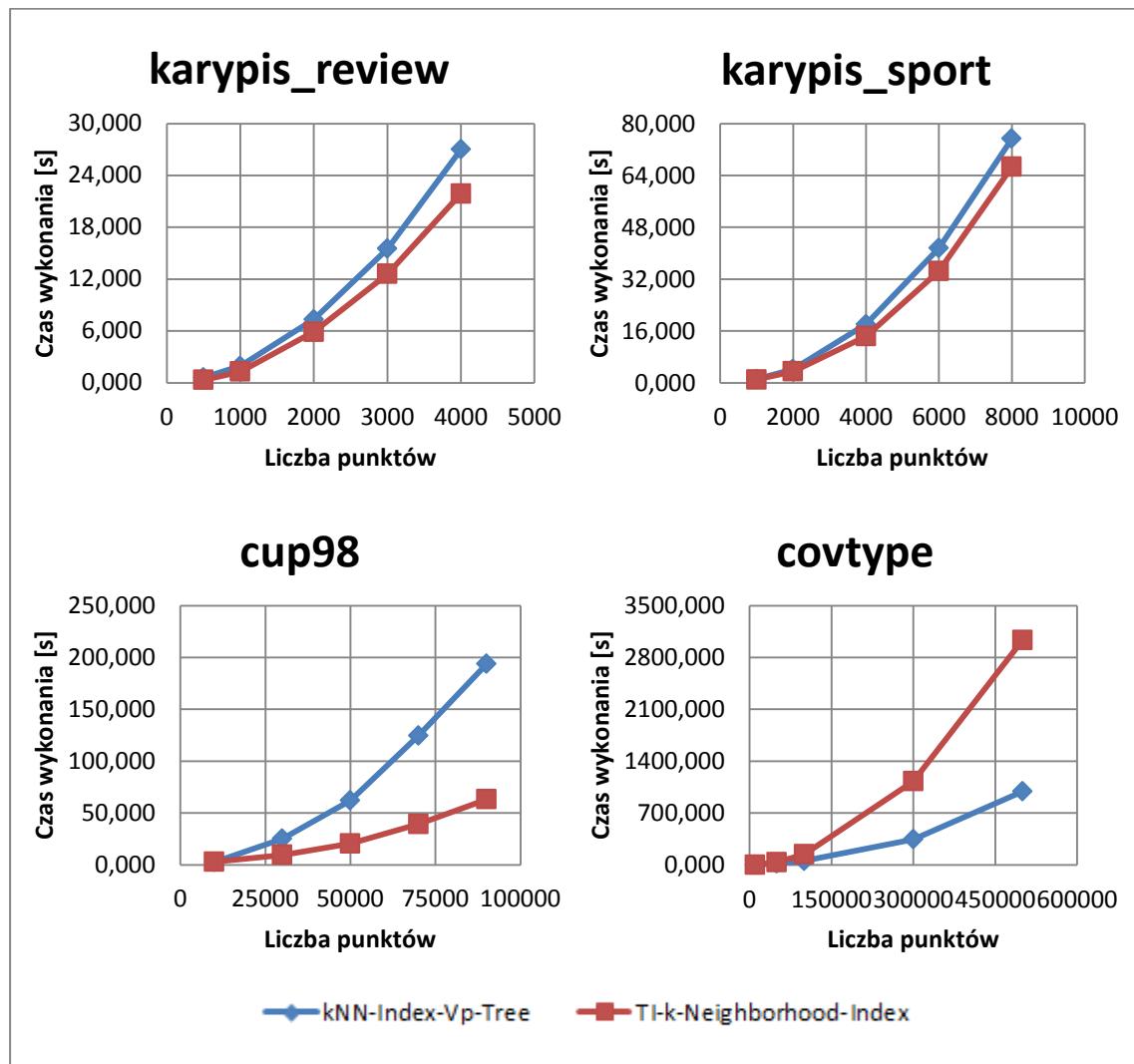


Rys. 44. Porównanie wydajności algorytmu kNN-Index-Vp-Tree w zależności od implementacji punktu przy zastosowaniu miary kosinusowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań k=5 sąsiadów w przykładowych zbiorach dla 50% losowo wybranych punktów zbioru danych

Wnioski płynące z wykonanych badań są takie same jak uzyskane w rozdziale 6.3.2. Stąd w dalszej części pracy badania algorytmów, *kNN-Index-Vp-Tree* stosujących miarę kosinusową jako miarę podobieństwa i działających na danych tekstowych będą wykonywane z użyciem implementacji punktu rzadkiego, a dla uruchamianych na danych gęstych o niewielkim wymiarze będą prezentowane wyniki uzyskane przy zastosowaniu implementacji punktu gęsteego.

6.8. Porównanie algorytmów TI-k-Neighborhood-Index z kNN-Index-Vp-Tree – miara kosinusowa

Na przykładowych zbiorach danych zbadałem algorytmy *kNN-Index-Vp-Tree* oraz *TI-k-Neighborhood-Index* z zastosowaniem miary kosinusowej jako miary podobieństwa. W tab. 42 i na rys. 45 zamieściłem czasy poszukiwań k sąsiadztwa przez oba algorytmy.



Rys. 45. Porównanie wydajności algorytmów kNN-Index-Vp-Tree i TI-k-Neighborhood-Index przy zastosowaniu miary kosinusowej jako miary podobieństwa. Wykresy zawierają czasy wykonania poszukiwań k=5 i k=5 sąsiadztwa sąsiadów w przykładowych zbiorach danych dla 50% losowo wybranych punktów zbioru danych

