

Rok akademicki 2011/2012

Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki



PRACA DYPLOMOWA MAGISTERSKA

inż. Jarosław Wawer

Gęstościowe grupowanie danych
z użyciem nierówności trójkąta

Opiekun pracy
prof. nzw. dr hab. Marzena Kryszkiewicz

Ocena:

.....
Podpis Przewodniczącego

Chciałbym w tym miejscu serdecznie podziękować Pani Profesor Marzenie Kryszkiewicz za inspirację, cenne porady, okazywaną cierpliwość i pomoc w trakcie przygotowywania pracy dyplomowej.

Dziękuję przy tym Rodzicom, Magdzie i Przyjaciółom za wsparcie w dążeniu do celu.



Specjalność: Inżynieria Systemów Informatycznych

Data urodzenia: 1986.01.02

Data rozpoczęcia studiów: 2010.09.30

Życiorys

Urodziłem się 2 stycznia 1986 roku w Wołominie, gdzie rozpoczęłem swoją edukację. Po ukończeniu Sportowej Szkoły Podstawowej nr 5 im. Cypriana Kamila Norwida i Sportowego Gimnazjum nr 5 im. Polskich Olimpijczyków, kontynuowałem naukę w klasie matematyczno-informatycznej w I Liceum Ogólnokształcącym im. Wacława Nałkowskiego w Wołominie. Po egzaminie dojrzałości, w lutym 2006 roku podjąłem studia dzienne na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej na kierunku Informatyka. W trakcie otrzymywałem stypendium naukowe za dobre wyniki, a także działałem w organizacji studenckiej BEST oraz Kole Naukowym Zarządzania Projektami PMArt. Po zdobyciu tytułu inżyniera, w lutym 2010 kontynuowałem naukę na studiach uzupełniających magisterskich, na specjalności Inżynieria Systemów Informatycznych. Część z nich odbyła się na University of Southampton w Wielkiej Brytanii, w ramach programu wymiany międzynarodowej Erasmus. Podczas testów na Politechnice Warszawskiej zostałem zakwalifikowany do stowarzyszenia Mensa Polska.

.....
Podpis studenta

EGZAMIN DYPLOMOWY

Złożył egzamin dyplomowy w dniu 20__ r

z wynikiem

Ogólny wynik studiów:

Dodatkowe wnioski i uwagi Komisji:

.....
.....

STRESZCZENIE

Praca magisterska rozpoczyna się od krótkiego wprowadzenia w dziedzinę gęstościowego grupowania danych, w tym przedstawienia trzech wybranych algorytmów: DBSCAN, NBC i PreDeCon. W dalszej części, po opisaniu teoretycznych podstaw wykorzystania nierówności trójkąta w gęstościowym grupowaniu danych, przedstawione zostały oparte na tej własności algorytmy TI-DBSCAN i TI-NBC oraz nowy algorytm TI-PreDeCon. Opis ich parametrów skoncentrowany jest w szczególności na możliwych strategiach wyboru punktu referencyjnego, wykorzystywanego do efektywnego grupowania danych z zastosowaniem nierówności trójkąta. Uwzględnione zostały także wersje używające wielu punktów referencyjnych oraz bardziej złożone strategie ich wyboru.

Po opisie najistotniejszych aspektów implementacji, testowych zbiorów danych, jak i testów, znalazła się najważniejsza część pracy, czyli omówienie wyników przeprowadzonych przeze mnie badań eksperymentalnych. Na testowych zbiorach danych, o różnej liczbie wymiarów i charakterystyce, przeanalizowany został wpływ liczby i strategii wyboru punktów referencyjnych na wydajność algorytmów wykorzystujących nierówność trójkąta do wyszukiwania epsilonowego sąsiedztwa lub k-sąsiedztwa.

Wnioski w podsumowaniu obejmują porównanie wydajności algorytmów opartych na nierówności trójkąta z wersjami wykorzystującymi indeksy przestrzenne, takie jak R*-drzewo czy VA-File. Poza podsumowaniem mocnych stron algorytmów wykorzystujących nierówność trójkąta, wskazane zostały także możliwości dalszego rozwoju.

Słowa kluczowe: gęstościowe grupowanie danych, DBSCAN, NBC, PreDeCon, nierówność trójkąta, TI-DBSCAN, TI-NBC, TI-PreDeCon, R*-tree, VA-File

DENSITY-BASED CLUSTERING BY MEANS OF THE TRIANGLE INEQUALITY

The Master's Thesis begins with a brief introduction to a density-based clustering domain, including the description of three selected algorithms: DBSCAN, NBC and PreDeCon. Following the explanation of accelerating density-based clustering by means of the Triangle Inequality, further chapters concentrate on the algorithms using this property, TI-DBSCAN and TI-NBC, and new algorithm TI-PreDeCon. The description of these algorithms' parameters mainly focuses on possible strategies for selecting a reference point, used for efficient clustering by means of the Triangle Inequality. Furthermore, variants with many reference points and more complex strategies for their selection were taken into account.

The main part of the dissertation – the results of my research – follows the description of the most crucial implementation issues, data sets used in the experiments and test scenarios. The use of diversified data sets allowed analyzing the influence of number and strategy of reference points' selection on efficiency of searching ϵ -neighborhood or k-neighborhood.

In the concluding part of the thesis, the original algorithms that use spatial indices (such as R*-tree or VA-File) are compared with the versions using the Triangle Inequality in favour of the latter. Apart from pointing out strengths of the algorithms using the Triangle Inequality, the conclusions also provide the list of issues for further consideration.

Keywords: density-based clustering, DBSCAN, NBC, PreDeCon, Triangle Inequality, TI-DBSCAN, TI-NBC, TI-PreDeCon, R*-tree, VA-File

SPIS TREŚCI

1. Wprowadzenie	7
1.1. Przegląd literatury.....	8
1.2. Motywacja i cel pracy.....	9
1.3. Układ pracy.....	9
2. Wybrane algorytmy gęstościowego grupowania danych	11
2.1. DBSCAN.....	12
2.2. NBC.....	16
2.3. PreDeCon.....	23
3. Wykorzystanie nierówności trójkąta.....	29
3.1. Parametryzacja rozwiązania.....	34
3.1.1. Strategia wyboru jednego punktu referencyjnego	34
3.1.2. Strategia wyboru wielu punktów referencyjnych	35
3.2. Zastosowanie nierówności trójkąta w algorytmach gęstościowego grupowania danych	38
3.2.1. Algorytm TI-DBSCAN	38
3.2.2. Algorytm TI-NBC	44
3.2.3. Algorytm TI-PreDeCon	47
4. Szczegóły implementacji	49
4.1. Założenia i ograniczenia	49
4.2. Implementacja indeksów.....	50
4.2.1. Implementacja R*-drzewa.....	50
4.2.2. Implementacja VA-File.....	51
4.3. Implementacja zoptymalizowanych algorytmów	52
4.4. Implementacja interfejsu użytkownika	53
5. Badania eksperymentalne	55
5.1. Dane testowe.....	55

SPIS TREŚCI

5.2. Parametry testowe	57
5.3. Testy algorytmu DBSCAN	58
5.3.1. Implementacja TI-DBSCAN	58
5.3.2. Strategie wyboru punktów referencyjnych	61
5.3.3. Ocena wydajności	74
5.4. Testy algorytmu NBC	80
5.4.1. Implementacja TI-dNBC	80
5.4.2. Strategie wyboru punktów referencyjnych	81
5.4.3. Ocena wydajności	88
5.5. Testy algorytmu PreDeCon	91
6. Podsumowanie	95
Bibliografia	99
Dodatek A. Opis programu	103

1. WPROWADZENIE

Ilość danych generowanych i gromadzonych we współczesnych systemach komputerowych rośnie w niewyobrażalnym jeszcze kilka lat temu tempie. O ile ich gromadzenie i przechowywanie na różnego rodzaju nośnikach pamięci masowej nie stanowi dużego problemu, o tyle operowanie na takiej ilości danych, mimo ciągłego wzrostu mocy obliczeniowej komputerów, wciąż stanowi duże wyzwanie dla współczesnej informatyki. Nawet największe zbiory danych nie stanowią prawie żadnej wartości same w sobie, ale racjonalnie wykorzystane mogą być nieocenionym źródłem często unikalnej wiedzy. Właśnie jej odkrywaniem zajmuje się coraz popularniejsza dziedzina informatyki, zwana eksploracją danych (ang. „data mining”). Eksploracja danych jest jednym z etapów zaawansowanego procesu odkrywania wiedzy, na który składają się także operacje przygotowujące, jak integracja, selekcja, czyszczenie i transformacja danych oraz niezbędna na koniec interpretacja i analiza wyników.

Wśród najbardziej popularnych metod eksploracji danych wyróżnia się odkrywanie asocjacji i sekwencji, klasyfikację oraz grupowanie. Wszystkie odkrywają różnego rodzaju zależności lub korelacje pomiędzy danymi, znacznie różniąc się przy tym formą otrzymywanych wyników, a co za tym idzie zastosowaniem. Niniejsza praca dyplomowa skupia się właśnie na zagadnieniu grupowania danych (ang. *clustering*). Nie można mylić go jednak z innym zagadnieniem eksploracji danych - klasyfikacją, czyli przydzielaniem obiektów do zdefiniowanych wcześniej na podstawie danych treningowych kategorii. Grupowanie danych definiowane jest jako wyznaczanie zbiorów obiektów „podobnych” lub podziale zbioru wejściowego na mniejsze grupy przy zachowaniu właściwości maksymalizacji podobieństwa obiektów należących do tej samej grupy i minimalizacji podobieństwa obiektów należących do innych grup.

1.1. PRZEGŁĄD LITERATURY

Tematyka grupowania danych, z racji swojej popularności i szerokiego zakresu dziedzin, z którymi się łączy, została szeroko opisana w literaturze. W przypadku algorytmów wykorzystujących metody gęstościowe, na których skupiłem się w niniejszej pracy, szczególnie użyteczne okazują się artykuły naukowe.

Jednym z najpopularniejszych algorytmów gęstościowego grupowania danych jest DBSCAN [7], który często występuje jako punkt odniesienia w przypadku porównań z innymi algorytmami z tej kategorii. Doczekał się on nie tylko wielu publikacji, ale także dużej liczby modyfikacji. Algorytm NBC [23], mimo iż mniej popularny w publikacjach, okazał się ciekawy z racji wykorzystania innego niż DBSCAN warunku gęstości, opartego na gęstości lokalnej, a nie globalnej. Na grupowaniu danych wielowymiarowych z wykorzystaniem podprzestrzeni skupia się natomiast artykuł wprowadzający algorytm PreDeCon [5]. Niezbędnym uzupełnieniem powyższych publikacji są artykuły na temat wykorzystywanych w nich indeksach przestrzennych: VA-File [21] czy R*-tree [3].

Nowym pomysłem na zwiększenie wydajności powyższych algorytmów jest wykorzystanie nierówności trójkąta do ograniczenia liczby kosztownych operacji wyznaczenia podobieństwa dwóch obiektów. Próby wykorzystania nierówności trójkąta w algorytmach grupowania danych przedstawiane były na przykładzie algorytmu k-środków [6] [16]. Jednak po raz pierwszy, posłużyła ona do porządkowania dostępu do danych w algorytmach gęstościowego grupowania TI-DBSCAN [14] i TI-NBC [13]. Pomimo osiągnięcia w powyższych algorytmach bardzo dobrych rezultatów, brak jest w literaturze śladów wykorzystania nierówności trójkąta do zwiększenia wydajności innych algorytmów gęstościowego grupowania danych, takich jak na przykład wspomniany PreDeCon. Nie znalazłem w literaturze także analizy, jaki wpływ na efektywność algorytmów mają przyjęte parametry, takie jak liczba punktów referencyjnych czy strategia ich wyboru.

1.2. MOTYWACJA I CEL PRACY

Grupowanie danych to proces powszechnie wykorzystywany w rozpoznawaniu obrazów, organizacji dokumentów i stron internetowych czy też segmentacji klientów w marketingu. Co więcej, zdolność grupowania podobnych elementów wymieniana jest jako jeden z kluczowych elementów inteligencji, na której w dużym stopniu bazuje także szeroko rozumiana sztuczna inteligencja. Coraz szersze zastosowanie algorytmów grupowania danych we współczesnym świecie sprawia, że budzą one szczególnie duże zainteresowanie badaczy i naukowców, którzy opracowują coraz nowsze algorytmy lub modyfikują te już istniejące, które wydawały się do tego czasu najwydajniejsze. Często pozwalały one wielokrotnie zwiększyć wydajność dotychczasowych rozwiązań, co pozwala na przetwarzanie zbiorów danych z coraz większą liczbą obiektów lub atrybutów. Może oznaczać to możliwość użycia ich w zupełnie nowych, niedostępnych wcześniej zastosowaniach.

Wykorzystanie nierówności trójkąta jest jednym z najnowszych pomysłów na zwiększenie wydajności algorytmów gęstościowego grupowania danych. Jak wspomniałem w przeglądzie literatury, pierwsze testy [13], [14] przynosiły bardzo pozytywne rezultaty, co motywuje do dalszych badań.

Celem pracy jest bardziej rozbudowana analiza algorytmów TI-DBSCAN i TI-NBC, w szczególności wpływu strategii wyboru punktów referencyjnych i ich liczby oraz wyboru implementacji opartej na sortowaniu lub indeksie na wydajność powyższych algorytmów w zależności od charakterystyki zbioru danych. Wyciągnięte wnioski pozwolić mają na wykorzystanie nierówności trójkąta do zwiększenia wydajności kolejnego, bardziej złożonego algorytmu gęstościowego grupowania danych PreDeCon.

1.3. UKŁAD PRACY

W kolejnym rozdziale, po wprowadzaniu w tematykę algorytmów grupowania danych, szczegółowo przedstawiłem trzy wybrane algorytmy, na których skupiona została praca: DBSCAN, NBC i PreDeCon. Opisy stosowanych w nich taksonomii

1. WPROWADZENIE

i charakterystycznych cech uzupełnione są o pseudokody, do których odnoszę się w rozdziale trzecim. Pozwala to wygodnie i precyzyjnie przedstawić modyfikacje, jakie wprowadzane są w opisywanych algorytmach gęstościowego grupowania danych w związku z wykorzystaniem nierówności trójkąta. Wcześniej, na początku rozdziału, przedstawiłem teoretyczne podstawy wprowadzanych modyfikacji oraz opisałem możliwe parametryzacje rozwiązania, w tym znaczenie poszczególnych strategii wyboru punktów referencyjnych.

Rozdziały 4 i 5 skupiają się na praktycznej stronie pracy. W rozdziale czwartym przedstawiłem najistotniejsze szczegóły implementacji, w szczególności założenia i ograniczenia. Natomiast rozdział piąty opisuje badania eksperymentalne. Po opisie wykorzystanych w testach danych i parametrów, przedstawione zostały wyniki badań wszystkich algorytmów. Najbardziej rozbudowana analiza dotyczy w szczególności wyników testów TI-DBSCAN i TI-NBC, które posłużyły do formułowania wniosków na temat wykorzystania nierówności trójkąta do znajdowania odpowiednio: epsilonowego sąsiedztwa i k-sąsiedztwa. Pracę zamyka podsumowanie, w którym znalazły się ostateczne konkluzje oraz spis wykorzystywanej literatury.

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. WYBRANE ALGORYTMY GĘSTOŚCIOWEGO GRUPOWANIA DANYCH

W literaturze [9] można spotkać wiele sposobów rozwiązywania problemu grupowania danych, czyli wykrywania grup obiektów, które zgodnie z przyjętą miarą są do siebie podobne, a przy tym różnią się od obiektów z innych grup. Popularnym przykładem takiej miary jest odległość Euklidesowa, która jako podobne klasyfikuje obiekty leżące blisko siebie. Większość algorytmów jest jednak niezależna od przyjętej funkcji odległości. Zróżnicowanie problemu, wynikające z dużej liczby zastosowań o odmiennych wymaganiach co do rezultatu oraz często nieporównywalnych zbiorów wejściowych o odmiennych charakterystykach (np. różnej liczbie atrybutów oraz rozkładzie ich wartości), jest przyczyną powstania dużej liczby algorytmów. Zazwyczaj osiągają one najlepsze rezultaty w przypadku konkretnych typów zbiorów danych, podczas gdy dla innych zbiorów wyniki bywają zdecydowanie gorsze. Każdy z nich ma pewne wady i zalety, a do tej pory nie został przedstawiony żaden algorytm, który sprawdzałby się we wszystkich scenariuszach. Co więcej, bezpośrednie porównania utrudnia fakt, że cel grupowania określony został na ogólnym poziomie, a szczegóły zależą także od konkretnego zastosowania, dlatego algorytmy często różnią się nie tylko sposobem grupowania danych, ale także samą definicją grupy.