Tab. 42. Porównanie wydajności algorytmów kNN-Index-Vp-Tree i TI-k-Neighborhood-Index przy zastosowaniu miary kosinusowej jako miary podobieństwa. Tabela zawiera czasy wykonania poszukiwań k=5 sąsiadów i k=5 sąsiedztwa w przykładowych zbiorach dla 50% losowo wybranych punktów zbioru danych

| zbior | l.p. | kNN-Index-Vp-Tree | | | | | TI-k-Neighborhood-Index | | | | |
|----------------|--------|-------------------|---------|---------|-------|----------|-------------------------|-------|--------|-------|-------|
| | | wyk. | wysz. | bud. | norm. | wyk. | wysz. | bud. | obl. | sort. | norm. |
| | | alg. | ind. | | | alg. | ind. | ind. | odl. | | |
| karypis_sport | 1000 | 1,145 | 1,059 | 0,083 | 0,002 | 1,034 | 1,027 | 0,000 | 0,005 | 0,000 | 0,002 |
| | 2000 | 4,252 | 4,084 | 0,164 | 0,004 | 3,519 | 3,510 | 0,000 | 0,005 | 0,001 | 0,004 |
| | 4000 | 18,095 | 17,719 | 0,367 | 0,009 | 14,359 | 14,337 | 0,000 | 0,012 | 0,001 | 0,009 |
| | 6000 | 41,553 | 40,977 | 0,563 | 0,013 | 34,451 | 34,418 | 0,000 | 0,018 | 0,001 | 0,013 |
| | 8000 | 75,349 | 74,594 | 0,737 | 0,018 | 66,656 | 66,607 | 0,000 | 0,029 | 0,003 | 0,017 |
| | 500 | 0,599 | 0,375 | 0,223 | 0,001 | 0,337 | 0,333 | 0,000 | 0,003 | 0,000 | 0,002 |
| karypis_review | 1000 | 1,930 | 1,509 | 0,418 | 0,003 | 1,293 | 1,286 | 0,000 | 0,004 | 0,000 | 0,003 |
| | 2000 | 7,335 | 6,598 | 0,731 | 0,007 | 5,894 | 5,878 | 0,000 | 0,008 | 0,001 | 0,007 |
| | 3000 | 15,511 | 14,728 | 0,774 | 0,009 | 12,591 | 12,569 | 0,000 | 0,011 | 0,001 | 0,010 |
| | 4000 | 26,991 | 26,361 | 0,616 | 0,013 | 21,877 | 21,847 | 0,000 | 0,016 | 0,001 | 0,013 |
| | 10000 | 2,135 | 1,682 | 0,447 | 0,006 | 2,913 | 2,553 | 0,000 | 0,322 | 0,003 | 0,005 |
| | 50000 | 16,985 | 14,367 | 2,585 | 0,033 | 37,549 | 34,851 | 0,001 | 2,934 | 0,016 | 0,025 |
| covtype | 100000 | 53,101 | 33,074 | 19,969 | 0,059 | 141,164 | 134,889 | 0,001 | 6,105 | 0,038 | 0,050 |
| | 300000 | 345,252 | 248,231 | 96,841 | 0,181 | 1128,400 | 1106,590 | 0,003 | 21,494 | 0,147 | 0,150 |
| | 500000 | 989,915 | 812,569 | 177,028 | 0,318 | 3034,090 | 2997,940 | 0,003 | 35,633 | 0,275 | 0,250 |
| | 10000 | 2,637 | 2,214 | 0,417 | 0,006 | 2,868 | 2,414 | 0,000 | 0,447 | 0,003 | 0,005 |
| | 30000 | 25,135 | 22,861 | 2,255 | 0,019 | 9,568 | 8,799 | 0,000 | 0,744 | 0,010 | 0,015 |
| | 50000 | 62,006 | 56,783 | 5,191 | 0,032 | 20,456 | 18,402 | 0,001 | 2,009 | 0,019 | 0,025 |
| cup98 | 70000 | 124,572 | 117,935 | 6,592 | 0,045 | 39,183 | 35,322 | 0,001 | 3,796 | 0,029 | 0,035 |
| | 90000 | 193,587 | 186,308 | 7,220 | 0,058 | 63,047 | 58,520 | 0,001 | 4,440 | 0,039 | 0,046 |

Z rezultatów przeprowadzonych badań wynika, że algorytm *TI-k-Neighborhood-Index* wyszukuje k sąsiedztwo szybciej niż *kNN-Index-Vp-Tree* k sąsiadów. Wyjątek stanowi zbiór covtype, w którego przypadku *kNN-Index-Vp-Tree* wykonuje się szybciej. Z przebiegów wykresów na rys. 45 dla zbiorów różnych od covtype można wnioskować, że algorytm *TI-k-Neighborhood-Index* będzie wykonywał się tym szybciej niż *kNN-Index-Vp-Tree* im większy będzie zbiór, na którym działają te algorytmy.

7. Podsumowanie

W ramach pracy zrealizowałem oprogramowanie implementujące wszystkie przedstawione algorytmy, w tym algorytmy korzystające z nierówności trójkąta jako metody szacowania odległości, zarówno w ujęciu podstawowym jak i z wykorzystaniem indeksu metrycznego VP-Tree. W swoich eksperymentach badałem wydajność algorytmów gęstościowego grupowania w zależności od implementacji algorytmu, zastosowanej miary podobieństwa, przyjętej metody szacowania odległości, a także doboru parametrów charakterystycznych dla danego algorytmu.

W niniejszej pracy eksperymentalnie wykazałem, że wykorzystanie nierówności trójkąta w celu szacowania odległości w procesie gęstościowego grupowania danych, pozwala istotnie ograniczyć liczbę obliczanych rzeczywistych odległości między punktami. Mimo że zastosowanie tej metody obarczone jest kosztem sortowania zbioru punktów (lub indeksu) względem odległości do punktu referencyjnego, wpływa ona na skrócenie czasu pełnego procesu grupowania. Wyniki eksperymentów potwierdzają, że wykorzystanie tej metody pozwala zmniejszyć czas wykonania algorytmu nawet o rzędy wielkości. Warto zwrócić uwagę na to, że różnica wydajności algorytmów stosujących nierówność trójkąta i nie stosujących jej jest tym większa im liczniejszy jest zbiór, na którym wykonywane jest grupowanie.

Otrzymane rezultaty świadczą o tym, że zastosowanie nierówności trójkąta w celu szacowania odległości zwiększa wydajność zarówno wyszukiwania epsilonowego sąsiedztwa, jak i wyszukiwania k sąsiedztwa. W przypadku k sąsiedztwa wartość epsilona nie jest znana przed rozpoczęciem wykonania algorytmu, co sprawia, że nierówność trójkąta nie może być zastosowana w całym procesie wyznaczania sąsiedztwa jak w TI-DBSCAN, a jedynie w procesie weryfikacji potencjalnych k sąsiadów. Stąd, osiągany wzrost wydajności jest mniejszy. Jednakże wciąż stosunkowo wysoki. Jak wykazały rezultaty eksperymentów, wyznaczanie k sąsiadów z wykorzystaniem indeksu metrycznego VP-Tree nie jest wydajniejsze niż użycie pozostałych badanych algorytmów wyznaczania k sąsiedztwa, które stosują nierówność trójkąta w ujęciu podstawowym. Warto odnotować, że zarówno wyznaczanie k sąsiadów przy użyciu indeksu metrycznego VP-Tree jak i wyznaczanie k sąsiedztwa z wykorzystaniem pozostałych badanych algorytmów w równej mierze staje się mniej wydajne

wraz ze wzrostem liczby wymiarów. W przyszłych badaniach warto rozpatrzyć i podać analizie sposoby dostępu do zbioru danych.

Wyniki przeprowadzonych eksperymentów wskazują, że punktami referencyjnymi najbardziej poprawiającymi wydajność są brzegowe punkty zbioru danych – w szczególności punkt minimalny. Wyjątek stanowi sytuacja, w której miara kosinusowa stosowana jest jako miara podobieństwa. Dla takiego przypadku punkt minimalny jest najgorszym z badanych punktów referencyjnych, ponieważ znajduje się blisko środka hipersfery, na powierzchni której leżą znormalizowane punkty zbioru danych. W takiej sytuacji różnice odległości punktów zbioru danych po normalizacji do punktu referencyjnego są na tyle małe, że zastosowanie nierówności trójkąta nie przynosi wzrostu wydajności algorytmu.

W swojej pracy zbadałem również użycie dwóch punktów referencyjnych. Otrzymane rezultaty pokazały, że zastosowanie dodatkowego punktu referencyjnego nieznacznie poprawia wydajność algorytmu. Warto zwrócić uwagę, że dalsze zwiększenie liczby punktów referencyjnych może negatywnie wpływać na wydajność. W swoich badaniach testowałem także zastosowanie rzutowania na wymiar jako metody szacowania odległości. Tylko w jednym przypadku podejście to okazało się równie wydajne co zastosowanie odległości do punktu referencyjnego. Jednakże, wadą metody z rzutowaniem jest silna zależność jej wydajności od kardynalności wymiaru, na który wykonywana jest projekcja.

Istotnym zamiarem przeprowadzonych eksperymentów była analiza wydajności badanych algorytmów w kontekście zbioru danych, na którym działają. W tym celu wybrałem cztery zbiory danych testowych o odmiennych charakterystykach stosowane w badaniach algorytmów eksploracji danych: cup98, covtype, karypis_sport i karypis_review. Pierwsze dwa zbiory są gęste i posiadają mniej niż 60 atrybutów. Resztę zbiorów stanowią zbiory danych tekstowych, których cechami charakterystycznymi jest rzadkość oraz bardzo duża liczba wymiarów.

Dane testowe pozwoliły pokazać, że wydajność stosowanych strategii zależy od charakterystyki zbioru danych. Wyniki eksperymentów z użyciem rzadkich danych wielowymiarowych wykazały, że gdy badania przeprowadzane są na danych tekstowych, to najlepiej stosować implementację punktu rzadkiego, a działając z danymi gęstymi o niewielkiej liczbie atrybutów, najlepiej posłużyć się gęstą implementacją punktu. Rezultaty przeprowadzonych badań wykazały, że stosowanie nierówności trójkąta przynosi mniejsze zyski wydajności, gdy grupowane są dane tekstowe niż gdy przetwarzane są gęste dane o niewielkiej liczbie wymiarów. W testach wyboru punktu referencyjnego, punktów referencyjnych oraz wymiaru rzutowania różnice między otrzymanymi rezultatami

eksperymentów przeprowadzonych na danych tekstowych są tak niewielkie, że na ich podstawie nie można wskazać wartości parametru, dla której algorytm działa najwydajniej. Stąd wniosek, że eksperymenty te warto powtórzyć dla danych tekstowych o większej liczbie punktów, rzędu liczby punktów zbiorów cup98 i covtype, ponieważ w ich przypadku różnice między otrzymanymi rezultatami eksperymentów pozwalają wskazać wartości parametrów, dla których algorytm działa najwydajniej.

Kolejnym celem pracy było zbadanie wpływu zastosowania miary kosinusowej jako miary odległości na wydajność testowanych algorytmów. Należy przypomnieć, że metoda ta dokonuje normalizacji zbioru, tym samym zmieniając jego charakter. Przykładowym efektem takiej zmiany jest drastyczny spadek wydajności grupowania przy użyciu punktu minimalnego jako punktu referencyjnego, gdy miara odległości zmieniana jest z euklidesowej na miarę kosinusową. Porównanie uzyskanych wyników eksperymentów badanych algorytmów wyszukiwania k sąsiedztwa dla obu miar podobieństwa wskazuje, że zastosowanie miary kosinusowej zapewnia wydajność nieznacznie gorszą od wydajności uzyskiwanej, gdy miarą podobieństwa jest miara odległości euklidesowej. Wpływ użycia miary kosinusowej jako miary podobieństwa nie był badany explicite dla algorytmów wyszukujących sąsiedztwo epsilonowe.

W trakcie wykonywania eksperymentów napotkałem problem losowych zakłóceń wyników czasowych. Przeciwność tej próbowałem zwalczać podnosząc priorytet procesu wykonującego algorytm oraz powtarzając dany przypadek testowy wiele razy, a następnie kierować się średnią z uzyskanych rezultatów. Niestety żadne z wymienionych rozwiązań nie wyeliminowały problemu, a jedynie zmniejszyły jego wpływ na otrzymywane rezultaty. Dlatego sądzę, że warto w przyszłych pracach z dziedziny algorytmów grupowania danych rozważyć inne kryteria porównania wydajności algorytmów, np.: liczbę wykonanych obliczeń rzeczywistych odległości.