Stosowanych jest wiele klasifikacji algorytmów, rozróżniających je ze względu na metody grupowania czy charakterystyki otrzymywanych wyników [12], ale najbardziej podstawowe i znane metody obejmują algorytmy hierarchiczne oraz algorytmy oparte na podziale. Wynikiem tych pierwszych są dendrogramy¹, których liście to pojedyncze obiekty, a węzły środkowe na wyższych poziomach reprezentują coraz obszerniejsze grupy zawierające w sobie podgrupy. Powstają one w wyniku rekurencyjnego dzielenia grup zaczynając od całego zbioru danych lub przeciwnego procesu rekurencyjnego łączenia grup, począwszy od pojedynczych obiektów. Wymienione

¹ Dendrogram to diagram w kształcie drzewa stosowany często do prezentacji związków pomiędzy wybranymi elementami lub grupami elementów.

2. WYBRANE ALGORYTMY GĘSTOŚCIOWEGO GRUPOWANIA DANYCH

powyżej jako drugie, metody oparte na podziale polegają na znalezieniu najlepszego podziału zbioru na zadaną z góry liczbę rozłącznych, jak najbardziej jednorodnych grup. Początkowy podział, wybrany na przykład na podstawie średnich czy median iteracyjnie, zgodnie z konkretną strategią, optymalizowany jest zgodnie z funkcją celu. Obydwie metody posiadają pewne ograniczenia i wady. W przypadku algorytmów opartych na podziale jest to przede wszystkim konieczność określenia z góry liczby grup, a także ograniczenie kształtów znajdowanych grup, które reprezentowane są zazwyczaj przez ich środki. Natomiast głównym problemem w przypadku metod hierarchicznych jest manualny wybór podziału, na tyle małego, by uwzględniał wszystkie grupy i na tyle dużego, by nie dzielił ich sztucznie i niepotrzebnie na mniejsze części.

Wyniki powyższych metod nie zawsze odpowiadają oczekiwaniom, co tłumaczyć można odmiennym od stosowanego przez ludzi mechanizmem grupowania. Człowiek grupując punkty w dwuwymiarowej przestrzeni nie dzieli zbioru hierarchicznie na coraz mniejsze zbiory, ani nie próbuje podzielić go na z góry ustaloną liczbę części. Grupuje on punkty w zbiory o dowolnym kształcie, a warunkiem przydziela punktów do grup jest zazwyczaj odległość pomiędzy punktami. Mówiąc dokładniej, za „naturalne” grupy uznawane są zbiory punktów leżące w obszarze o gęstości wyraźnie większej niż w obszarze otaczającym je. Najbliżej takiej definicji grupy i metody grupowania są algorytmy gęstościowe. To właśnie na nich skupiłem się w niniejszej pracy, w kolejnych podrozdziałach opisując trzy charakterystyczne algorytmy DBSCAN, NBC i PreDeCon.

2.1. DBSCAN

Algorytm Density-Based Spatial Clustering of Applications with Noise (DBSCAN) to jeden z najpopularniejszych algorytmów gęstościowego grupowania danych. Mimo, iż został on zaproponowany przez M. Ester i innych [7] ponad 15 lat temu, wciąż jest flagowym algorytmem stanowiącym punkt odniesienia w wielu pracach naukowych dotyczących tej tematyki oraz proponujących nowe algorytmy lub rozwiązania.

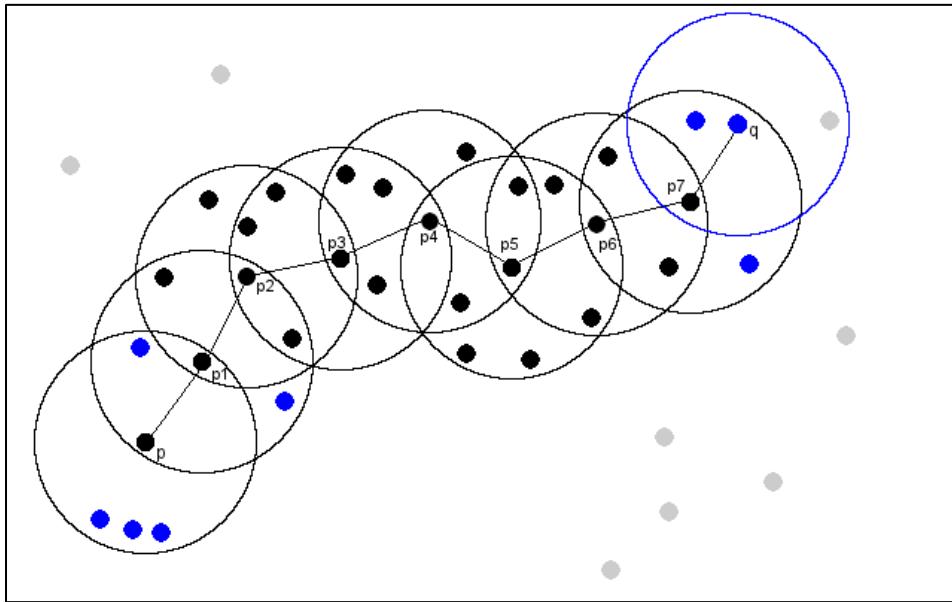
Każda wykrywana przez DBSCAN grupa, to w uproszczeniu zbiór punktów z obszaru o dużej gęstości punktów. Przeciwieństwem są punkty leżące w obszarze o małej gęstości, które uznawane są za szum. Jedną z głównych zalet algorytmu, poza wydajnością, jest operowanie tylko na dwóch ścisłe ze sobą powiązanych parametrach wejściowych, dobrze charakteryzujących wymagania dotyczące minimalnej grupy, która może być obiektem zainteresowania. Jest to epsilon (ε) określający promień wokół punktu, w zakresie którego musi znaleźć się zadana minimalna liczba (μ) sąsiadujących punktów, by były one uznane za grupę. Jest to tym samym ograniczenie na minimalną liczność wykrywanych grup.

Podstawowym pojęciem używanym w kontekście algorytmu DBSCAN jest *epsilonowe otoczenie* (oznaczane jako $N_\varepsilon(p)$) punktu p zawierające wszystkie punkty ze zbioru D oddalone od punktu p według przyjętej funkcji odległości $dist(p,q)$ o nie więcej niż ε , co można zdefiniować jako:

$$(2.1) \quad N_\varepsilon(p) = \{q \in D / dist(p,q) \leq \varepsilon\}$$

Jeżeli tak obliczone epsilonowe otoczenie punktu p zawiera wymaganą minimalną liczbę punktów ($|N_\varepsilon(p)| \geq \mu$), punkt p nazywany jest *punktem rdzeniowym* (ang. *core point*), a punkty należące do jego otoczenia epsilonowego określamy jako *bezpośrednio gęstościowo osiągalne* z punktu p dla danych parametrów μ i ε . Pojęcia te wykorzystane są w definicji gęstościowej osiągalności, która opisuje punkt q jako *gęstościowo osiągalny* z punktu p , jeżeli pomiędzy punktami p i q można znaleźć dowolną sekwencję takich punktów p_0, \dots, p_n , że $p_0=p$ i $p_n=q$, a każdy kolejny punkt p_{i+1} jest bezpośrednio gęstościowo osiągalny z punktu go poprzedzającego p_i , gdzie $i=0 \dots n-1$. Warto zwrócić uwagę, że tak zdefiniowana gęstościowa osiągalność jest relacją niesymetryczną, a więc gęstościowa osiągalność punktu q z punktu p nie oznacza, że istnieje taka sekwencja punktów bezpośrednio gęstościowo osiągalnych w przeciwnym kierunku. Z tego powodu, dodatkowo wprowadzona została symetryczna relacja gęstościowego połączenia. Zgodnie z nią punkty p i q są *punktami gęstościowo połączonymi*, jeżeli istnieje dowolny punkt o , z którego obydwa są gęstościowo osiągalne. Wyjaśnieniem może być tu przykład z rysunku 2.1, na którym dla ułatwienia zaznaczone zostały okręgi o promieniu ε ograniczające epsilonowe otoczenie wybranej grupy.

nych punktów. W przypadku przyjęcia parametru μ równego 5, punkt q jest gęstościowo osiągalny z punktu p (poprzez sekwencję punktów $p1 \dots p7$). Natomiast mimo tego, że p nie jest gęstościowo osiągalny z q (nie jest to punkt rdzeniowy, dlatego żadne punkty nie są z niego bezpośrednio gęstościowo osiągalne), to są one gęstościowo połączone np. poprzez punkt $p7$.



Rysunek 2.1 Przykład punktów q i p gęstościowo połączonych (dla $\mu=5$), np. poprzez punkt $p4$, z którego są gęstościowo osiągalne. Kolorem czarnym zaznaczone zostały punkty rdzeniowe, szarym szum, a niebieskim punkty brzegowe wykrytej grupy.

W takim kontekście, *grupa* zdefiniowana została jako maksymalny niepusty podzbiór wszystkich punktów zbioru D , które są ze sobą gęstościowo połączone. Pozostałe punkty zbioru D , które nie są bezpośrednio gęstościowo osiągalne z punktów rdzeniowych, przez co nie należą do żadnych grup, nazywane są *szumem*. Trzecim charakterystycznym rodzajem punktów są *punkty brzegowe*, które należą do grup, mimo iż nie są punktami rdzeniowymi, a jedynie leżą w ich epsilonowym otoczeniu.

Podstawowa wersja algorytmu DBSCAN iteruje wejściowy zbiór punktów i dla każdego punktu, który nie został jeszcze sklasyfikowany (tzn. nie został przypisany do grupy, ani zidentyfikowany jako szum) uruchamia procedurę tworzenia nowej grupy *expandCluster*. Procedura *expandCluster* rozpoczyna się od wyznaczenia epsilonowego otoczenia danego punktu i oznaczenia go, jako szum, jeśli nie jest to punkt

2. WYBRANE ALGORYTMY GĘSTOŚCIOWEGO GRUPOWANIA DANYCH

rdzeniowy lub budowy nowej grupy w przeciwnym przypadku. Pierwszymi punktami należącymi do tworzonej grupy są wszystkie punkty z wyznaczonego epsilonowego otoczenia badanego punktu (w tym on sam). Wszystkie punkty z epsilonowego sąsiedztwa² dodawane są następnie do tzw. *zbioru ziaren*, zawierającego punkty, które mają szanse rozszerzyć grupę o dodatkowe punkty. Algorytm iteruje zbiór ziaren, dla każdego punktu wyznaczając jego epsilonowe otoczenie. Jeśli jest to punkt rdzeniowy, wszystkie nienależące jeszcze do żadnej grupy punkty z wyznaczonego epsilonowego sąsiedztwa także dodawane są do budowanej grupy, a te, które nie były jeszcze oznaczone jako szum, dodatkowo dodawane są na koniec zbioru ziaren, jako potencjalne punkty rdzeniowe, które mają szanse dalej rozszerzać tworzoną grupę. Opisany algorytm można zapisać w postaci poniższego pseudokodu:

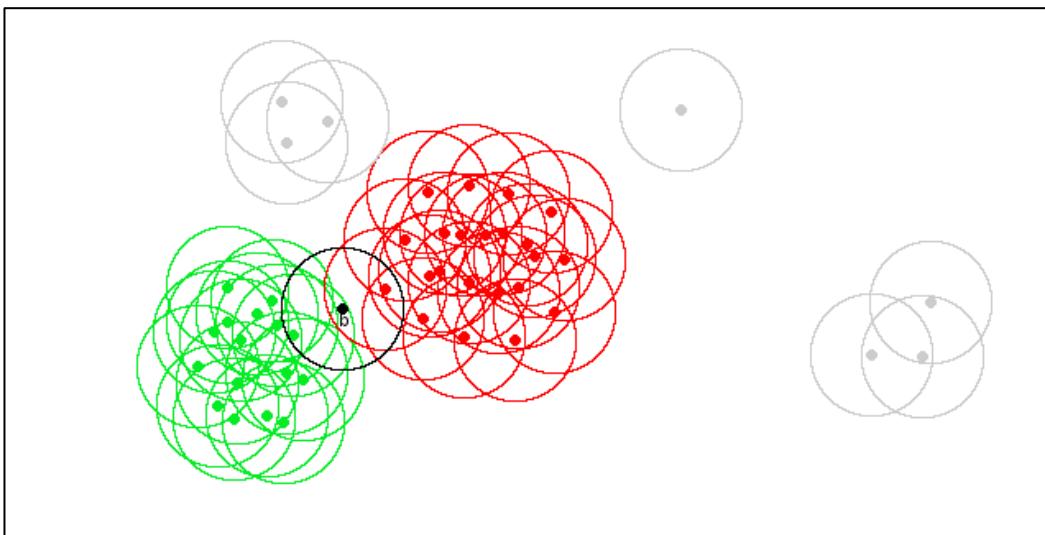
```
Algorithm DBSCAN (D, ε, μ) :
    preparation(D);
    generate new clusterID;
    for each unclassified p ∈ D
        if expandCluster(clusterID, p, D, ε, μ) then
            generate new clusterID;
    return D;
end.

Function preparation (D) :
    mark all p ∈ D as unclassified;
end.

Function expandCluster (clusterID, p, D, ε, μ) : Boolean
    insert all q ∈ Neps(p) into queue Φ;
    if |Φ| < μ then
        mark p as noise;
        return false;
    else
        assign current clusterID to all q ∈ Φ;
        delete p from Φ;
        while Φ ≠ ∅ do
            q = first point in Φ;
            if |Neps(q)| ≥ μ then
                for each x ∈ Neps(q) do
                    if x is unclassified or noise then
                        if x is unclassified then add x to Φ;
                        assign current clusterID to x;
                remove q from Φ;
            return true;
    end.
```

2 Zwrócić należy uwagę na różnicę pomiędzy stosowanymi pojęciami sąsiedztwa i otoczenia punktu, które mimo iż w anglojęzycznej literaturze zazwyczaj tłumaczone są na jedno pojęcie *neighbourhood*, w języku polskim są definiowane w różny sposób. Sąsiedztwo jest pojęciem węższym, określającym otoczenie punktu bez niego samego.

Dokładniejsza analiza pseudokodu pozwala zauważyć pewną nieścisłość. Powyższy algorytm jest deterministyczny, ale tylko z dokładnością do punktów brzegowych. Nie uwzględnia on, że mogą one należeć do więcej niż jednej grupy, jeśli znajdują się pomiędzy położonymi blisko siebie grupami, tak jak na przykładzie, przedstawionym na rysunku 2.2. Wynik działania powyższego algorytmu może w pewnym stopniu zależeć od kolejności przeglądania punktów, ponieważ punkt brzegowy przypisany już do jakieś grupy może, w wyniku rozbudowy kolejnych grup, zostać przypisany także do nich. Problem ten można prosto rozwiązać modyfikując atrybut *clusterId* punktu, tak by przechowywał zbiór identyfikatorów grup, a nie jeden identyfikator. Tak jak autorzy algorytmu DBSCAN, ten problem uznajemy jednak za pomijalny.



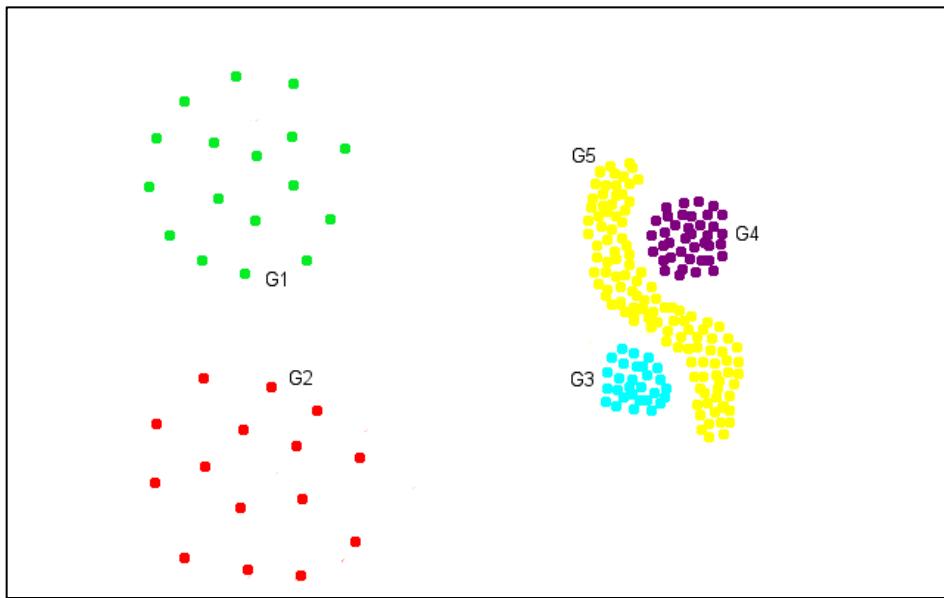
Rysunek 2.2 Przykład ilustrujący przypadek punktu *b*, który jest punktem brzegowym dla dwóch grup (oznaczonych kolorem czerwonym i zielonym). Jego przypisanie zależy od kolejności w jakiej DBSCAN będzie badał punkty.