Podsumowując, otrzymane w pracy dyplomowej rezultaty badań posłużenia się nierównością trójkąta w celu usprawnienia algorytmów gęstościowego grupowania danych świadczą o istotnym zwiększeniu wydajności, w szczególności metoda ta pozostaje sprawna, gdy algorytm operuje na danych tekstowych, które wciąż stanowią wyzwanie dla większości algorytmów gęstościowego grupowania danych.

Bibliografia

- [1] Bentley J. L.: Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 1975, 509-517.
- [2] Blackard J.A.: The Forest CoverType dataset. 1999, Lipiec. [Online]. <http://ftp.ics.uci.edu/pub/machine-learning-databases/covtype/covtype.info>
- [3] Bozkaya T., Ozsoyoglu M.: Distance based indexing for high-dimensional metric spaces. Materiały z *1997 ACM SIGMOD International Conference on Management of Data*, 1997, 357-368.
- [4] Elkan C.: Using the Triangle Inequality to Accelerate k-Means. Materiały z *ICML-2003*, 2003, 147-153.
- [5] Ester M., Kriegel H.P., Sander J., Xu X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Database with Noise. Materiały z *2nd International Conference on Knowledge Discovery and Data Mining KDD'96*, 1996, 226-231.
- [6] Karypis G. :The various datasets used in evaluating the performance of CLUTO's clustering algorithms. 2012, Marzec. [Online]. <http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download>
- [7] Kryszkiewicz M., Lasek P.: A Neighborhood Based Clustering by Means of the Triangle Inequality and Reference Points. Materiały z *IDEAL 2010*, 2010, 284-291.
- [8] Kryszkiewicz M., Lasek P.: TI-DBSCAN: Clustering with DBSCAN by Means of the Triangle Inequality. Materiały z *RSCTC 2010*, 2010, 60-69.
- [9] Kryszkiewicz M., Skonieczny Ł.: Faster Clustering with DBSCAN. Materiały z *IIPWM'05*, 2005, 605-614.
- [10] Kryszkiewicz M.: Cosine Similarity in Terms of Euclidean Distance. Manuskrypt, 2012.
- [11] Kryszkiewicz M.: Determining Cosine Similarity Neighborhoods by Means of the Euclidean Distance. *Rough Sets and Intelligent Systems*, 2013, 323-345.
- [12] Kryszkiewicz M.: The Triangle Inequality versus Projection onto a Dimension in Determining Cosine Similarity Neighborhoods of Non-Negative Vectors, Materiały z *RSCTC*, 2012, 229-236.

- [13] Parsa I.: The UCI KDD Archive: KDD Cup 1988 Data. 1999, Luty. [Online].
<http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html>
- [14] Wawer J.: Gęstościowe grupowanie danych z użyciem nierówności trójkąta. Praca magisterska, Politechnika Warszawska, 2012.
- [15] Yanilos P. N.: Data Structures and Algorithms of Nearest Neighbor Search in General Metric Spaces. Materiały z *4th ACM-SIAM Symposium on Discrete Algorithms*, 1993, 311-321.
- [16] Zhou S., Zhoa Y. Guan J., Huang J.Z.: A Neighborhood-Based Clustering Algorithm. Materiały z *PAKDD*, 2005, 261-371.

A Informacje dla użytkowników oprogramowania

W niniejszym dodatku przedstawiono oprogramowanie zaprojektowane i stworzone do analizy algorytmów badanych w pracy dyplomowej. Oprogramowanie zostało napisane w języku C++, jest przeznaczone dla systemu operacyjnego Microsoft Windows i implementuje wszystkie algorytmy omówione w poprzednich rozdziałach. Stworzona aplikacja wykonywana jest w trybie wsadowym. Jako wejście program przyjmuje listę plików parametrów definiujących wykonanie algorytmów. Wyjście aplikacji stanowią tekstowe pliki raportów z uruchomień algorytmów. Rozdział ten zawiera opisy:

- Struktury plików, z których korzysta implementacja,
- Pliku parametrów uruchomienia aplikacji,
- Pliku parametrów uruchomienia algorytmu,
- Znaczenia poszczególnych czasów wykonania dla uruchomień poszczególnych algorytmów,
- Wykorzystanych zbiorów danych,
- Uruchomienia aplikacji

A.1. Struktura plików

Do pracy dyplomowej załączono plik *implementacja.zip* o następującej strukturze folderów:

- Implementacja
 - algorithms_engine_properties
 - code
 - hdr
 - src
 - datasets
 - logs
 - properties

Zawartość folderów opisano w tab 42.

Tab. 43. Opis zawartości folderów pliku implementacja.zip

| Nazwa folderu | Zawartość folderu |
|-------------------------------------|---|
| implementacja | Foldery algorithms_engine_properties , code , datasets , logs , properties oraz plik wykonywalny implementacja.exe . |
| algorithms_engine_properties | Plik parametrów algorithms_engine_properties.txt . Opis pliku parametrów uruchomienia aplikacji znajduje się w rozdziale ?? |
| code | Foldery hdr i src . |
| hdr | Pliki nagłówkowe *.h aplikacji. |
| src | Pliki źródłowe *.cpp aplikacji. |
| datasets | Pliki ze zbiorami danych *.txt . |
| logs | Pliki raportów wykonania algorytmu *.txt , oraz pliki raportów wykonania aplikacji *.csv . Krótki opis plików raportu znajduje się w rozdziale ?? |
| properties | Pliki parametrów uruchomienia algorytmu. Opis pliku parametrów uruchomienia algorytmu znajduje się w rozdziale ??. |

Plik wykonywalny *implementacja.exe* jest programem implementującym badane w pracy dyplomowej odmiany algorytmów. Program skompilowany jest dla środowiska WIN32 z podniesioną flagą LARGEADDRESSAWARE, która umożliwia aplikacji użycie do 3GB pamięci wirtualnej. Uruchomienia wykonywane są na środowisku Windows 7 x64 z procesorem Intel® i7™ 950 z dostępną pamięcią RAM równą 6GB.

Dane wejściowe programu tworzą 3 rodzaje plików:

- Plik parametrów uruchomienia aplikacji – plik tekstowy zawierający rekordy postaci „nazwa_parametru=wartość_parametru”,
- Pliki parametrów uruchomienia algorytmu – pliki tekstowe zawierające rekordy postaci „nazwa_parametru=wartość_parametru”,
- Pliki danych wejściowych – pliki tekstowe zawierające zbiór wektorów w formacie gęstym lub rzadkim.

Dane wyjściowe programu tworzą 3 rodzaje plików:

- Raport wykonania algorytmu – plik tekstowy, który zawiera:
 - wartości parametrów, z którymi uruchomiono algorytm,
 - czasy wykonania poszczególnych kroków algorytmu,
 - wyniki wykonania algorytmu.
- Raport wykonania aplikacji – plik *.csv, będący raportem zbiorczym zawierającym dane raportów wykonania algorytmu z wyłączeniem wyników wykonania algorytmu. Każdy rekord pliku *.csv odpowiada pewnemu raportowi z uruchomienia algorytmu w ramach uruchomienia aplikacji.
- Oczyszczony raport wykonania aplikacji – plik *.csv, będący uśrednieniem raportu wykonania aplikacji. Każdy rekord pliku *.csv odpowiada serii (ilość powtórzeń) wykonów algorytmu. Uruchomienie algorytmu opisane jest plikiem parametrów uruchomienia algorytmu. Wielkość serii opisana jest parametrem **test_repeats** w pliku parametrów uruchomienia aplikacji. W rekordzie pliku oczyszczonego raportu wykonania aplikacji każda wartość odpowiadająca czasowi wykonania pewnego kroku K algorytmu A w serii S jest tworzona w następujący sposób: spośród wszystkich wartości czasowych wykonania K w algorytmie A w serii S usuwane są jedna maksymalna i jedna minimalna wartość, a z pozostałych wartości liczona jest wartość średnia. Wartość ta stanowi czas wykonania kroku K algorytmu A.

A.2. Plik parametrów uruchomienia aplikacji

W folderze **algorithms_engine_properties** znajduje się plik parametrów uruchomienia aplikacji **algorithms_engine_properties.txt**. W tab. 43. opisano znaczenie poszczególnych parametrów zawartych w tym pliku.

Tab. 44. Opis znaczenia parametrów pliku algorithms_engine_properties.txt

| Nazwa parametru | Opis |
|---------------------|---|
| alfa | Wartość współczynnika, przez który przemnażane są wartości wymiarów znormalizowanych wektorów (Normalizacja wektorów wykonywana jest w fazie przetwarzania danych gdy stosowana jest miara kosinusowa jako miara podobieństwa). |
| test_repeats | Liczba uruchomień algorytmu opisanego plikiem parametrów uruchomienia algorytmu. Parametr ten pozwala na wielokrotne powtórzenie uruchomienia algorytmu. Parametr ten definiuje wielkość serii. |

A.3. Plik parametrów uruchomienia algorytmu

W folderze **properties** znajdują się pliki uruchomienia algorytmów. Każdy plik parametrów uruchomienia algorytmu definiuje wykonanie algorytmu. Program kolejno wczytuje pliki parametrów z folderu **properties** i dla każdego z nich wykonuje zdefiniowany w nim algorytm tyle razy ile specyfikuje parametr **test_repeats** pliku parametrów uruchomienia aplikacji opisany w poprzednim rozdziale. W tab. 44. Opisano poszczególny parametry pliku parametrów uruchomienia algorytmu.

Tab. 45. Opis znaczenia parametrów pliku uruchomienia algorytmu

| Nazwa parametru | Opis |
|-----------------------|---|
| algorithm_name | Nazwa algorytmu przyjmująca wartości: *dbSCAN – algorytm DBSCAN. Wymaga zdefiniowania parametrów: <i>eps</i> , <i>min_pts</i> . *dbSCAN_points_elimination – algorytm DBSCAN z eliminacją rozpatrzonych punktów. Wymaga zdefiniowania parametrów: <i>eps</i> , |

min_pts.

***ti_dbSCAN** – algorytm TI-DBSCAN.

Wymaga zdefiniowania parametrów:

eps,

min_pts,

reference_point,

reference_point_format,

use_dataset_index_access.

***ti_dbSCAN_ref** – algorytm TI-DBSCAN wykorzystujący wiele punktów referencyjnych do szacowania odległości.

Wymaga zdefiniowania parametrów:

eps,

min_pts,

reference_point,

reference_point_format,

use_dataset_index_access.

***ti_dbSCAN_ref_projection** – algorytm TI-DBSCAN wykorzystujący wiele punktów referencyjnych oraz rzuty na osie do szacowania odległości.

Wymaga zdefiniowania parametrów:

eps,

min_pts,

reference_point,

reference_point_format,

use_dataset_index_access,

projection_dimensions,

projection_source_sequence.

***k_neighborhood** – algorytm k-Neighborhood-Index. Sąsiedzi wyznaczani metodą brute-force liczenia odległości na zasadzie ‘każdy z każdym’.

Wymaga zdefiniowania parametrów:

k,

use_dataset_index_access,

classification_subset_factor,

use_binary_placement.

***ti_k_neighborhood** – algorytm TI- k-Neighborhood-Index.

Wymaga zdefiniowania parametrów:

k,
reference_point,
reference_point_format,
use_dataset_index_access,
classification_subset_factor,
use_binary_placement.

***ti_k_neighborhood_ref** – algorytm TI k-Neighborhood-Index-Ref wykorzystujący wiele punktów referencyjnych do szacowania odległości.

Wymaga zdefiniowania parametrów:

k,
reference_point,
reference_point_format,
use_dataset_index_access,
classification_subset_factor,
use_binary_placement.

***ti_k_neighborhood_ref_projection** – algorytm TI k-Neighborhood-Index-Ref wykorzystujący wiele punktów referencyjnych lub rzuty na osie do szacowania odległości.