2.2. NBC

Inne podejście do gęstościowego grupowania danych zaprezentowane zostało przez autorów, opublikowanego prawie dekadę później, algorytmu Neighbourhood-Based Clustering Algorithm (NBC). Podstawowa różnica polega na wykorzystaniu relatywnej gęstości lokalnej, a nie, jak w przypadku opisanego w poprzednim rozdziale DBSCAN, absolutnej gęstości globalnej. Lista parametrów wejściowych NBC ograni-

2. WYBRANE ALGORYTMY GĘSTOŚCIOWEGO GRUPOWANIA DANYCH

czona została do definiującej liczbę sąsiadów wartości k , która zastępuje stosowane w DBSCAN parametry ε i μ , nie zawsze pozwalające na poprawne rozpoznanie grup o różnej gęstości. Jest to szczególnie zauważalne w przypadkach, gdy w jednym zbiorze danych znajdują się zarówno małe i gęste, jak i większe, ale mniej gęste grupy, a odległość między najbliższymi punktami dwóch gęstych grup jest mniejsza niż odległość pomiędzy punktami rzadkich grup. Na przykładzie z rysunku 2.3 przypisanie wartości parametrów, pozwalających na poprawne rozpoznanie gęstych i położonych blisko siebie grup G3, G4 i G5, powodowałoby potraktowanie punktów z grup G1 i G2 jako szum. Natomiast zwiększenie parametrów, by wykryte mogłyby być grupy G1 i G2, oznacza, że punkty z położonych blisko siebie, gęstych grup G3, G4 i G5 trafiłyby do tej samej grupy. NBC, właśnie z racji wykorzystania gęstości lokalnej, może poradzić sobie w takiej sytuacji.

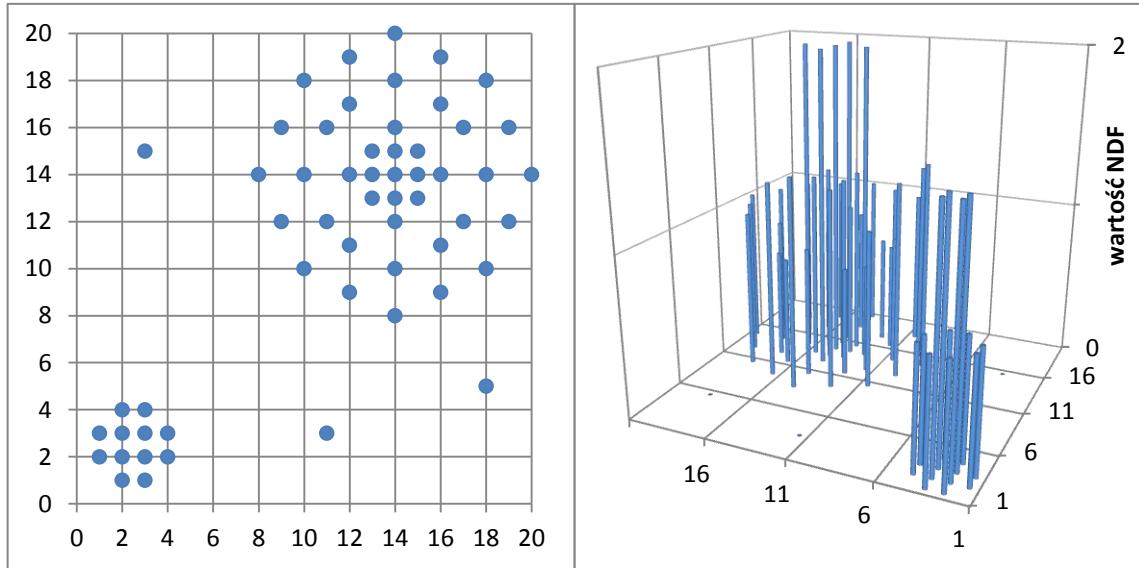


Rysunek 2.3 Przykład zbioru danych z zaznaczonym kolorami wynikiem grupowania algorytmu NBC, niemożliwym do uzyskania w algorytmie DBSCAN.

Istotnymi definicjami w tym algorytmie są: k -sąsiedztwo *punktu p* oznaczane, jako $kNB(p)$ (ang. *k-Neighborhood*) oraz *odwrotne k-sąsiedztwo punktu p*, czyli $R-kNB(p)$ (ang. *Reverse-k-Neighborhood*). Zgodnie z nazwą, k -sąsiedztwo punktu p to maksymalny zbiór punktów ze zbioru D , które oddalone są od punktu p o nie więcej niż k -ty najbliższy sąsiad tego punktu p . Liczba punktów należących do k -sąsiedztwa

2. WYBRANE ALGORYTMY GĘSTOŚCIOWEGO GRUPOWANIA DANYCH

dla większości punktów wynosi k , lecz może być większa, jeśli więcej punktów znajduje się w odległości takiej samej, jak k -ty najbliższy sąsiad. Natomiast liczność odwrotnego k -sąsiedztwa punktu p , czyli zbioru punktów, dla których punkt p należy do ich k -sąsiedztwa, może być natomiast dowolna i silnie zależy od gęstości najbliższego otoczenia.



Wykres 2.1 Ilustracja zależności pomiędzy obliczonymi dla parametru $k = 3$ wartościami NDF a gęstością otoczenia punktów w przykładowym zbiorze danych.

Stosunek powyższych wartości zdefiniowany został jako *wskaźnik gęstości sąsiedztwa NDF* (ang. *Neighborhood-based Density Factor*), który określa stosunek liczby punktów, których k -sąsiedztwa zawierają punkt p do liczby punktów należących do k -sąsiedztwa tego punktu p :

$$(2.2) \quad NDF(p) = \frac{|R-kNB(p)|}{|kNB(p)|}$$

W przypadku gdy współczynnik $NDF(p)>1$, punkt p nazywamy *punktem gęstym* (ang. *dense point*). Punkt p nazywamy *punktem rzadkim* (ang. *sparse point*), kiedy $NDF(p)<1$, a wartość taka oznacza, że nie należy on do k -sąsiedztwa punktów z jego k -sąsiedztwa. Natomiast *punkt równomierny* (ang. *even point*) zdefiniowany został [23] z użyciem wzoru $NDF(p)\approx 1$ oznaczającego, że wskaźnik może być tylko w pew-

2. WYBRANE ALGORYTMY GĘSTOŚCIOWEGO GRUPOWANIA DANYCH

nym przybliżeniu równy 1. Oznacza to jednak niespójność przedstawionego podziału punktów, dlatego w dalszym opisie przyjęłem definicję punktów równomiernych o $NDF(p)=1$.

Powysze podstawowe pojęcia wykorzystane zostały w analogicznych jak w algorytmie DBSCAN definicjach bezpośredniej gęstościowej osiągalności, gęstościowej osiągalności i gęstościowego połączenia. Punkt rdzeniowy z DBSCAN odpowiada w tym przypadku punktom o $NDF(p) \geq 1$, czyli punktom gęstym i równomiernym, co powiązane jest z zastąpieniem epsilonowego otoczenia k-sąsiedztwem. Zmiany te powodują, że mimo iż definicja *grupy* jako maksymalnego niepustego podzbioru punktów ze zbioru D, które są ze sobą gęstościowo połączone wygląda tak samo, to wyniki grupowania obydwu algorytmów mogą się od siebie znaczco różnić [21].

Algorytm NBC podzielić można na dwie główne fazy. Obliczanie wartości NDF można za niezależną fazę poprzedzającą właściwe grupowanie. Do obliczenia wartości NDF konieczne jest oczywiście wyznaczenie k-sąsiedztwa i odwrotnego k-sąsiedztwa wszystkich punktów. Jest to najbardziej czasochłonna operacja, dlatego autorzy algorytmu NBC sugerują wykorzystanie indeksu VA-File. W fazie właściwego grupowania zbiór wszystkich punktów jest iterowany w poszukiwaniu nieskasyfikowanych jeszcze punktów gęstych lub równomiernych, od których rozpoczyna się budowanie nowych grup. Po przypisaniu takiemu punktowi identyfikatora nowej grupy, jest on przypisywany wszystkim punktom z jego, wyznaczonego już wcześniej, k-sąsiedztwa. Dodatkowo, wszystkie punkty gęste lub równomiernie z jego k-sąsiedztwa są dodawane do zbioru ziaren, czyli punktów mogących potencjalnie rozszerzyć grupę o dodatkowe punkty. W kolejnym kroku zbiór ziaren jest iterowany, a dla każdego punktu, analogicznie jak powyżej, badane są wszystkie punkty z jego k-sąsiedztwa. Jeśli nie zostały one wcześniej przypisane do żadnej grupy, przypisywany jest im identyfikator aktualnie rozszerzanej grupy, a dodatkowo punkty gęste lub równomiernie dodawane są do zbioru ziaren. Pełen algorytm przedstawiony został za pomocą poniższego pseudokodu:

```

Algorithm NBC (D, k):
    preparation(D);
    calculateNDF(D, k);
    generate new clusterID;
    for each unclassified p ∈ D
        if expandCluster(clusterID, p, D) then
            generate new clusterID;
    return D;
end.

Function preparation(D):
    for each point p ∈ D
        mark p as unclassified;
        clear parameters p.kNB, p.rkNBno;
end.

Function calculateNDF(D, k):
    for each point p ∈ D
        insert into p.kNB all q ∈ k-Neighbourhood(p);
        for each q ∈ p.kNB increase q.rkNBno;
end.

Function expandCluster(clusterID, p, D): Boolean
    if p is dense then
        assign current clusterID to p;
        assign current clusterID to all q ∈ p.kNB;
        insert into queue Φ all q ∈ p.kNB such as q.NDF≥1;
        while Φ ≠ ∅ do
            q = first point in Φ;
            for each unclassified or noise x ∈ p.kNB
                assign current clusterID to x;
                if x.NDF≥1 then add x to Φ;
            remove q from Φ;
        return true;
    else
        mark p as noise;
        return false;
end.

```

Autorzy artykułu [23] nie przedstawili pseudokodu kluczowej w algorytmie procedury obliczania wskaźnika *NDF* wszystkich punktów. Sugerowany przez nich sposób wykorzystania indeksu VA-File nie jest opisany także w źródle [21], na które się powołują. Z ogólnego opisu można wywnioskować jedynie, że po zbudowaniu indeksu VA-File, k-sąsiedztwo każdego punktu liczone jest niezależnie.

Analiza pseudokodu pozwala zauważyc, że wprowadzony podział na punkty gęste, równomierne i rzadkie nie jest nigdzie wykorzystywany, a punkty z dwóch pierwszych grup (wszystkie o $NDF \geq 1$) traktowane są w jednolity sposób. Bardziej znaczącym wnioskiem jest jednak zauważona niespójność z przedstawioną wcześniej definicją grupy, z którą wiąże się niedeterminizm wyniku działania algorytmu, który zależy

2. WYBRANE ALGORYTMY GĘSTOŚCIOWEGO GRUPOWANIA DANYCH

od kolejności, w jakiej iterowane są punkty. Przedstawiony przez autorów algorytm tworzy grupę, jako maksymalny podzbiór punktów ze zbioru D , które są gęstościowo osiągalne z gęstego lub równomiernego punktu p , zupełnie pomijając fakt niesymetryczności tej relacji. Nie jest to zgodne z przyjętą przez nich definicją grupy, jako maksymalnego zbioru punktów gęstościowo połączonych.

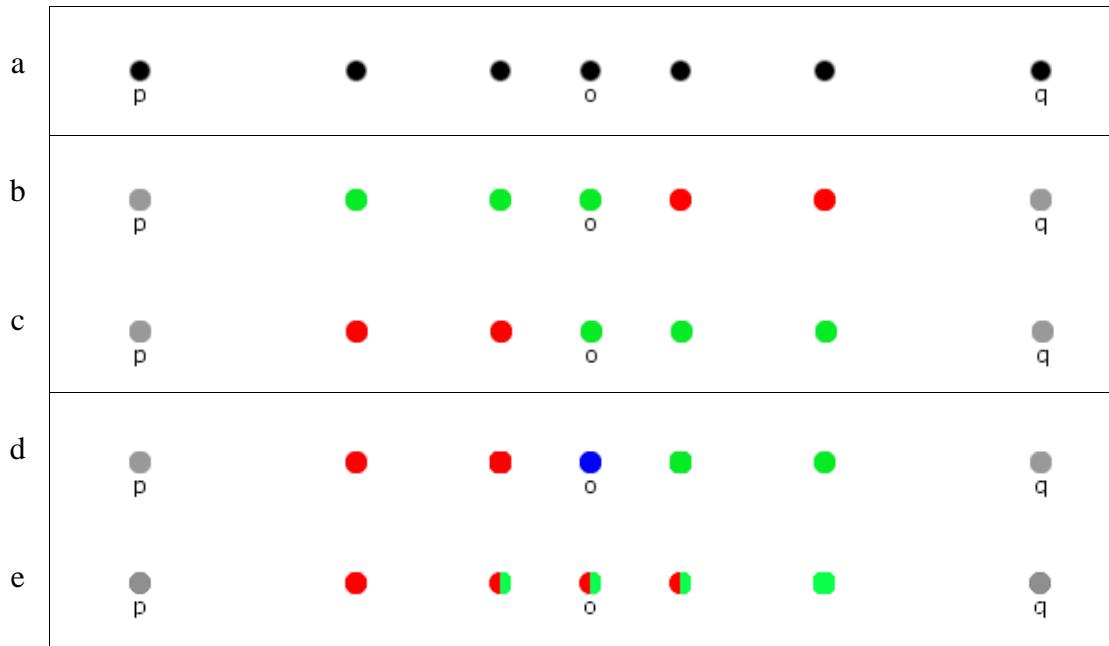
W związku z powyższą niespójnością, przedstawiam zmodyfikowaną wersję algorytmu, który działa w sposób deterministyczny i jest w pełni zgodna z przedstawionymi definicjami, w szczególności z definicją grupy, jako maksymalnego niepustego podzbioru punktów ze zbioru D , które są ze sobą gęstościowo połączone. Warto zauważyć, że zgodnie z definicjami dopuszczalne jest należenie punktu do wielu grup. Zmodyfikowana wersja, nazywana dalej dNBC (ang. Deterministic Neighbourhood-Based Clustering Algorithm), różni się od oryginalnej mniej restrykcyjnym warunkiem w funkcji *expandCluster*. NBC nie rozszerza grupy o punkty, które wcześniej przypisane były do innej grupy, podczas gdy dNBC rezygnuje z takiego ograniczenia. Zmiana wyróżniona została w poniższym pseudokodzie:

```
Function expandCluster(clusterID, p, D): Boolean
    if p is dense then
        assign current clusterID to p;
        assign current clusterID to all q ∈ p.kNB;
        insert into queue Φ all dense q ∈ p.kNB such as q. .NDF≥1;
        while Φ ≠ ∅ do
            q = first point in Φ;
            for each x ∈ q.kNB not classified to current clusterID
                assign current clusterID to x;
                if x.NDF≥1 then add x to Φ;
            remove q from Φ;
            return true;
    else
        mark p as noise;
        return false;
end.
```

Zlagodzenie warunku przerywającego rozszerzanie tworzonej grupy sprawia, że nawet jeśli dwa punkty nie są wzajemnie gęstościowo osiągalne to wystarczy jeden punkt, z którego są one gęstościowo osiągalne by zostały one przypisane do tej samej grupy. Podobnie jak w przypadku algorytmu DBSCAN, punkt należeć może do wielu grup, choć w przypadku dNBC możliwość przypisania punktu do kilku grup jest jednak jeszcze istotniejsza, ponieważ sytuacja taka dotyczyć może nie tylko punktów

2. WYBRANE ALGORYTMY GĘSTOŚCIOWEGO GRUPOWANIA DANYCH

brzegowych, lecz nawet większości punktów w grupie, w tym punktów gęstych i równomiernych.



Rysunek 2.4 Przykładowy zbiór danych (a) ilustrujący niedeterministyczność działania algorytmu NBC wraz z wynikami grupowania (z parametrem $k=1$) w przypadku: (b) NBC iterującego od punktu p w prawo, (c) NBC iterującego od punktu q w lewo, (d) NBC iterującego od punktu o w dowolnym kierunku, (e) dNBC, czyli zaproponowanej deterministycznej wersji NBC. Kolor szary oznacza szum, a pozostałe kolory odpowiadają przypisaniu do grup. W szczególności, kilka kolorów oznacza przypisanie do kilku grup.

Przykład z rysunku 2.4 ilustruje zarówno niedeterminizm oryginalnego NBC, jak i możliwość przynależności punktów do kilku grup w przypadku zastosowania dNBC. Nawet dość niewielki, bo tylko siedmioelementowy, zbiór danych. Zaprezentowany w części a rysunku, może zostać przez NBC pogrupowany na kilka różnych sposobów. NBC iterując po tym symetrycznym zbiorze od prawej do lewej strony generuje rozwiązanie inne (część b) niż w przypadku iterowania w przeciwnym kierunku (część c), ale co ciekawe, jeśli jako pierwszy badany będzie punkt o ze środka zbioru (dalejsza kolejność dowolna) otrzymany wynik (część d) zawiera nawet większą liczbę grup. W przypadku zastosowania dNBC umożliwiającego przypisywanie punktów do wielu grup, wynik jest deterministyczny i nie zależy od kolejności, w jakiej iterowane są punkty zbioru danych. W implementacji, podobnie jak w przypadku DBSCAN, wykorzystane zostało jednak uproszczenie polegające na zapamiętywaniu przypisania

tylko do jednej grupy, co nie wpływa jednak na złożoność i czas wykonywania algorytmu.

2.3. PREDECON

Wykorzystanie w algorytmie NBC wskaźnika gęstości sąsiedztwa rzeczywiście pozwala lepiej grupować dane w opisanych w poprzednim rozdziale przypadkach, gdy w jednym zbiorze danych znajdują się zarówno małe i gęste, jak i większe, ale rzadsze grupy. Zarówno DBSCAN, jak i NBC nie są jednak odporne na *przekleństwo wymiarów* [10], zgodnie z którym, w przypadku wielowymiarowych danych, stosowanie standardowych miar odległości opartych na wszystkich wymiarach traci sens. Wraz ze wzrostem liczbą wymiarów rozkład odległości pomiędzy punktami jest coraz węższy, a odległość pomiędzy najdalszymi sąsiadami jest coraz bliższa odległości pomiędzy najbliżymi sąsiadami, co utrudnia interpretację wykrywanych sąsiedztw.

Jednym ze sposobów rozwiązania problemu *przekleństwa wymiarów* w przypadku wielowymiarowych danych są techniki globalnej selekcji wymiarów, takie jak analiza głównych składowych (ang. Principal Component Analysis, PCA), operująca na wariancach w poszczególnych wymiarach. Po ograniczaniu liczby wymiarów poprzez wybór tych najbardziej znaczących, wykorzystane mogą być standardowe algorytmy grupowania danych. Analogicznie jak w przypadku DBSCAN użycie globalnych parametrów dla całego zbioru danych znacznie ogranicza możliwości wyszukiwania grup dla mocno zróżnicowanych zbiorów danych. W tym przypadku analogicznym problemem są przypadki zawierające zbiory punktów tworzące grupy w różnych podprzestrzeniach badanej przestrzeni.

Podejściem, które ma wykluczyć powyższe problemy, występujące przy zastosowaniu globalnej redukcji liczby wymiarów, ma być grupowanie w podprzestrzeniach [5]. Zgodnie z tą koncepcją każda grupa opisywana jest dodatkowo przez zbiór atrybutów definiujących podprzestrzenie, w której istnieje. Taki lokalny wybór podprzestrzeni, niezależnie dla każdej grupy, pozwala uzyskać większą elastyczność niż w przypadku metod globalnego wyboru podprzestrzeni. Zastosowania pierwszych algorytmów wykorzystujących przedstawione podejście, takich jak CLIQUE [2]

2. WYBRANE ALGORYTMY GĘSTOŚCIOWEGO GRUPOWANIA DANYCH

czy SUBCLU [11], są jednak mocno ograniczone ze względu na rozmiar uzyskiwanych wyników, które zawierają wszystkie grupy w 2^d (gdzie d to liczba wymiarów) podprzestrzeniach. Co więcej, w większości zastosowań oczekiwany jest deterministyczny i jednoznaczny podział elementów zbioru danych, podczas gdy w przypadku powyższych algorytmów elementy mogły należeć jednocześnie do grup w wielu podprzestrzeniach. Powyższy problem rozwiązać miały algorytmy Przewidywanego Grupowania (ang. Projected Clustering), które nie rozpoczynają grupowania od przeszukiwania pojedynczych wymiarów, lecz od szacowania i przewidywania w całej przestrzeni. Flagowy algorytm tej kategorii, PROCLUS [1] (PROjected CLUSTering), wykorzystuje do tego celu zmodyfikowaną metodę k-środków, która po określaniu początkowych „środków grup” w kolejnych iteracjach rozszerza grupy uwzględniając wagi oparte na wariancjach poszczególnych wymiarów. Wciąż nie rozwiązuje to niestety problemu jednoznacznego przypisania punktów do grup, zwłaszcza, że przez stosowanie przybliżeń i losowość algorytmu Przewidywanego Grupowania są niedeterministyczne.

Oparty na DBSCAN algorytm PreDeCon [5], dzięki wprowadzonej koncepcji preferencji podprzestrzeni, w przeciwieństwie do przedstawionych powyżej algorytmów wykorzystujących podprzestrzenie, deterministycznie z dokładnością do punktów brzegowych grupuje elementy w rozłączne grupy uwzględniając przy tym wszystkie wymiary. Zachowuje on podstawowe idee algorytmów gęstościowego grupowania danych, w tym kluczowe pojęcie punktów gęstościowo połączonych. PreDeCon wykorzystuje dodatkowo wariancję punktów *epsilonowego sąsiedztwa* wzdłuż wszystkich atrybutów przestrzeni, wyrażoną wzorem:

$$(2.3) \quad VAR_{A_i}(N_\varepsilon(p)) = \frac{\sum_{q \in N_\varepsilon(p)} (dist(\pi_{A_i}(p), \pi_{A_i}(q)))^2}{|N_\varepsilon(p)|},$$

gdzie $\pi_{A_i}(p)$ oznacza projekcję punktu p na i-ty atrybut. Wielkość tak obliczonej wariancji wzdłuż atrybutu A_i decyduje o wartości ω_i wektora preferencji podprzestrzeni $w_p = (\omega_1, \omega_2, \dots, \omega_d)$, która przyjmuje wartość stałej κ (gdzie $\kappa \gg 1$) w przypadku $VAR_{A_i}(N_\varepsilon(p)) < \delta$ lub wartość 1 w przeciwnym przypadku. Powyższe wartości

ω_i wykorzystywane są w zmodyfikowanym wzorze na $dist_p(p, q)$, *ważone preferencjami punktu p podobieństwo* punktu q do punktu p:

$$(2.4) \quad dist_p(p, q) = \sqrt{\sum_{i=1}^d \omega_i \cdot (\pi_{A_i}(p) - \pi_{A_i}(q))^2}$$

Taka definicja podobieństwa zmienia znaczenie parametru ε , który w przypadku użycia standardowej Euklidesowej odległości był bardzo podatny na opisane wcześniej *przekleństwo wymiarów*. Jak zauważali autorzy algorytmu, wprowadza ona jednak problem niesymetryczności, a co za tym idzie niedeterminizm działania algorytmu. Zaproponowanym przez nich rozwiązaniem jest połączenie ważonych wektorami preferencji punktów p i q miar $dist_p(p, q)$ i $dist_q(q, p)$ w *ważone preferencjami podobieństwo* $dist_{pref}(p, q)$ za pomocą operacji zwracającej maksymalną z tych wartości:

$$(2.5) \quad dist_{pref}(p, q) = \max(dist_p(p, q), dist_q(q, p))$$

Tak zdefiniowana, symetryczna miara, wykorzystywana jest następnie do obliczania *ważonego preferencjami epsilonowego sąsiedztwa* $N_\varepsilon^{w_0}(p)$, które analogicznie jak epsilonowe sąsiedztwo w DBSCAN składa się z punktów oddalonych nie dalej niż o wartość ε .

Zmodyfikowane w stosunku do DBSCAN zostały natomiast definicje *punktu rdzeniowego* $Core_{den}^{pref}$ i *bezpośredniej osiągalności* $DirReach_{den}^{pref}$. Modyfikacja definicji zaostrza wymagania na punkt rdzeniowy i bezpośrednio osiągalny o warunek *maksymalnej wymiarowości preferowanej podprzestrzeni* dla danego punktu, $PDim(N_\varepsilon(p))$, zdefiniowanej, jako liczba wymiarów o małej wariancji punktów z epsilonowego sąsiedztwa, która nie może być większa od parametru λ :

$$(2.6) \quad DirReach_{den}^{pref}(q, p) \Leftrightarrow Core_{den}^{pref}(q) \wedge p \in N_\varepsilon^{w_0}(p) \wedge PDim(N_\varepsilon(p)) \leq \lambda$$

$$(2.7) \quad Core_{den}^{pref}(q) \Leftrightarrow |N_\varepsilon^{w_0}(p)| \geq \mu \wedge PDim(N_\varepsilon(q)) \leq \lambda$$

2. WYBRANE ALGORYTMY GĘSTOŚCIOWEGO GRUPOWANIA DANYCH

Analogicznie jak w bazowym algorytmie DBSCAN *grupa* zdefiniowana została jako maksymalny niepusty podzbiór połączonych punktów, które spełniają warunki ważonej preferencjami osiągalności. Pozostałe punkty oznaczane są w pseudokodzie [5] jako *szum*, mimo iż autorzy artykułu jawnie nie definiują takiego pojęcia.

```
Algorithm PreDeCon (D, ε, μ, λ, δ):
    preparation(D, λ, δ, ε, κ);
    generate new clusterID;
    for each unclassified p ∈ D
        if expandCluster(clusterID, p, D, λ) then
            generate new clusterID;
    return D;
end.

Function preparation(D, λ, δ, ε, κ):
    mark all p ∈ D as unclassified;
    for each p ∈ D calculateVectorW0(Nε(p), λ, δ);
    for each p ∈ D calculateWeightedNeighbourhood(ε, κ);
end.

Function expandCluster(clusterID, p, D, λ): Boolean
    if p is core then
        insert all q ∈ Npref(p) into queue Φ;
        while Φ ≠ ∅ do
            x = first point in Φ;
            for each r ∈ DirReachw0(x)
                if r is unclassified or noise then
                    assign current clusterID to r;
                    if r is unclassified then add r to Φ;
            remove x from Φ;
        return true;
    else
        mark p as noise;
        return false;
end.
```

Przedstawiony za pomocą powyższego pseudokodu algorytm PreDeCon podzielić można na dwie fazy. Główna pętla, analogiczna jak w DBSCAN, iteruje po wszystkich niepogrupowanych wcześniej punktach. W przypadku napotkania punktu rdzeniowego, algorytm buduje nową grupę, rozszerzając ją o kolejne punkty osiągalne z danego punktu rdzeniowego. W rzeczywistości, aby sprawdzić, czy punkt jest rdzeniowy lub wyznaczyć punkty osiągalne w preferowanej podprzestrzeni, wcześniej wykonana musi być pierwsza faza, w ramach której dla każdego punktu wyznaczany jest jego wektor preferencji podprzestrzeni w_0 , do czego konieczne jest znalezienie standardowego epsilonowego sąsiedztwa. Oznacza to, że dla każdego punktu, czaso-

2. WYBRANE ALGORYTMY GĘSTOŚCIOWEGO GRUPOWANIA DANYCH

chłonna operacja wyznaczania sąsiedztwa wykonywana jest podwójnie, raz dla sąsiedztwa epsilonowego i raz dla ważonego wektorem preferencji, przy czym, w tym drugim, bardziej skomplikowanym obliczeniowo przypadku, dla każdej pary punktów odległość liczona jest nie raz, ale dwa razy, z wykorzystaniem wektorów preferencji obydwu punktów.

3. WYKORZYSTANIE NIERÓWNOŚCI TRÓJKĄTA

Wszystkie przedstawione w poprzednim rozdziale algorytmy gęstościowego grupowania danych opierają się na definicjach punktów osiągalnych i sąsiedztw. Niezależnie od tego, czy jest to epsilonowe otoczenie w DBSCAN, czy k-sąsiedztwo w NBC, najprostszym sposobem ich wyznaczania jest obliczenie odległości pomiędzy wszystkimi punktami. Analiza złożoności przedstawionych algorytmów prowadzi do wniosku, że są to operacje mające największy wpływ na wydajność całego procesu grupowania, w szczególności w przypadku zbiorów wielowymiarowych. Proste obliczanie odległości pomiędzy wszystkimi punktami oznacza, że algorytm miałby co najmniej kwadratową złożoność, co znacznie ograniczałoby możliwości jego zastosowania do większych zbiorów danych.

Rozwiązaniem usprawniającym wyznaczanie sąsiedztwa, wybranym przez autorów algorytmu DBSCAN [7], jest wykorzystanie indeksu wyszukiwania przestrzennego, a dokładnie indeksu R*-drzewo [3]. Główna idea indeksu opiera się na grupowaniu obiektów leżących blisko siebie i aproksymowaniu ich za pomocą minimalnych prostokątów ograniczających (ang. minimum bounding rectangle). Prostokąty te grupowane są rekurencyjnie w większe minimalne regiony ograniczające, aż do poziomu całego zbioru danych reprezentowanego przez korzeń R*-drzewa. Algorytm wyszukiwania epsilonowego sąsiedztwa z wykorzystaniem R*-drzewa oparty jest na właściwości mówiącej, że jeśli obszar zapytania nie przecina się z badanym minimalnym regionem ograniczającym, to zgrupowanie w nim punkty nie należą do zbioru odpowiedzi. Na tej podstawie, na kolejnych poziomach drzewa, ograniczany jest zbiór gałęzi, które mogą zawierać rozwiązanie, czyli w przypadku DBSCAN ograniczany jest zbiór punktów, które mogą należeć do epsilonowego otoczenia. Przy założeniu, że wyszukiwane sąsiedztwo jest stosunkowo niewielkim podzbiorem całego zbioru danych, a co za tym idzie, zapytanie odczytuje zwykle tylko kilka ścieżek R*-drzewa, średnia złożoność zapytania o sąsiedztwo pojedynczego punktu szacowana jest na $O(\log n)$. Musi być ona pomnożona przez liczbę wszystkich punktów n, ponieważ

3. WYKORZYSTANIE NIERÓWNOŚCI TRÓJKĄTA

zapytanie powtarzane jest dla każdego z nich, co daje ostateczną złożoność rzędu $O(n \cdot \log n)$, wciąż zdecydowanie mniejszą od złożoności kwadratowej.

Tradycyjne struktury R*-drzewa nie najlepiej radzą sobie jednak z zapytaniami o epsilonowe otoczenie czy k-sąsiedztwo, w przypadku zbiorów o dużej liczbie wymiarów. Tłumacząc to chęcią uniezależnienia się od tego ograniczenia, twórcy algorytmu NBC zdecydowali się wykorzystać, oparty na płaskiej strukturze, indeks VA-File (Vector Approximation file) [21]. Przestrzeń danych reprezentowana jest przez zbiór wielowymiarowych komórek, do których przydzielone są poszczególne punkty. Przydział zapisywany jest za pomocą przypisanych do punktów bitowych wektorów, które umożliwiają efektywne odfiltrowywanie punktów i ograniczanie zbioru potencjalnych k-sąsiadów. Bitowy wektor reprezentujący przynależność punktu do konkretnej komórki, w zależności od zastosowanego stopnia przybliżenia i liczby wymiarów, jest wielokrotnie mniejszy od oryginalnego punktu. W połączeniu z faktem, że w procesie filtrowania binarne wektory przeszukiwane są sekwencyjnie, a nie losowo, jak w przypadku indeksów drzewiastych, oznacza to znaczne ograniczenie kosztownych operacji odczytu pamięci.

Jednym z nowszych pomysłów na usprawnienie wyszukiwania sąsiedztwa przez ograniczenie liczby obliczeń rzeczywistych odległości jest wykorzystanie nierówności trójkąta. W przeciwieństwie do opisywanych powyżej indeksów R*-drzewo i VA-File, nie wykorzystuje on bezpośredniego podziału przestrzeni na komórki, lecz opiera się na opisanych poniżej własnościach.