Wymaga zdefiniowania parametrów:

k,
reference_point,
reference_point_format,
use_dataset_index_access,
classification_subset_factor,
use_binary_placement,
projection_dimensions,
projection_source_sequence.

***vp_tree** – algorytm wyszukujący k najbliższych sąsiadów. Buduje a następnie przeszukuje indeks metryczny VP-Tree.

Wymaga zdefiniowania parametrów:

k,
p_sample_index,
s_sample_index,
classification_subset_factor,

| | |
|--|--|
| | <i>use_binary_placement,</i> <i>search_method,</i> <i>use_boundaries.</i> |
| eps | Promień sąsiedztwa. |
| min_pts | Minimalna liczba punktów w grupie. |
| k | Liczba sąsiadów. |
| use_cosine_similarity | Flaga definiująca rodzaj wyszukiwanego otoczenia przyjmująca wartości: * true – jeśli jako miara podobieństwa ma zostać użyta miara kosinusowa, * false – jeśli jako miara podobieństwa ma zostać użyta miara euklidesowa. |
| dataset_file_format | Flaga definiująca format wczytywanych danych przyjmująca wartości: * dense – jeśli wczytywane wektory zapisane są w formacie gęstym, * sparse – jeśli wczytywane wektory zapisane są w formacie rzadkim. |
| dataset_file_path | Ścieżka dostępu do pliku ze zbiorzem punktów, plik powinien się znajdować w katalogu datasets . |
| dataset_dimension | Liczba definiująca największy wymiar każdego punktu jaki zostanie wczytany do pamięci programu. Wymiary punktu większe niż <i>dataset_dimension</i> nie zostaną wczytane do pamięci programu. |
| dataset_dimension_value_threshold | Współczynnik wyznaczający poziom istotności wartości wymiaru punktu. Jeśli wartość dowolnego wymiaru punktu będzie niższa od <i>dimension_value_threshold</i> to temu wymiarowi zostanie przypisana wartość 0. |
| dataset_elements_number | Liczba punktów zbioru <i>dataset_file_path</i> , które zostaną wczytane do pamięci programu jako zbiór punktów. |
| dataset_internal_format | Flaga definiująca wewnętrzny sposób przechowywania punktów w programie przyjmująca wartości: * dense – punkt przechowywany jest w formacie gęstym, * sparse – punkt przechowywany jest w formacie rzadkim |
| reference_point | Definicja punktów referencyjnych. Punkty referencyjne mogą być definiowane zarówno w formacie gęstym jak i rzadkim. Format według, którego <i>reference_point</i> będzie parsowany przechowywany jest w parametrze <i>reference_point_format</i> . Punkty referencyjne przechowywane są w programie w formacie zgodnym z parametrem <i>internal_representation</i> . |

Funkcje:

[max] – punkt referencyjny o maksymalnych wartościach każdego wymiaru w danym zbiorze danych. ([max, max, ..., max])

[min] – punkt referencyjny o minimalnych wartościach każdego wymiaru w danym zbiorze danych. ([min, min, ..., min])

[max_min] – punkt referencyjny o maksymalnych wartościach wymiarów w danym zbiorze danych dla wymiarów nieparzystych i minimalnych wartościach wymiarów w danym zbiorze danych dla wymiarów parzystych. ([max, min, max, min, max, min, ...])

[n] – punkt referencyjny postaci [n,n,...,n]

[(a,b)*] – punkt referencyjny, tworzony na zasadzie powtarzania wzorca aż do maksymalnego wymiaru. Przykładowo dla:

[(a,b)*] punkt referencyjny byłby postaci [a,b,a,b,a,b,a,b,a,b,...], a dla [(a,b,c,d)*] punkt referencyjny byłby postaci [a,b,c,d,a,b,c,d,a,b,c,d,...].

[rand] – punkt referencyjny o losowych i nie większych niż maksymalne wartościach poszczególnych wymiarów w danym zbiorze danych.

max – maksymalna wartość wymiaru w danym zbiorze danych.

Przykładowe użycie dla przestrzeni dwuwymiarowej:

format gęsty:

[max,1],

[max,max] \Leftrightarrow [max].

format rzadki: (1 odpowiada pierwszemu wymiarowi)

[1,max,2,1],

[1,max,2,max] \Leftrightarrow [max].

min - minimalna wartość wymiaru w danym zbiorze danych.

Przykładowe użycie dla przestrzeni dwuwymiarowej: (analogicznie do max).

reference_point może definiować więcej niż jeden punkt referencyjny, przykładowo:

reference_point = [0][min][max]

reference_point_format

Flaga definiująca format definicji *reference_point* przyjmująca wartości:

***dense** – definicja *reference_point* w formacie gęstym,

***sparse** – definicja *reference_point* w formacie rzadkim.

| | |
|-----------------------------------|---|
| use_dataset_index_access | Flaga definiująca sposób dostępu do zbioru wektorów przyjmująca wartości: *true – dostęp do zbioru wektorów przez indeks. Jeśli wymagane jest sortowanie zbioru to sortowany jest indeks, *false – bezpośredni dostęp do zbioru wektorów lub dostęp do zbioru przez indeks jeśli wymaga tego implementacja (algorytmy DBSCAN z nierównością trójkąta). Jeśli wymagane jest sortowanie zbioru to sortowany jest bezpośrednio zbiór wektorów. |
| projection_dimension | Numery wymiarów oddzielone przecinkami, na które mogą być rzutowane punkty w celu szacowania odległości (d1 odpowiada pierwszemu wymiarowi, brak wymiaru zerowego 0). Funkcje: dmax – wymiar o najszerzej dziedzinie. dmin – wymiar o najwęższej niezerowej dziedzinie. drand – losowy wymiar o niezerowej dziedzinie. |
| projection_source_sequence | Kolejność kryteriów, według których realizowane będzie szacowania odległości. Przykładowo, mając dane: <i>reference_point=[0][max][min]</i> <i>projection_dimensions=1,2</i> <i>projection_source_sequence=d1,r2,r3,r1,d2</i> Zbiór punktów zostanie posortowany względem wartości kryterium. Kryterium dla każdego punktu wyznaczane jest na podstawie <i>projection_source_sequence</i> . Sposób obliczania kryterium: Kryterium stanowi wektor wartości: *(d1) wartość pierwszego wymiaru punktu, *(r2) odległość punktu do drugiego punktu referencyjnego, *(r3) odległość punktu do trzeciego punktu referencyjnego, *(r1) odległość punktu do pierwszego punktu referencyjnego, *(d2) wartość drugiego wymiaru punktu. |