Zgodnie z powszechnie znaną nierównością trójkąta, dla trzech dowolnych punktów odległość pomiędzy dowolną parą punktów jest mniejsza niż lub równa sumie odległości pomiędzy pozostałymi dwoma parami, co można zapisać jako:

$$(3.1) \quad \bigwedge_{p,q,r} dist(p,r) \leq dist(p,q) + dist(q,r),$$

gdzie $dist(p,q)$ oznacza odległość pomiędzy punktami p i q . W artykule [14] sugerowane jest przekształcenie powyższego równania do bardziej intuicyjnej w dalszych rozważaniach postaci:

$$(3.2) \quad \bigwedge_{p,q,r} dist(p,r) - dist(q,r) \leq dist(p,q)$$

Na jej podstawie autorzy wspomnianego artykułu wnioskują, że jeśli różnica odległości (prawa strona równania) jest większa od wartości ε , to odległość pomiędzy nimi (lewa strona równania) także jest większa od ε :

$$(3.3) \quad dist(p,r) - dist(q,r) > \varepsilon \Rightarrow dist(p,q) > \varepsilon$$

Prowadzi to do wniosku, że jeśli wspomniana różnica odległości dwóch punktów od tego samego punktu referencyjnego jest większa od wartości ε , to nie mogą one wzajemnie należeć do epsilonowych otoczeń:

$$(3.4) \quad dist(p,r) - dist(q,r) > \varepsilon \Rightarrow p \notin N_\varepsilon(q) \wedge q \notin N_\varepsilon(p)$$

Na powyższej *nierówności wykluczania z epsilonowego sąsiedztwa* opiera się główna idea modyfikacji opartej na nierówności trójkąta [14]. Zastosowanie implikacji (3.4) umożliwia proste oszacowanie odległości pomiędzy punktami i odfiltrowanie tych punktów, które nie mogą należeć do epsilonowego sąsiedztwa punktu p . Dodatkowo, posortowanie zbioru danych według odległości od tego samego punktu, nazywanego dalej *punktem referencyjnym*, umożliwia ograniczenie liczby punktów, dla których powyższe szacowanie musi zostać sprawdzone. W zbiorze posortowanym według niemalejącej odległości od punktu referencyjnego, dla dowolnego punktu p łatwo wyznaczyć jest punkty q_b , jako pierwszy punkt poprzedzający punkt p , dla którego spełniona jest nierówność:

$$(3.5) \quad dist(p,r) - dist(q_b,r) > \varepsilon$$

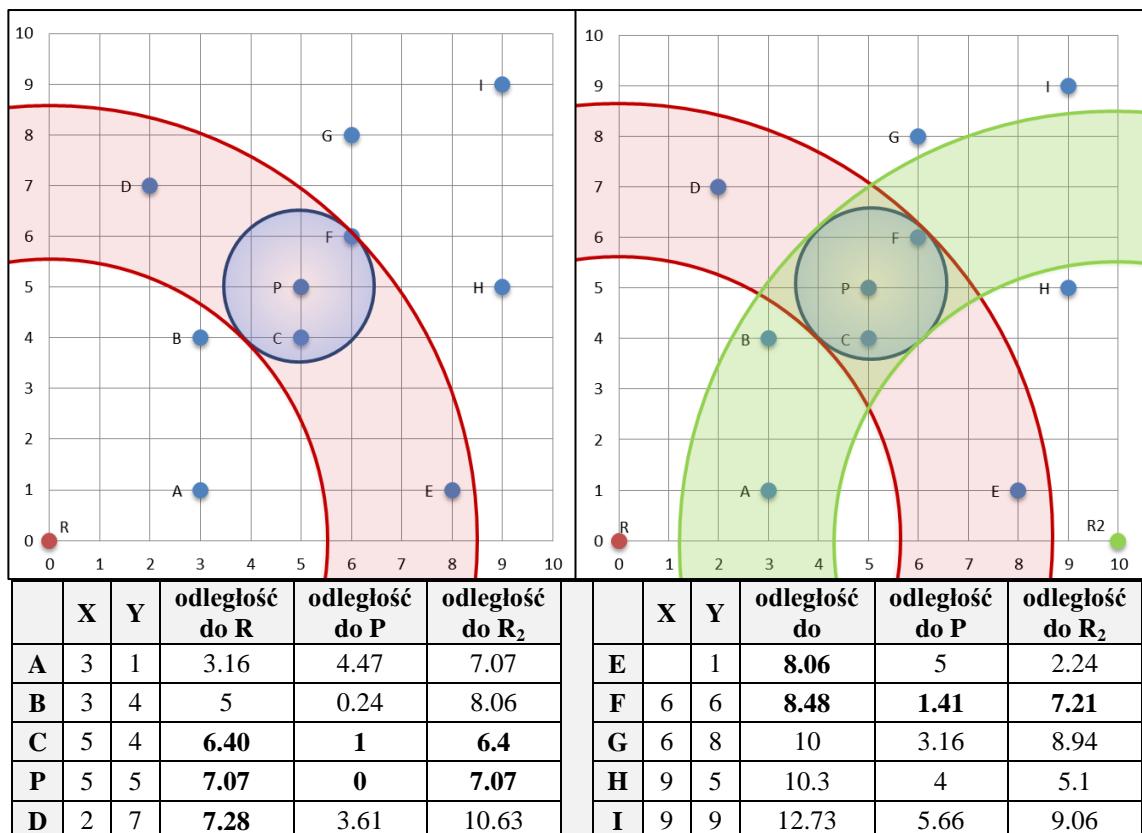
i q_f , jako pierwszy po punkcie p , dla którego spełniona jest analogiczna nierówność:

$$(3.6) \quad dist(q_f,r) - dist(p,r) > \varepsilon.$$

3. WYKORZYSTANIE NIERÓWNOŚCI TRÓJKĄTA

Przedstawione powyżej nierówności autorzy [14] wykorzystują do udowodnienia, że do epsilonowego sąsiedztwa punktu p mogą należeć tylko punkty pomiędzy q_b i q_f . Dlatego tylko dla takiego podzbioru punktów sprawdzona musi być nierówność (3.4).

Złożoność obliczeniowa wyszukania epsilonowego otoczenia n punktów bardzo zależy od uzyskanej selektywności, na którą wpływać może charakterystyka zbioru danych oraz dobór punktu referencyjnego. Może zostać ona oszacowana jako $O(s \cdot n)$, gdzie s to liczba punktów niespełniających nierówności wykluczania z epsilonowego sąsiedztwa (3.4), dla których obliczone muszą być rzeczywiste odległości w ramach wyszukiwania epsilonowego otoczenia pojedynczego punktu. Porównanie rzeczywistego wzrostu wydajności dla testowych zbiorów danych przedstawiłem w rozdziale 5.



Wykres 3.1. Przykład zastosowania nierówności trójkąta z jednym punktem referencyjnym (lewa strona) i wieloma punktami referencyjnymi (prawa strona) do znalezienia epsilonowego otoczenia punktu P dla wartości ε równej 1,5.

Przykład szacowania odległości z wykorzystaniem nierówności trójkąta usprawniającego poszukiwanie epsilonowego otoczenia zilustrowany został na wykresie 3.1

(lewa strona). Punkty w tabeli posortowane zostały według niemalejącej odległości do punktu referencyjnego R o współrzędnych $(0,0)$. Zaznaczony na czerwono wycinek ilustruje jak nierówność trójkąta ogranicza obszar poszukiwań w przypadku ε równego 1,5. Tylko dla należących do tego zakresu punktów C, D, E i F , których odległość od punktu referencyjnego R różni się od odległości referencyjnej punktu P o nie więcej niż ε , obliczone muszą być rzeczywiste odległości do punktu P . Pozostałe punkty, czyli A, B, G, H i I , mogą zostać pominięte, ponieważ zgodnie z przedstawionymi wyżej równaniami nie mogą należeć do epsilonowego otoczenia punktu P . Oczywiście zaznaczony na czerwono wycinek wciąż pozostaje znaczco większy od niebieskiego okręgu ograniczającego epsilonowe otoczenie punktu P .

Rozwiązaniem dodatkowo ograniczającym brany pod uwagę zakres dziedziny jest wprowadzenie dodatkowych punktów referencyjnych [14], tak jak zostało to przedstawione na wykresie 3.1 (prawa strona), na którym dodany został punkt R_2 . Zgodnie z dalszym opisem, rzeczywiste odległości muszą być obliczane tylko dla punktów spełniających nierówność trójkąta w stosunku do wszystkich punktów referencyjnych, a więc znajdujących się na przecięciu wycinka czerwonego i zielonego. W tym przypadku ze zbioru punktów dla których muszą być obliczone rzeczywiste odległości, na podstawie niespełnienia nierówności trójkąta dla punktu R_2 , wykluczane są także punkty D i E .

Zaprezentowane w powyższym przykładzie rozwiązanie z wieloma punktami referencyjnymi jest modyfikacją opisanej wcześniej wersji z jednym punktem referencyjnym. Wszystkie punkty zbioru danych, tak jak w podstawowej wersji sortowane są według niemalejącej odległości do głównego punktu referencyjnego i to na jej podstawie ograniczany jest wstępnie zbiór punktów, które mogą należeć do epsilonowego otoczenia. Punkty spełniające warunek nierówności trójkąta dla głównego punktu referencyjnego sprawdzane są dodatkowo pod względem spełniania tego warunku dla kolejnych punktów referencyjnych. Niespełnienie warunku dla przynajmniej jednego punktu referencyjnego pozwala bez sprawdzania odległości do kolejnych punktów referencyjnych i obliczania rzeczywistej odległości wykluczyć punkt z epsilonowego otoczenia. Zwiększenie selektywności rozwiązania, oznacza jednak większy koszt przygotowań, podczas których dla wszystkich n punktów obliczane są odległości do wielu, a nie jednego punktu referencyjnego. Dla dużej liczby punk-

3. WYKORZYSTANIE NIERÓWNOŚCI TRÓJKĄTA

tów referencyjnych coraz większe znaczenie może mieć także konieczność wielokrotnego sprawdzenia warunków dla wszystkich punktów pomiędzy q_b i q_f . Oznacza to, że zwiększanie liczby punktów referencyjnych nie zawsze musi oznaczać zwiększenie wydajności. Największe znaczenie ma tu ewentualny wzrost selektywności rozwiązania, na którą wpływ mają przedstawione w kolejnym podrozdziale parametry.

3.1. PARAMETRYZACJA ROZWIĄZANIA

Zgodnie z przedstawionymi powyżej wzorami, opisywana modyfikacja, wykorzystująca nierówność trójkąta może być oparta na dowolnym punkcie referencyjnym, na który nie są nakładane żadne ograniczenia. Jedynym warunkiem, który musi być spełniony, jest obliczanie odległości do tego samego punktu referencyjnego dla wszystkich punktów zbioru danych. Jednak, tak jak zostało to już wspomniane, wybór punktu referencyjnego może mieć wpływ na wielkość zbioru punktów, dla których konieczne jest obliczenie rzeczywistych odległości od punktu p . Szczegóły zostały przedstawione w kolejnych podrozdziałach, dotyczących możliwych strategii wybierania jednego lub wielu punktów referencyjnych.

3.1.1. Strategia wyboru jednego punktu referencyjnego

Spotykanym w literaturze [13], [14] przykładem naturalnego punktu referencyjnego jest punkt zerowy, czyli punkt o zerowych wartościach wszystkich atrybutów. Jego podstawową zaletą jest uproszczony sposób obliczania odległości do niego. W zależności od dziedziny punktów i ich rozłożenia, wybór innego punktu może znaczco zwiększyć selektywność warunków nierówności trójkąta. Celem jest uzyskanie jak najszerzego rozkładu odległości od punktu referencyjnego, jak najbardziej podobnego do rozkładu jednostajnego. Punkt zerowy nie jest najlepszym wyborem w przypadku, gdy dziedziny atrybutów zbioru danych obejmują wartości położone daleko od środka układu współrzędnych lub też otaczają go ze wszystkich stron. Rozwiązaniem może być przeskalowanie zbioru danych, jednak w przypadku dużych ilości danych może okazać się to kosztowne.

Alternatywnym rozwiązaniem jest wybór innego punktu referencyjnego. W pracy dyplomowej skupiłem się na analizie kilku charakterystycznych strategii wyboru punktów referencyjnych, takich jak:

- *strategia punktu zerowego* – o zerowych wartościach wszystkich atrybutów;
- *strategia punktu minimalnego* – o wartościach wszystkich atrybutów równych minimalnym wartośćom ich dziedzin;
- *strategia punktu maksymalnego* – o wartościach wszystkich atrybutów równych maksymalnym wartośćom ich dziedzin;
- *strategia punkty środkowego* – o wartościach wszystkich atrybutów równych środkowym wartośćom ich dziedzin;
- *strategia punktu losowego* – wybierany losowo punkt ze zbioru danych.

Taki wybór punktów referencyjnych pokrywa najbardziej charakterystyczne, a zarazem intuicyjne przypadki, dzięki czemu możliwe powinno być na ich podstawie wyciągnięcie ogólnych wniosków. Przedstawione strategie wyboru punktów referencyjnych nie są przy tym złożone obliczeniowo. Wybór punktu minimalnego, maksymalnego i środkowego wymaga jedynie jednokrotnego odczytu zbioru danych (przy założeniu, że nie znamy dziedzin atrybutów a'priori), a punkt zerowy i losowy wybierane są w czasie stałym. Przy znajomości wniosków z analizy powyższych strategii możliwa wydaje się próba opracowania bardziej złożonych strategii i heurystyk pozwalających dobierać punkty referencyjne do charakterystyki zbioru danych.

Mocne i słabe strony wszystkich przedstawionych strategii, poparte wynikami testów i przykładami przedstawione zostały w rozdziale 5.3.2.1.

3.1.2. Strategia wyboru wielu punktów referencyjnych

Alternatywnym rozwiązaniem do opisanego zwiększenia selektywności warunku nierówności trójkąta jest dodanie kolejnych warunków opartych na innych punktach referencyjnych. Dodanie każdego kolejnego punktu referencyjnego to koszt obliczenia i przechowania n odległości. Pozwala to jednak wykluczyć kolejne punkty ze zbioru punktów, dla których obliczona musi być rzeczywista odległość. Wystarczy, że punkt nie spełnia warunku przynajmniej dla jednego z punktów referencyjnych, by móc wykluczyć go z tego zbioru. Jak ilustruje to wykres 3.1, przecięcie dwóch podprzestrzeni

3. WYKORZYSTANIE NIERÓWNOŚCI TRÓJKĄTA

spełniających warunek nierówności trójkąta dla dwóch różnych punktów referencyjnych może znacząco ograniczać zbiór punktów, które mogą należeć do epsilonowego otoczenia. Nie jest to jednak stała zależność, a zwiększenie liczby punktów referencyjnych może mieć także negatywny wpływ na wydajność, jeśli nie ograniczają one wystarczająco podprzestrzeni, a powodują jedynie dodatkowy narzut na dodatkowe obliczanie odległości i sprawdzanie spełnienia nierówności wykluczania z epsilonowego sąsiedztwa.

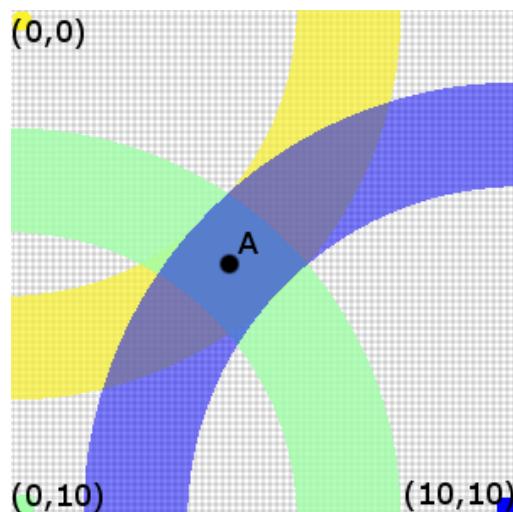
Podobnie jak w przypadku jednego punktu referencyjnego, tak też w przypadku wykorzystywania wielu punktów referencyjnych, duże znaczenie ma nie tylko liczba punktów, ale też strategia ich wyboru. Mając na uwadze cel, jakim jest maksymalne zwiększenie zbioru punktów, które wykluczyć możemy z epsilonowego otoczenia badanego punktu, dobrać należy najwydajniejszą strategię wyboru punktów referencyjnych. Przedstawione w poprzednich podrozdziałach strategie wyboru pojedynczego punktu musiały zostać zmodyfikowane tak, by możliwy był wybór wielu punktów, w szczególności od jednego do d punktów, gdzie d to liczba atrybutów. Duży wpływ na wybór punktów referencyjnych ma dodatkowy parametr, jakim jest strategia wyboru atrybutów. W pracy dyplomowej analizuję następujące strategie:

- *strategia atrybutów o najszerzzych dziedzinach;*
- *strategia atrybutów o najwęższych dziedzinach;*
- *strategia losowych atrybutów;*
- *strategia pierwszych atrybutów.*

Szczególnie ciekawe wydają się wyniki analizy wydajności najbardziej charakterystycznych strategii wyboru atrybutów o najszerzzych lub najwęższych dziedzinach. Strategia wyboru losowych atrybutów umożliwia ma porównanie i ocenę wpływu strategii na wydajność. Strategia pierwszych atrybutów, w której atrybuty wybiera się w oryginalnej kolejności, uznać można za szczególny przypadek strategii losowej. Co istotne, nie wymaga ona znajomości charakterystyki zbioru danych, a jej wynik jest przewidywalny i powtarzalny.

Selekcja atrybutów ma bezpośredni wpływ na zbiór punktów referencyjnych wybieranych w przypadku zastosowania strategii punktów minimalnych, maksymalnych

i quasi-środkowych³. Natomiast jest ona zbędna w przypadku strategii punktów losowych oraz strategii punktu zerowego. W pierwszym przypadku, punkty wybierane są losowo z całego zbioru danych, a zwiększoną jest tylko ich liczba. W drugim, nawet wybrana liczba punktów referencyjnych nie ma znaczenia, ponieważ może być tylko jeden punkt zerowy.



Rysunek 3.1 Porównanie przecięć obszarów spełniających nierówność trójkąta dla trzech skrajnych punktów referencyjnych.

W przypadku *strategii punktów minimalnych*, jak pierwszy wybierany jest punkt o minimalnych wartościach dziedzin wszystkich atrybutów. Następnie, każdy kolejny punkt referencyjny generowany jest przez zmianę wartości jednego atrybutu na maksymalną. Zmieniane są wartości kolejnych atrybutów z listy utworzonej zgodnie z wybraną przez użytkownika strategią wyboru atrybutów. Taki algorytm gwarantuje, że niezależnie od zastosowanej strategii wyboru atrybutów, kolejne punkty referencyjne różnić się będą wartością co najwyżej jednego atrybutu. Jak widać na przykładzie z rysunku 3.1, przecięcia obszarów spełniających nierówność wykluczania z epsilonowego sąsiedztwa dla sąsiednich punktów skrajnych (np. żółtego i zielonego) są mniejsze niż dla punktów przeciwnielego (żółty i niebieski).

³ Nazwa strategii zmieniona została dla podkreślenia, że zgodnie z zamieszczonym dalej opisem, w przypadku wielu punktów referencyjnych, są to punkty środkowe w kontekście dziedzin wybranych atrybutów, a nie całego zbioru danych, tak jak miało to miejsce w przypadku wyboru jednego punktu referencyjnego.

3. WYKORZYSTANIE NIERÓWNOŚCI TRÓJKĄTA

W przypadku *strategii punktów maksymalnych* punkty referencyjne generowane są w sposób analogiczny do opisanego powyżej. Jednak jako pierwszy wybierany jest punkt maksymalny, a wartości wybranych atrybutów zmieniane są na minimalne.

W przypadku *strategii punktów quasi-środkowych* nie jest już wybierany punkt o wartościach wszystkich atrybutów równych środkom ich dziedzin. Dla wielu punktów referencyjnych większą selektywność uzyskać można przy wyborze punktów quasi-środkowych, czyli środkowych w kontekście wybranych atrybutów, a nie całego zbioru danych. Dla i-tego punktu referencyjnego, środkową wartość dziedziny przyjmuje i-ty z listy atrybutów wygenerowanej zgodnie z wybraną strategią wyboru atrybutów, a pozostałe przyjmują wartości skrajne, a dokładnie minimalne.

Jak pokazują wyniki testów z rozdziału 5.3.2.2, strategia wyboru atrybutów jest tym istotniejsza, im bardziej zróżnicowane są dziedziny poszczególnych atrybutów.

3.2. ZASTOSOWANIE NIERÓWNOŚCI TRÓJKĄTA W ALGORYTMACH GĘSTOŚCIOWEGO GRUPOWANIA DANYCH

Zwiększająca wydajność algorytmów gęstościowego grupowania danych modyfikacja wykorzystująca nierówność trójkąta z powodzeniem może być stosowana nie tylko w przypadku procesu wyznaczania epsilonowego otoczenia dowolnego punktu, ale także k-sąsiedztwa. Wykorzystanie jej w analizowanych algorytmach gęstościowego grupowania danych DBSCAN, NBC i PreDeCon opisałem w kolejnych podrozdziałach.

3.2.1. Algorytm TI-DBSCAN

Wykorzystanie nierówności trójkąta, jako nowego sposobu usprawnienia algorytmu DBSCAN, przedstawione zostało po raz pierwszy w artykule [14]. Opublikowana wersja zmodyfikowanego algorytmu, nazwana przez autorów TI-DBSCAN, podczas inicjalizacji punktów poza oznaczeniem ich jako niegrupowane, oblicza dla każdego z nich odległość do wcześniejszego wybranego punktu referencyjnego. To właśnie zgodnie z tymi wartościami sortowane są niemalejąco wszystkie punkty wejściowego

zbioru danych. Alternatywnym rozwiązaniem pozwalającym uniknąć sortowania, potencjalnie bardzo dużych zbiorów danych, wydaje się tu stworzenie oddzielnej struktury posortowanych wskaźników. W praktyce, co potwierdziły wyniki, rozwiązanie z sortowaniem oryginalnego zbioru danych znacznie przyspiesza powtarzany później dla każdego punktu proces odczytów danych. Dzięki wprowadzeniu sekwencyjności może być znaczco zoptymalizowany, ponieważ kolejne odczytywane punkty znajdują się w jednym lub w sąsiednich blokach pamięci.

```

Function TI-preparation(D) :
    select reference point r;
    for each p ∈ D
        set p as unclassified;
        calculate p.refDistance as distance(p, r);
        p.neighboursNo = 1;
        p.border = Ø;
    sort D according to refDistance;
end.

Function TI-Neighbourhood(p, D, ε) : set of points
    return union of TI-BackwardNeighbourhood(p, D, ε)
        and TI-ForwardNeighbourhood(p, D, ε)
end.

Function TI-BackwardNeighbourhood (p, D, ε) : set of points
    backwardThreshold = p.refDistance - ε;
    iterate backward D starting with x = point before p
        if x.refDistance < backwardThreshold then break;
        if distance(p, x) ≤ ε then add x to resultSet;
    return resultSet;
end.

Function TI-ForwardNeighbourhood (p, D, ε) : set of points
    forwardThreshold = p.refDistance + ε;
    iterate forward D starting with x = point after p
        if x.refDistance > forwardThreshold then break;
        if distance(p, x) ≤ ε then add x to resultSet;
    return resultSet;
end.

```

Za pomocą powyższych pseudokodów przedstawione zostały funkcje modyfikowane w stosunku do oryginalnego DBSCAN. Podstawową modyfikacją głównej pętli algorytmu jest wykorzystanie funkcji *TI-Neighbourhood*, zwracającej epsilonowe sąsiedztwo zadaneego punktu. Jej wynikiem jest suma teoriomnogościowa zbiorów zwracanych przez wywoływane po sobie funkcje *TI-Backward-Neighbourhood* i *TI-Forward-Neighbourhood*, które wyszukują punkty epsilonowego sąsiedztwa,

3. WYKORZYSTANIE NIERÓWNOŚCI TRÓJKĄTA

znajdujące się w indeksie odpowiednio: przed i po zadanym punkcie. Iteruję one indeks w tył i przód tylko do momentu napotkania punktów, dla których różnica wcześniejszych obliczonych odległości referencyjnych pomiędzy nimi i badanym punktem jest większa od wartości ε . Zgodnie z teorią przedstawioną w rozdziale 3, sprawdzanie dalszych punktów jest zbędne, ponieważ nie mogą one należeć do epsilonowego sąsiedztwa badanego punktu.

```
Function TIRef-preparation(D, refs) :
    Select refs reference points refsSet;
    for each p ∈ D
        set p as unclassified;
        for each i = 1..refs
            calculate p.refDistance(i) as distance(p, refsSet(i));
        p.neighboursNo = 1;
        p.border = Ø;
    sort D according to refDistance(1);
end.

Function TINeighbourhoodRef(p, D, ε, refs): set of points
    return union of TIRef-BackwardNeighbourhood(p, D, ε, refs)
        and TIRef-ForwardNeighbourhood(p, D, ε, refs)
end.

Function TIRef-BackwardNeighbourhood (p, D, ε, refs): set of points
    backwardThreshold = p.refDistance(1) - ε;
    iterate backward D starting with x = point before p
        if x.refDistance(1) < backwardThreshold then break;
        for each i = 2..refs
            if |x.refDistance(i)-p.refDistance(i)| > ε then
                skip to next point x;
            if distance(p, x) ≤ ε then add x to resultSet;
    return resultSet;
end.

Function TIRef-ForwardNeighbourhood (p, D, ε, refs): set of points
    forwardThreshold = p.refDistance(1) + ε;
    iterate forward D starting with x = point after p
        if x.refDistance(1) > forwardThreshold then break;
        for each i = 2..refs
            if |x.refDistance(i)-p.refDistance(i)| > ε then
                skip to next point x;
            if distance(p, x) ≤ ε then add x to resultSet;
    return resultSet;
end.
```

Powyższy pseudokod, z zaznaczonymi kolorem żółtym zmianami w stosunku do kodów przytoczonych na poprzedniej stronie, przedstawia funkcje przedstawionej w tym samym artykule [14] wersji algorytmu TI-DBSCAN-Ref, wykorzystującej wiele punktów referencyjnych, a nie tylko jeden z nich, jak w podstawowej wersji. Pod-

czas inicjalizacji algorytmu obliczane są w nim także odległości do wcześniejszej wybranych, dodatkowych punktów referencyjnych. Są one po kolei badane tylko w przypadku spełnienia warunku nierówności trójkąta dla wszystkich poprzednich punktów referencyjnych, a w szczególności pierwszego punktu, który decyduje o kolejności punktów w indeksie.

W zaprezentowanym algorytmie TI-DBSCAN poza modyfikacją związaną z nierównością trójkąta, autorzy zastosowali także dodatkową modyfikację bazującą na przedstawionym w [15] pomyśle usuwania ze zbioru D punktów, które okazały się punktami rdzeniowymi. Rozwiążanie zostało jednak dodatkowo zmodyfikowane w [14]. Definicja punku została rozszerzona o pola przechowujące liczbę znalezionych do danego momentu punktów sąsiednich oraz zbioru potencjalnych punktów brzegowych znalezionych później grup. Dzięki temu z przeszukiwanego zbioru danych D mogą usuwane być nie tylko punkty rdzeniowe, ale wszystkie zbadane do danego momentu punkty, niezależnie od tego czy okazały się rdzeniowe, czy też nie.

Główna pętla algorytmu iteruje nie po całym zbiorze danych, a jedynie po zbiorze punktów, które nie zostały wcześniej zbadane. Dodatkowy warunek w pętli umożliwia jedynie ewentualne pomijanie sprawdzeń, czy punkt został już przypisany do jakiejś grupy lub oznaczony jako szum. Największa korzyść z operowania na ograniczonym zbiorze punktów wcześniej niebadanych występuje w przypadku funkcji wyszukiwania sąsiedztwa, gdzie uniknąć można wielu niepotrzebnych obliczeń rzeczywistych odległości pomiędzy punktami. Jeśli punkt był już wcześniej badany, oznacza to, że zostały obliczone wszystkie rzeczywiste odległości pomiędzy nim, a punktami, których nie można wykluczyć z epsilonowego sąsiedztwa danego punktu i znalezieni zostali wszyscy sąsiadzi. Skoro wyznaczanie odległości pomiędzy punktami jest operacją symetryczną, to punkt nie może okazać się sąsiadem punktu, nienależącego do wcześniej wyznaczonego sąsiedztwa. Mechanizm zapamiętywania informacji o usuwanym ze zbioru sąsiedztwie gwarantuje, że kosztowna operacja obliczania rzeczywistej odległości pomiędzy dowolnymi dwoma punktami nie będzie przeprowadzana więcej niż jeden raz. W zamian ponoszony jest koszt zwiększonego zapotrzebowania na pamięć oraz skomplikowania algorytmu. Podczas sprawdzania warunku na minimalną liczbę sąsiadów do długości listy aktualnie znalezionych w epsilonowym otoczeniu punktów dodawana jest zapamięta liczba sąsiadów danego punktu

3. WYKORZYSTANIE NIERÓWNOŚCI TRÓJKĄTA

znalezionych w poprzednich iteracjach. W związku z tym, niezależnie od tego czy badany punkt jest rdzeniowy i tworzy grupę, wszystkim punktom z jego epsilonowego sąsiedztwa inkrementowany jest licznik sąsiadów. Dodatkowo, jeśli badany punkt okazuje się punktem rdzeniowym, odpowiedni numer grupy przypisywany jest nie tylko jemu i aktualnie znalezionym sąsiadom, ale także zapamiętanym wcześniej nierdzeniowym sąsiadom danego punktu, które stają się punktami brzegowymi. Po zakończeniu analizy dowolnego punktu, niezależnie od tego czy wybrany został on w głównej pętli jako pierwszy niesprawdzony punkt, czy też trafił na listę ziaren do sprawdzenia jako sąsiad punktu rdzeniowego, jest on usuwany z indeksu i przenoszony do zbioru punktów przeanalizowanych, a jego zmienne są czyszczone. Uwzględniające powyższe modyfikacje funkcje ilustruje zamieszczony na kolejnej stronie pseudokod.

Algorytm TI-DBSCAN-Ref wygląda analogicznie, z dokładnością do funkcji *TI-preparation* i *TI-Neighbourhood*, które zastąpione muszą być odpowiednikami *TIRef-preparation* i *TIRef-Neighbourhood*, zgodnymi z zamieszczonymi wcześniej pseudokodami. By mogły one zostać poprawnie wywołane, przekazywany jest do nich dodatkowy parametr wejściowy, określający liczbę punktów referencyjnych.

```

Algorithm TI-DBSCAN (D,  $\epsilon$ ,  $\mu$ ):
    TI-preparation(D);
    generate new clusterID;
    for each p  $\in$  D
        if TI-expandCluster(clusterID, p, D, D',  $\epsilon$ ,  $\mu$ ) then
            generate new clusterID;
    return D';
end.

Function TI-expandCluster(clusterID, p, D, D',  $\epsilon$ ,  $\mu$ ): Boolean
    insert into queue  $\Phi$  all q  $\in$  TI-Neighbourhood(p, D,  $\epsilon$ );
    p.neighboursNo = p.neighboursNo + | $\Phi$ |;
    if p.neighboursNo <  $\mu$  then
        mark p as noise;
        for each point q  $\in$   $\Phi$ 
            add p to q.border;
            q.neighboursNo = q.neighboursNo + 1;
        clear p.border;
        move p from D to D';
        return false;
    else
        assign current clusterID to p;
        assign current clusterID to all q  $\in$   $\Phi$ ;
        assign current clusterID to all b  $\in$  p.border;
        for each point q  $\in$   $\Phi$ 
            q.neighboursNo = q.neighboursNo + 1;
        clear p.border;
        move p from D to D';
        while  $\Phi \neq \emptyset$  do
            q = first point in  $\Phi$ ;
            insert into queue  $\Phi'$  all x  $\in$  TI-Neighbourhood(q, D,  $\epsilon$ );
            q.neighboursNo = q.neighboursNo + | $\Phi'$ |;
            if q.neighboursNo <  $\mu$  then
                for each point x  $\in$   $\Phi'$ 
                    x.neighboursNo = x.neighboursNo + 1;
            else
                for each point x  $\in$   $\Phi'$ 
                    x.neighboursNo = x.neighboursNo + 1;
                    if x is unclassified then
                        assign current clusterID to x;
                        add x to  $\Phi$ ;
                    remove x from  $\Phi'$ ;
                assign current clusterID to all b  $\in$  q.border;
                clear q.border;
                move q from D to D';
                remove q from  $\Phi$ ;
            return true;
    end.

```

3.2.2. Algorytm TI-NBC

Autorzy opisanego w poprzednim podrozdziale algorytmu TI-DBSCAN zaproponowali wykorzystanie nierówności trójkąta także w algorytmie TI-NBC [23], czyli zmodyfikowanej wersji NBC. O ile w przypadku TI-DBSCAN zadanie znajdowania epsilonowego sąsiedztwa punktu dla zadanej wartości epsilon odbywało się z intuicyjnym wykorzystaniem nierówności trójkąta, o tyle w przypadku TI-NBC zadanie jest bardziej skomplikowane. Na wejściu wartość epsilon nie jest znana, a wyznaczyć trzeba ją niezależnie dla każdego punktu na podstawie odległości k-tego najbliższego sąsiada, gdzie k jest wejściowym parametrem algorytmu.