W czasie przycinania kolejnymi kryteriami branymi pod uwagę będą:

- wartość pierwszego wymiaru zdefiniowanego w liście *projection_dimensions* (*d1=1*)
- odległość do drugiego punktu referencyjnego zdefiniowanego w liście *reference_point* (*r2=[max]*),
- odległość do trzeciego punktu referencyjnego zdefiniowanego w liście *reference_point* (*r3=[min]*),
- odległość do pierwszego punktu referencyjnego zdefiniowanego w liście *reference_point* (*r1=[0]*),
- wartość drugiego wymiaru zdefiniowanego w liście *projection_dimensions* (*d2=2*)

| | |
|-------------------------------------|--|
| classification_subset_factor | Definiuje, co który rekord zbioru danych ma zostać wybrany do zbioru, które będzie podlegał klasyfikacji. |
| use_placement | Flaga definiująca sposób wybrania najlepszego odpowiednika punktu klasyfikowanego w zbiorze punktów <i>dataset_file_path</i> przyjmująca wartości: *true – z zastosowaniem wyszukiwania binarnego, *false – z zastosowaniem wyszukiwania liniowego. |
| p_sample_index | Maksymalna liczba elementów zbioru randomSampleP wykorzystywanego do znalezienia vantage point. |
| s_sample_index | Maksymalna liczba elementów zbioru randomSampleD wykorzystywanego do znalezienia vantage point. |
| search_method | Zmienna definiująca metodę poszukiwania sąsiadów w drzewie vantage point. Dostępne metody to: *range – wyszukiwanie zakresowe (wymaga zdefiniowania parametr <i>eps</i>), *k_neighborhood – wyszukiwanie k sąsiadów (wymaga zdefiniowania parametru <i>k_neighborhood</i>). |
| use_boundaries | Flaga definiująca sposób wyszukiwania sąsiadów w drzewie VP przyjmująca wartości: *true – wyszukiwanie z wykorzystaniem wartości sąsiednich mediany (największej wartości mniejszej od mediany i najmniejszej wartości większej od mediany), *false – wyszukiwanie z wykorzystaniem mediany. |

A.4. Wyniki uruchomień programu

Wyniki uruchomień programu znajdują się w folderze logs. Każdemu poprawnemu ruchomieniu algorytmu odpowiada plik o nazwie *run_report%data_i_czas_uruchomienia%.txt*. Raport zawiera:

- wartości parametrów, z którymi uruchomiono program,
- czasy wykonania poszczególnych etapów algorytmów,
- wyniki wykonania algorytmu.

W tab. 45. opisano znaczenie rekordów raportu dotyczących czasów wykonania w zależności od użytego algorytmu. Wartości czasów podawane są w sekundach.

Tab. 46. Opis rekordów raportu wykonania algorytmu dotyczących czasów wykonania

| Algorytm | Nazwa czasu | Opis |
|---------------------------|-----------------------------|--|
| | Dataset read | Czas wczytywania zbioru punktów z pliku. |
| | Reference point calculation | Czas obliczania punktów referencyjnych. |
| DBSCAN | Normalization | Czas normalizacji punktów. |
| | Algorithm execution | Czas wykonania algorytmu. |
| DBSCAN_POINTS_ELIMINATION | Algorithm execution | Czas wykonania algorytmu. |
| TI-DBSCAN | Algorithm execution | Czas wykonania algorytmu. Algorithm execution = Clustering + Distance + Sorting + Building index; |
| | Clustering | Czas wykonania grupowania. |
| | Distance | Czas obliczania odległości punktów zbioru punktów do punktu referencyjnego. |
| | Building index | Tworzenie indeksu dostępu do danych. |
| | Sorting | Czas sortowania zbioru punktów według kryterium odległości do punktu referencyjnego. |
| TI-DBSCAN-REF | Algorithm execution | Czas wykonania algorytmu. Algorithm execution = Clustering + Distance + Sorting + Building index; |

| | | |
|---------------------------------|---------------------|---|
| | Clustering | Czas wykonania grupowania. |
| | Distance | Czas obliczania odległości punktów zbioru punktów do każdego z punktów referencyjnych. |
| | Building index | Tworzenie indeksu dostępu do danych. |
| | Sorting | Czas sortowania zbioru punktów według kryteriów odległości do punktów referencyjnych. |
| TI-DBSCAN-REF-PROJECTION | Algorithm execution | Czas wykonania algorytmu. Algorithm execution = Clustering + Distance + Sorting; |
| | Clustering | Czas wykonania grupowania. |
| | Distance | Czas obliczania odległości punktów zbioru punktów do każdego z punktów referencyjnych oraz wykonania projekcji. |
| | Sorting | Czas sortowania zbioru punktów według kryteriów odległości do punktów referencyjnych oraz projekcji. |
| K-NEIGHBORHOOD | Algorithm execution | Czas wykonania algorytmu. |
| TI-K-NEIGHBORHOOD | Algorithm execution | Czas wykonania algorytmu. Algorithm execution = Clustering + Distance + Sorting + Building index; |
| | Clustering | Czas znajdowania k sąsiadów. |
| | Distance | Czas obliczania odległości punktów zbioru punktów do punktu referencyjnego |
| | Building index | Tworzenie indeksu dostępu do danych. (Wartość zliczana jeśli algorytm wykonywany jest z opcją dostępu do danych przez indeks) |
| | Sorting | Czas sortowania zbioru punktów według kryterium odległości do punktu referencyjnego. |
| | Positioning | Czas pozycjonowania. |
| TI-K-NEIGHBORHOOD-REF | Algorithm execution | Czas wykonania algorytmu. Algorithm execution = Clustering + Distance + Sorting + Building index; |
| | Clustering | Czas znajdowania k sąsiadów. |
| | Distance | Czas obliczania odległości punktów zbioru punktów do każdego z punktów referencyjnych. |