Algorytm, analogicznie jak TI-DBSCAN, rozpoczyna się od obliczenia odległości każdego punktu od wcześniej wybranego punktu referencyjnego i posortowania ich według niemalejących wartości tych odległości. Główna pętla algorytmu nie ulega modyfikacjom, ale przyspieszana jest wywoływana wcześniej dla wszystkich punktów procedura obliczania wartości współczynnika lokalnej gęstości NDF. W jej ramach, dla każdego punktu wywoywana jest funkcja *TI-k-Neighbourhood* zwracająca jego k-sąsiedztwo. Samą procedurę znajdowania k-sąsiedztwa podzielić można na dwa podstawowe kroki. Algorytm zaczyna od znalezienia zbioru kandydatów, czyli k punktów o najmniejszej różnicy odległości do punktu referencyjnego, które zapamiętywane są w kolejności niemalejącej rzeczywistej odległości do badanego punktu. Pozwala to w mało kosztowny sposób oszacować odległość k-tego kandydata, czyli minimalną wartość epsilon, która gwarantuje znalezienie k sąsiadów. Rzeczywiste k-sąsiedztwo musi znajdować się w promieniu tak oszacowanej wartości epsilon lub mniejszym od badanego punktu. Dlatego drugim krokiem algorytmu jest weryfikacja wyznaczonego zbioru kandydatów, podczas której badane są kolejne punkty ze zbioru danych, niesprawdzone w pierwszym kroku poszukiwania kandydatów. Zbiór danych iterowany jest w przód i tył, aż do napotkania punktów, w przypadku których różnica odległości badanego punktu i ich odległości do punktu referencyjnego jest większa niż aktualny epsilon. W celu przyspieszenia poszukiwań i dodatkowego ograniczenia liczby punktów, dla których obliczana jest rzeczywista odległość, wartość epsilon aktualizowana jest za każdym razem, gdy znajdowany jest nowy sąsiad. Jego wstawienie do posortowanego zbioru kandydatów na k-sąsiedztwo może spowodować automatyczne usunięcie najdalszych kandydatów, o ile nie występuje sytuacja,

gdy wiele punktów znajduje się w odległości takiej samej jak k-ty sąsiad. Opisane powyżej algorytmy można przedstawić za pomocą pseudokodu:

```

Function TI-preparation(D) :
    select reference point r;
    for each p ∈ D
        set p as unclassified;
        clear p parameters p.kNB, p.rkNBno;
        calculate p.refDistance as distance(p, r);
        sort D according to refDistance;
    end.

Function TI-k-Neighbourhood(p, D, k) : set of points
    b = p;
    f = f;
    backSearch = precedingPoint(D, b);
    forSerach = followingPoint(D, f);
    kNB = findCandidateNeighbours(D, p, b, f, backSearch, forSearch, k);
    verifyCandidateNeighboursBackward(kNB, D, p, b, backSearch, k);
    verifyCandidateNeighboursForward(kNB, D, p, f, forSearch, k);
    return kNB;
end.

Function findCandidateNeighbours(D, p, b, f, backSearch, forSearch,
k) : set of pairs {distance, point}
    prepare kNB = empty ordered set of pairs {distance, point};
    while backSearch & forSearch & |kNB|<k
        if (p.refDistance-b.refDistance) ≤ (f.refDistance-p.refDistance)
            insert into kNB pair {distance(p,b), b};
            backSearch = precedingPoint(D, b);
        else
            insert into kNB pair {distance(p,f), f};
            forSearch = followingPoint(D, f);
        while backSearch & |kNB|<k
            insert into kNB pair {distance(p,b), b};
            backSearch = precedingPoint(D, b);
        while forSerach & |kNB|<k
            insert into kNB pair {distance(p,f), f};
            forSerach = followingPoint(D, f);
        end.

Function verifyCandidateNeighboursBackward(kNB, D, p, b, backSearch,
k) :
    ε = distance of last element of kNB;
    while backSearch & (p.refDistance-b.refDistance ≤ ε)
        distancePB = distance(p,b);
        if distancePB < ε then
            if |{e ∈ kNB | e.distancePB < ε }| ≥ k - 1 then
                delete all x ∈ {e ∈ kNB | e.distancePB = ε };
                insert into kNB pair {distancePB, b};
                ε = distance of last element of kNB;
            else
                insert into kNB pair {distancePB, b};
            elseif distancePB = ε then insert into kNB pair {distancePB, b};
            backSearch = precedingPoint(D, b);
        end.

```

3. WYKORZYSTANIE NIERÓWNOŚCI TRÓJKĄTA

```
Function verifyCandidateNeighboursForward(kNB, D, p, f, forSearch, k):
    ε = distance of last element of kNB;
    while forSearch & (f.refDistance-p.refDistance ≤ ε)
        distanceFP = distance(f,p);
        if distanceFP < ε then
            if |{e ∈ kNB | e.distanceFP < ε }| ≥ k - 1 then
                delete all x ∈ {e ∈ kNB | e.distanceFP = ε };
                insert into kNB pair {distanceFP, f};
                ε = distance of last element of kNB;
            else
                insert into kNB pair {distanceFP, f};
            elseif distanceFP = eps then
                insert into kNB pair {distanceFP, f};
            forSearch = followingPoint(D, f);
        end.

Function PrecedingPoint(D, p):
    if there is a point in D preceding p then
        p = point immediately preceding p in D;
        return true;
    else
        return false;
    end.

Function FollowingPoint(D, p):
    if there is a point in D following p then
        p = point immediately following p in D;
        return true;
    else
        return false;
    end.
```

Analogiczne jak w przypadku opisanego w poprzednim rozdziale algorytmu TI-DBSCAN, niewielkie modyfikacje umożliwiają zastosowanie wariantu z wieloma punktami referencyjnymi. Przyspieszają one funkcje weryfikacji kandydatów na k sąsiedztwo, ponieważ w trakcie wyznaczania zbioru kandydatów, nie znając nawet szacunkowej wartości epsilon, nie możemy zaoszczędzić warunku wykluczenia kandydatów z tego zbioru. Dodatkowy warunek spełnienia nierówności trójkąta dla wszystkich punktów referencyjnych dodany został w funkcjach weryfikacji kandydatów (*verifyCandidateNeighboursBackward* i *verifyCandidateNeighboursForward*), dzięki czemu ograniczana jest liczba punktów, dla których obliczana musi być rzeczywista odległość. Pseudokod z zaznaczonymi zmianami (analogicznie modyfikowana jest symetryczna funkcja *verifyCandidateNeighboursBackward*) to:

```

Function verifyCandidateNeighboursForward(kNB, D, p, f, forSearch, k,
refs):
    ε = distance of last element of kNB;
    while forSearch & (f.refDistance(1)-p.refDistance(1) ≤ ε)
        ok = true
        for each i = 2..refs
            if |x.refDistance(i)-p.refDistance(i)| > ε then
                ok = false; break;
        if ok then
            distanceFP = distance(f,p);
            if distanceFP < ε then
                if |{e ∈ kNB| e.distanceFP < ε }| ≥ k - 1 then
                    delete all x ∈ {e ∈ kNB | e.distanceFP = ε };
                    insert into kNB pair {distanceFP, f};
                    ε = distance of last element of kNB;
                else
                    insert into kNB pair {distanceFP, f};
                elseif distanceFP = ε then
                    insert into kNB pair {distanceFP, f};
            forSerach = followingPoint(D, f);
        end.
    
```

3.2.3. Algorytm TI-PreDeCon

Rezultaty otrzymane przez autorów opublikowanych algorytmów TI-DBSCAN i TI-NBC zachęcają do prób modyfikacji wykorzystującej nierówność trójkąta także innych algorytmów gęstościowego grupowania danych, jak choćby algorytmów grupowania w podprzestrzeniach.

W przypadku analizowanego w rozdziale 2.3 algorytmu PreDeCon, optymalizacja z wykorzystaniem nierówności trójkąta w wersji analogicznej jak w TI-DBSCAN z powodzeniem zastosowana może być w pierwszej części algorytmu, w której, w celu wyznaczenia wektorów preferencji podprzestrzeni, na początku wyznaczone muszą być standardowe epsilonowe otoczenia wszystkich punktów. Gdyby w drugiej części wykorzystywana była globalna podprzestrzeń z globalnym wektorem preferencji uwzględnianym we wzorze na odległość, z pewnością wartym uwagi rozwiązań byłoby zaktualizowanie odległości referencyjnych i kolejności obiektów w zbiorze danych. Zastosowane w PreDeCon rozwiązanie z niezależnymi wektorami preferencji dla każdego punktu uniemożliwia wykorzystanie przedstawionej modyfikacji, tak samo jak opisywanych indeksów przestrzennych, takich jak R*-drzewo czy VA-File.

Jak pokazują wyniki testów z rozdziału 5.5 TI-PreDeCon, czyli zmodyfikowana wersja algorytmu PreDeCon, wykorzystująca, opisaną w rozdziale 3.2.1, funkcję

3. WYKORZYSTANIE NIERÓWNOŚCI TRÓJKĄTA

TI-Neighbourhood, dużo wydajniej wyznacza wektor preferencji. Warto jednak zwrócić uwagę, że wzrost wydajności całego algorytmu w porównaniu do wyników otrzymywanych dla TI-DBSCAN czy TI-NBC jest zdecydowanie mniejszy. Wynika to z kosztownej operacji wyznaczania sąsiedztwa uwzględniającego wektor preferencji, która nie została usprawniona.

4. SZCZEGÓŁY IMPLEMENTACJI

W ramach przeprowadzonej analizy i badania algorytmów gęstościowego grupowania danych zaimplementowałem opisane w rozdziale 2 algorytmy: DBSCAN, NBC i PreDeCon. Oprócz podstawowych wersji używających indeksów R*-drzewo oraz VA-File zaimplementowałem także usprawnione z wykorzystaniem nierówności trójkąta wersje z rozdziału 3.2. Najistotniejsze założenia i szczegóły implementacji przedstawiłem w poniższych podrozdziałach.

4.1. ZAŁOŻENIA I OGRANICZENIA

Głównym celem implementacji była analiza i testy opisywanych algorytmów, a w szczególności możliwości ich usprawnienia opartego na nierówności trójkąta. Miało to wpływ na architekturę całego rozwiązania, w którym wykorzystana została rozbudowana hierarchia dziedziczenia. Zaletą wykorzystania wspólnych dla wielu algorytmów klas indeksów, a także dziedziczenia pozwalającego na definiowanie tylko modyfikowanych metod czy atrybutów, jest ujednolicenie wszystkich wspólnych funkcjonalności i procesów. Nie tylko ułatwia to rozwój i modyfikację poszczególnych funkcji, ale przede wszystkim zmniejsza ryzyko wpływu różnic implementacyjnych na testowaną wydajność algorytmów.

Przedstawiony cel implementacji wiąże się także z koniecznością rejestracji jak największej liczby pomiarów, w tym dokładnych czasów trwania wszystkich znaczących operacji oraz liczby obliczanych odległości czy sprawdzanych warunków nierówności wykluczające z epsilonowego sąsiedztwa. Uproszczeniem, o którym wspomniałem już we wcześniejszych rozdziałach, jest możliwość przypisania danego obiektu do maksymalnie jednej grupy, mimo iż w przypadku algorytmów DBSCAN i dNBC obiekty mogą należeć do wielu grup jednocześnie. Ogranicza to możliwość rzeczywistego wykorzystania omawianych implementacji, lecz nie ma znaczącego

4. SZCZEGÓŁY IMPLEMENTACJI

wpływ na wydajność algorytmów, a z racji przyjęcia takiego uproszczenia także w literaturze ([7], [14], [23]) zwiększa to porównywalność wyników. W zależności od zbioru danych wybrać można implementację operującą na wartościach zmiennoprzecinkowych lub całkowitoliczbowych, które okazały się wystarczające do badań eksperymentalnych na najpopularniejszych testowych zbiorach danych.

Jako język programowania wybrany został C++, który łączy zalety obiektowości z dużą wydajnością, a przede wszystkim z kontrolą wykonywania procesu na wystarczająco niskim poziomie. W przeciwieństwie do drugiego, popularnego w literaturze dotyczącej algorytmów gęstościowego grupowania danych, języka Java, w C++ nie występują m.in. niekontrolowane sytuacje czyszczenia pamięci (ang. *garbage collector*). W przypadku algorytmów operujących na dużych ilościach danych zdarzenia takie są mocno prawdopodobne, a są one zdecydowanie niepożądane w przypadku testów pomiaru czasów wykonywania poszczególnych operacji, ponieważ mogą powodować losowe zakłócenia testów.

4.2. IMPLEMENTACJA INDEKSÓW

4.2.1. Implementacja R*-drzewa

Za wyborem C++ stały także dostępne w tym języku implementacje R*-drzewa. Wykorzystana została implementacja z udostępnionej na licencji GNU biblioteki *SpatialIndex Library* [8] (w najaktualniejszej na dzień wyboru wersji 1.6.1). Musiała być ona rozszerzona o odpowiednio przystosowane do potrzeb zapytań o epsilonowe sąsiedztwo: definiującą kształt regionu i strategię poszukiwania klasę *Epsilon* i zarządzającą przekazywanie wyniku klasę *IdVisitor*. O ile *IdVisitor* jedynie zapamiętuje identyfikator (indeks) punktów należących do epsilonowego sąsiedztwa, o tyle dziedzicząca po *IShape* klasa *Epsilon* zapewnić musi metody pozwalające odpowiednio przejść przez R*-drzewo w trakcie poszukiwań epsilonowego sąsiedztwa. Kształt epsilonowego sąsiedztwa reprezentowany jest przez punkt środkowy i promień, jednak podczas przechodzenia przez drzewo stosowane jest przybliżenie w postaci hipersześcianu, co oznacza, że przybliżeniem koła w dwóch wymiarach jest kwadrat,

a w przypadku trzech wymiarów przybliżeniem kuli jest sześcian. Dany węzeł drzewa wykluczany jest z dalszych poszukiwań tylko, jeśli reprezentujący go hipersześcian nie przecina się z hipersześcianem przybliżającym region zapytania, co pozwala wnioskować, że nie zawiera on punktów należących do odpowiedzi. Rzeczywisty kształt espsilonowego sąsiedztwa wykorzystywany jest dopiero w na poziomie procesowania liści reprezentujących pojedyncze punkty, dla których liczona i porównywana z epsilonem jest rzeczywista odległość od badanego punktu zgodnie z pseudokodem:

```

class Epsilon : public SpatialIndex::IShape {
    centerPoint; eps; dimensions;
    lowLimits[dimensions]; highLimits[dimensions];

    constructor Epsilon(p, ε) {
        center = p;
        eps = ε;
        dimensions = point.dimensions();
        for i=0...dimensions do
            lowLimits[i] = point.attribute[i] - eps;
            highLimits[i] = point.attribute[i] + eps;
    }

    bool intersectRegion(region) {
        for i=0...dimensions
            if (lowLimits[i] > region.highLimits[i] ) return false;
            if (highLimits[i] < region.lowLimits[i] ) return false;
        return true;
    }

    bool containsPoint(point) {
        if distance(point, center) < eps then return true;
        else return false;
    }
}

```

4.2.2. Implementacja VA-File

Strukturę VA-File zaimplementowałem samodzielnie na podstawie definicji i pseudokodów z artykułu publikującego indeks [21]. Żaden z zaproponowanych w artykule algorytmów wyszukiwania sąsiedztwa punktu nie odpowiadał w pełni charakterystyce, na jaką powoływali się autorzy NBC, dlatego wybrałem najbardziej uniwersalny algorytm VA-SSA (ang. *Simple Search Algorithm*). Wprowadzoną modyfikacją jest funkcja podziału bitów długości wektora proporcjonalnie do dziedzin poszczególnych atrybutów. W wektorze VA-File atrybuty o szerokich dziedzinach war-

4. SZCZEGÓŁY IMPLEMENTACJI

tości przybliżane są za pomocą proporcjonalnie większej liczby bitów niż atrybuty o wąskich dziedzinach, zgodnie z przedstawionym poniżej pseudokodem:

```
Function calculateBitsProportionally(domains vector, allBits) : bits
vector
    bits[domains] = empty vector;
    sort list Φ of pairs {domainRange, domainId} according to
        domainRange;
    sum = sum of domainRange of all domains;
    while (Φ ≠ Ø & sum>0 & allBits>0) do
        f = first pair in Φ;
        proportion = f.domainRange / sum;
        sum = sum - f.domainRange;
        maxBits = min of allBits and number of bits of f.domainRange in
            binary format;
        bits[f.domainId] = min of floor(allBits*proportion+0.5) and
            maxBits;
        allBits = allBits - bits[f.domainId];
        remove f from Φ;
    end.
```

4.3. IMPLEMENTACJA ZOPTYMALIZOWANYCH ALGORYTMÓW

Zgodnie z wcześniej przedstawionymi wnioskami, główne korzyści z wykorzystania nierówności trójkąta występują przy założeniu, że punkty zbioru danych posortowane są względem odległości od punktu referencyjnego. Samo sortowanie, jako jedna z najczęściej wykonywanych operacji na danych, należy do grupy dobrze zbadanych problemów informatyki. Wciąż jest to jednak obciążenie dla algorytmu, w szczególności w przypadku dużych zbiorów danych, dla których stworzone zostały omawiane algorytmy grupowania danych. Często stosowanym w przypadku dużych wolumenów danych, alternatywnym rozwiązaniem pozwalającym uniknąć sortowania oryginalnego zbioru danych, jest utworzenie indeksu. Jest to dodatkowa struktura, najczęściej w postaci drzewa, zawierająca jedynie posortowane wskaźniki do ułożonych w dowolnej kolejności punktów. Takie rozwiązanie zastosowane zostało w przypadku R*-drzewa, jak i VA-File. W przypadku TI-DBSCAN atutem dodatkowej struktury zawierającej wskaźniki na oryginalne elementy byłaby możliwość dokładnej implementacji mechanizmu polegającego na fizycznym usuwaniu zbadanych punktów z przeszukiwanego zbioru danych. W przypadku sortowania oryginalnego zbioru da-

nych usuwanie punktów z wektora byłoby zbyt kosztowne, ponieważ wiązałoby się z przepisaniem kolejnych elementów wektora. Sortowanie zbioru danych, zapewniające, że dane punktów podobnie oddalonych od punktu referencyjnego leżą blisko siebie w pamięci, pozwala jednak zdecydowanie zmniejszyć liczbę odczytów pamięci potrzebnych do znalezienia epsilonowego sąsiedztwa dowolnego punktu.