| | | |
|---|---------------------|---|
| | Building index | Tworzenie indeksu dostępu do danych. (Wartość zliczana jeśli algorytm wykonywany jest z opcją dostępu do danych przez indeks) |
| | Sorting | Czas sortowania zbioru punktów według kryteriów odległości do punktów referencyjnych. |
| | Positioning | Czas pozycjonowania . |
| TI-K-NEIGHBORHOOD-REF-PROJECTION | Algorithm execution | Czas wykonania algorytmu. Algorithm execution = Clustering + Distance + Sorting + Building index; |
| | Clustering | Czas znajdowania k sąsiadów. |
| | Distance | Czas obliczania odległości punktów zbioru punktów do każdego z punktów referencyjnych oraz wykonania. |
| | Building index | Tworzenie indeksu dostępu do danych. (Wartość zliczana jeśli algorytm wykonywany jest z opcją dostępu do danych przez indeks) |
| | Sorting | Czas sortowania zbioru punktów według kryteriów odległości do punktów referencyjnych oraz projekcji. |
| | Positioning | Czas pozycjonowania. |
| VP-TREE | Algorithm execution | Czas wykonania algorytmu. Algorithm execution = Clustering + Index building; |
| | Clustering | Czas znajdowania k sąsiadów lub otoczenia eps. |
| | Building index | Czas budowania drzewa vantage point z punktów zbioru punktów. |

Każdemu poprawnemu uruchomieniu aplikacji odpowiada plik raportu o nazwie *ultimate_run_report_%data_i_czas_uruchomienia%.csv*. Raport zawiera czasy wykonania poszczególnych etapów algorytmu oraz wartości parametrów z jakimi uruchomiono algorytm, dla wszystkich uruchomień algorytmów w danym uruchomieniu programu. Raport ten stanowi podsumowanie uruchomienia aplikacji i zawiera najważniejsze dane z uruchomień algorytmów. Plik raportu można otworzyć w arkuszu kalkulacyjnym i dzięki dostępnym narzędziom swobodnie porównywać dane raportów uruchomień algorytmów.

Każdemu poprawnemu uruchomieniu aplikacji odpowiada plik oczyszczonego raportu o nazwie *clean_ultimate_run_report_%data_i_czas_uruchomienia%.csv* zawierający uśrednione rezultaty czasów wykonania poszczególnych kroków algorytmu w serii.

A.5. Zbiory danych

W folderze **datasets** znajdują się pliki zawierające zbiory wektorów. W tab. 46. przedstawiono statystyki związane ze zbiorami danych, którymi posłużyono się w pracy dyplomowej. Pliki zostały nazwane według konwencji: *formatPliku_dLiczbaWymiarów_rLiczbaRekordów_NazwaZbioru.txt*.

Tab. 47. Opis zbiorów wektorów wykorzystanych w pracy dyplomowej

| Nazwa zbioru | Liczba wektorów | Liczba wymiarów |
|---|-----------------|-----------------|
| zbioru | | |
| dense_d56_r96367_cup98.txt | 96367 | 56 |
| dense_d55_r581012_covtype.txt | 581012 | 55 |
| sparse_d126373_r8580_karypis_sport.txt | 8580 | 126373 |
| sparse_d126373_r4069_karypis_reviews.txt | 4069 | 126373 |

A.6. Uruchomienie aplikacji

Aby program poprawnie zapisywał i odczytywał pliki należy je umieścić w następujący sposób. Obok pliku **implementacja.exe**, na tym samym poziomie drzewa katalogów, powinny znajdować się następujące katalogi:

- **datasets** – zawierający pliki zbiorów danych (ścieżka względem pliku **implementacja.exe \datasets**),
Żadna linia pliku zbioru danych nie powinna rozpoczynać się białym znakiem. Kolejne wartości opisujące punkt powinny być oddzielone dokładnie jedną spacją.
- **logs** – w nim zapisywane będą raporty wykonania algorytmów (ścieżka względem pliku **implementacja.exe \logs**),
- **properties** – zawierający pliki parametrów uruchomienia algorytmu (ścieżka względem pliku **implementacja.exe \properties**),

Żadna linia pliku parametrów uruchomienia algorytmu nie powinna rozpoczynać się białym znakiem. W każdym pliku parametrów po nazwie parametru powinien znajdować się dokładnie jeden znak ‘=’, a następnie wartość parametru.

- **algorithms_engine_properties** – zawierający plik parametrów uruchomienia aplikacji *algorithms_engine_properties.txt* (ścieżka względem pliku implementacja.exe \algorithms_engine_properties\)

Żadna linia pliku parametrów uruchomienia aplikacji nie powinna rozpoczynać się białym znakiem. W każdym pliku parametrów po nazwie parametru powinien znajdować się dokładnie jeden znak ‘=’, a następnie wartość parametru.

Jeżeli pliki parametrów nie będą znajdowały się w odpowiednim katalogu (\properties\), to program będzie działał niepoprawnie. Jeżeli katalog **logs** nie zostanie utworzony, to pliki raportów nie zostaną zapisane. Istnienie katalogu **datasets** nie jest wymagane, ponieważ w pliku parametrów podawana jest ścieżka do pliku ze zbiorom danych. Jednakże dla spójności rozwiązania wszystkie pliki ze zbiorami danych umieszczane są w folderze **datasets**.

Aby uruchomić implementację należy:

- dokonać odpowiednich zmian w plikach parametrów,
- uruchomić plik **implementacja.exe**.

Po poprawnym zakończeniu uruchomienia aplikacji w folderze logs będą znajdować się pliki raportów z uruchomień algorytmów. W celu zapoznania się z wynikami uruchomienia algorytmu należy przejrzeć jego raport wykonania.