Ostatecznie zaimplementowałem dwie wersje TI-DBSCAN. Jedna wykorzystuje dodatkową listę uporządkowanych wskaźników, z której usuwane są przebadane punkty. Druga jest oparta na sortowaniu zbioru danych i dodatkowej fladze informującej o tym, że dany punkt był już badany. Iterowanie wszystkich, ale za to zapisanych sekwencyjnie, elementów zbioru danych okazuje się jednak zdecydowanie wydajniejsze niż w przypadku iterowania po liście. Połączone wskaźnikami elementy listy mogą znajdować się w różnych obszarach pamięci, a co więcej są to tylko wskaźniki na punkty, rozrzucone losowo w pamięci zajmowanej przez zbiór danych. Oznacza to nie tylko większą liczbę odczytów, ale także zmniejsza szansę, czytania z sektorów znajdujących się akurat w pamięci operacyjnej. Potwierdzają to wyniki testów przedstawione w rozdziale 5.3.1.

4.4. IMPLEMENTACJA INTERFEJSU UŻYTKOWNIKA

Interfejsem do wszystkich algorytmów jest klasa *AbstractClusteringAlgorithm*. Główna funkcja run, jako atrybuty przyjmuje zbiór danych oraz mapę parametrów. Dla ułatwienia obsługi wszystkie algorytmy mogą być wywoływane z wygodnego interfejsu graficznego, zaimplementowanego w przenośnej bibliotece Qt [17]. GUI pozwala na wskazanie danych testowych i ich ograniczeń oraz wybór algorytmu i wszystkich jego parametrów. Dodatkowo, po zakończeniu grupowania, wyświetlane są statystyki działania algorytmu, takie jak wielkości znalezionych grup oraz czasu trwania poszczególnych operacji. Można także generować wizualizację wyniku grupowania w dwóch dowolnie wybranych wymiarach, prezentującą punkty z poszczególnych grup w innych kolorach. Jeśli istnieje taka potrzeba, można także zapisać wynik do pliku wyjściowego.

5. BADANIA EKSPERYMENTALNE

W badaniach eksperymentalnych sprawdziłem praktyczny wzrost wydajności wyszukiwania epsilonowego sąsiedztwa oraz k-sąsiedztwa z zastosowaniem usprawnienia opartego na nierówności trójkąta, w tym wpływ jego implementacji oraz strategii wyboru punktów referencyjnych. Algorytmy i indeksy zaimplementowałem na podstawie pseudokodów z artykułów [3], [5], [7], [13], [14], [21], [23] w języku C++, zgodnie z opisem z rozdziału 4. Wszystkie testy dla porównywalności przeprowadziłem na tym samym, przeznaczonym tylko dla nich, komputerze wyposażonym w procesor Intel Core Duo T2500 2GHz i 2 GB pamięci operacyjnej RAM. Testy uruchamiane jako proces o podwyższonym priorytecie mogły w pełni wykorzystać jeden rdzeń, co zmniejsza ryzyko losowych zakłóceń wyników.

5.1. DANE TESTOWE

Do testów wykorzystałem szeroko stosowane w dziedzinowej literaturze testowe zbiory danych, które dzięki odmiennym charakterystykom pozwalają sprawdzić działanie algorytmu w różnych przypadkach. Przedstawione poniżej statystyki dotyczą podzbiorów 60 tysięcy pierwszych rekordów, na których skupiłem się podczas testów i na których oparte zostały zaprezentowane w kolejnych podrozdziałach wnioski.

KDD-Cup98 [18] to powszechnie stosowany testowy zbiór danych opublikowany w międzynarodowym konkursie zorganizowanym podczas KDD-98, czwartej Międzynarodowej Konferencji Eksploracji Danych i Odkrywania Wiedzy w 1998 roku. Zbiór zawiera 96367 rekordów o 56 atrybutach z całkowitoliczbowymi, nieujemnymi wartościami. Aż 44 atrybuty posiadają taką samą dziedzinę [0, 99], z czego zdecydowana większość z ich wartości skupiona jest w pierwszej połowie dziedziny (średnia to 27, a odchylenie standardowe 11). Małe odchylenie standardowe od niskiej średniej

5. BADANIA EKSPERYMENTALNE

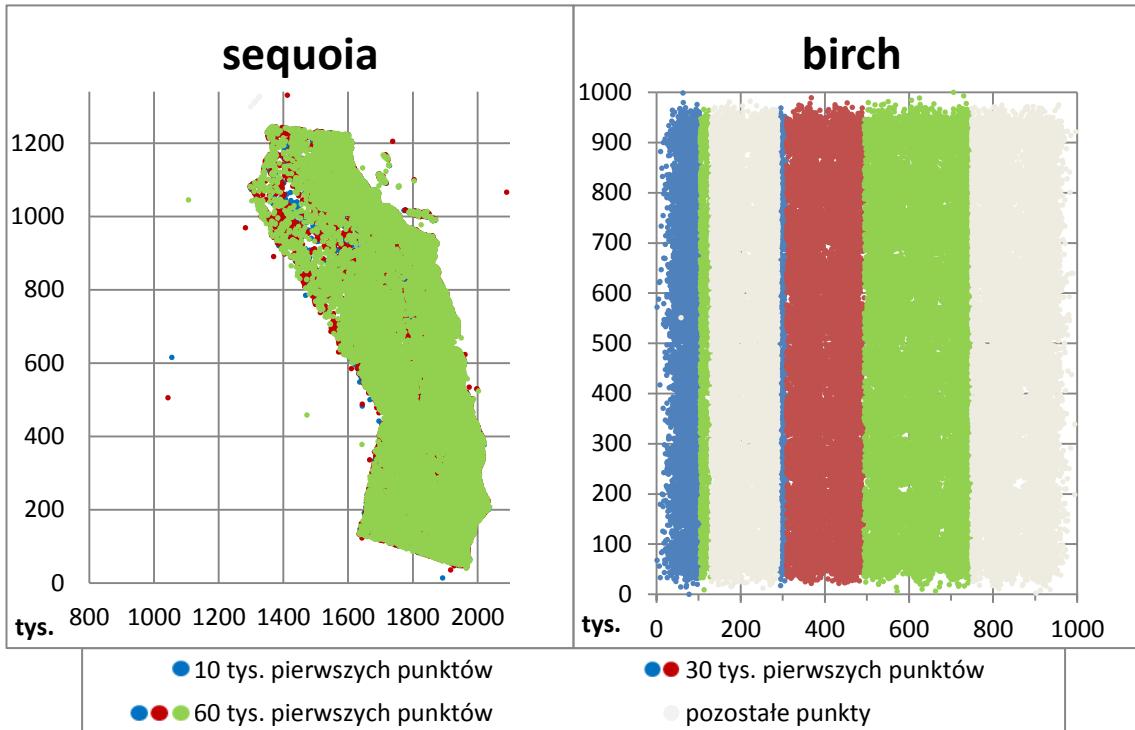
charakteryzuje także pozostałe atrybuty, w tym dwa atrybuty o najwęższej dziedzinie [0,13] i dwa o najszerzej dziedzinie [0,6000].

Covtype [4] to skrócona nazwa udostępnionego przez US Forest Service zbioru Forest CoverType, zawierającego informacje na temat gatunków drzew w amerykańskich lasach. Baza zawiera aż 581012 rekordów z 55 atrybutami, z których 10 pierwszych to wartości zmiennych ilościowych, 44 kolejnych zawiera binarne dane 0 lub 1, a ostatni określa jeden z siedmiu gatunków drzew. Warto zwrócić uwagę, na specyficzną właściwość, że każdy rekord posiada jako wartości atrybutów binarnych dokładnie dwa razy wartość 1 i 42 razy wartość 0, a rozkład wartości poszczególnych atrybutów jest bardzo nierównomierny. O ile 80% wartości pierwszego atrybutu stanowią wartości 1, o tyle aż trzy atrybuty mają same zera, a aż połowa ma ponad 99% wartości zerowych. Większe zróżnicowanie występuje w przypadku pierwszych 10 atrybutów ze zmiennymi ilościowymi, z których szczególnie interesujące są dwa o najszerzej dziedzinie (ponad 7000) i odchyleniu standardowym stanowiącym 25% zakresu. Zakresy, a przeważnie także różnorodność, pozostałych atrybutów są już mniejsze.

Sequoia [20] to zbiór danych zgromadzonych w ramach międzynarodowego projektu Sequoia 2000, mającego na celu wykorzystanie możliwości technik komputerowych przez naukowców badających Ziemię. Z petabajtów zebranych w projekcie danych, w raportach często opisywany jest zbiór 63556 rekordów ze współrzędnymi znaczących lokalizacji w Kalifornii, który po ograniczeniu do 60000 rekordów został wykorzystany w moich testach. Zgodnie z wizualizacją przedstawioną na wykresie 5.1, o ile szerokości dziedzin obydwu atrybutów uznać można za porównywalne, o tyle rozłożenie wartości poszczególnych rekordów mocno się różni. Dziedzina drugiego atrybutu (pionowa oś na wykresie) wypełniona jest dość równomiernie, podczas gdy wartości pierwszego atrybutu skupione są raczej w drugiej połowie dziedziny.

Birch [22] to zbiór danych wygenerowany na potrzeby testów hierarchicznego algorytmu o tej samej nazwie, który stał się następnie powszechnie wykorzystywany zbiorem testowym. Jak widać na wykresie 5.1 jest to najbardziej jednolity, dwuwykładowy zbiór z równomiernie rozłożonymi centrami grup. Pewne zaburzenie jednorodności zbioru wprowadza zastosowane na testach wydajnościowych ograniczenie liczby punktów. Ostatecznie, podzbiór 60 tysięcy pierwszych rekordów posiada atry-

buty o różnych dziedzinach (bez punktów zaznaczonych na wykresie kolorem szarym), z nierównomiernym rozłożeniem w ramach jednej z nich (pozioma oś na wykresie 5.1).



Wykres 5.1 Wizualizacja testowych zbiorów danych sequoia i birch z zaznaczeniem wykorzystywanych podczas testów ograniczeń na 10, 30 i 60 tysięcy punktów. Słaba widoczność podziału na podzbiory w przypadku zbioru sequoia wynika z pokrywania się równomiernie rozłożonych punktów.

5.2. PARAMETRY TESTOWE

Dla zwiększenia porównywalności uzyskiwanych wyników przyjąłem wspólne parametry wejściowe, dobrane z uwzględnieniem wskazówek z artykułów i zdefiniowane w odniesieniu do wejściowego zbioru danych. Jeśli w opisie danego testu nie sprecyzowano inaczej, przyjęte parametry wejściowe to:

- epsilon ϵ równy jednej dwudziestej, nazywanej dalej *przekątną dziedziny wejściowego zbioru danych*, odległości pomiędzy punktem p_{max} (o wartości

ściach wszystkich atrybutów równych maksimom ich dziedzin) i p_{min} (o wartościach wszystkich atrybutów równych minimom ich dziedzin);

- minimalna liczba punktów μ wynosząca jedną dwudziestą liczbę punktów wejściowego zbioru danych (nie mniej niż 1);
- liczba znaczących sąsiadów k wynosząca jedną dwudziestą liczbę punktów wejściowego zbioru danych (nie mniej niż 1);
- wartość stałej κ równa jednej czternastej przekątnej dziedziny wejściowego zbioru danych (nie mniej niż 100);
- maksymalna wariancja sąsiadów δ wynosząca jedną pięćdziesiątą przekątnej dziedziny wejściowego zbioru danych (nie mniej niż 1);
- maksymalna wymiarowość preferowanej podprzestrzeni λ równa połowie wymiarów wejściowego zbioru danych (nie mniej niż 1);

5.3. TESTY ALGORYTMU DBSCAN

DBSCAN, jako najpopularniejszy algorytm gęstościowego grupowania danych, był pierwszym testowanym algorytmem, na którym skupiłem szczególną uwagę. Przed porównaniem algorytmów TI-DBSCAN i TI-DBSCAN-Ref do wyjściowej wersji DBSCAN, przedstawione zostały wyniki testów wybranych implementacji oraz strategii wyboru punktów referencyjnych.

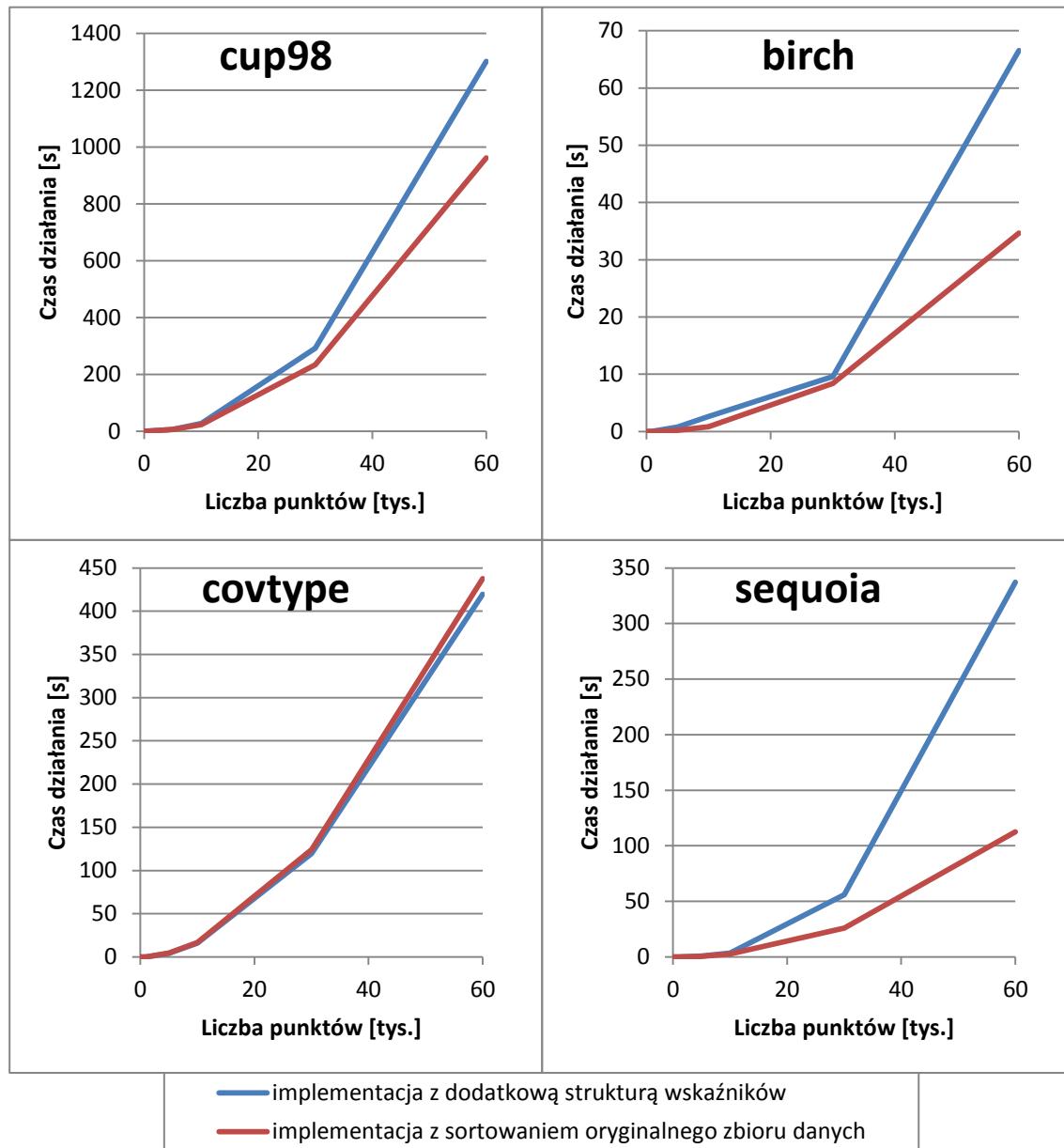
5.3.1. Implementacja TI-DBSCAN

Zgodnie z przestawionymi w rozdziale 4.1.1 rozważaniami dotyczącymi implementacji zoptymalizowanego algorytmu TI-DBCAN, na przykładowych zbiorach danych przetestowałem dwie wersje: wykorzystującą dodatkową strukturę indeksu ze wskaźnikami oraz opartą na sortowaniu oryginalnego zbioru danych. Jak pokazuje wykres 5.2, implementacja oparta na sortowaniu zbioru danych osiągała lepsze rezultaty dla większości testowych zbiorów danych. Zgodnie z bardziej szczegółowymi danymi z tabeli 5.1, sam etap przygotowywań, obejmujący obliczenie wszystkich odległości do punktu referencyjnego i w zależności od implementacji, utworzenie dodatkowej

uporządkowanej listy wskaźników lub posortowanie oryginalnego zbioru danych, trwa w przybliżeniu dwukrotnie dłużej w pierwszym przypadku. Powyższa oszczędność ma jednak pomijalny wpływ na wynik końcowy, który jest rzędu wielkości większy od opisywanych czasów przygotowywań. Mający największe znaczenie proces właściwego grupowania przebiega zdecydowanie szybciej po posortowaniu danych, a różnica wydajności jest tym większa, im większy jest analizowany zbiór danych. Wyjątkiem jest tu zbiór covtype, w którego przypadku implementacja wykorzystująca dodatkową listę wskaźników okazała się wydajniejsza nie tylko w części przygotowywania indeksu, ale także w części właściwego grupowania.

zbior	liczba punktów	TI-DBSCAN Index			TI-DBSCAN Sorting		
		przygotowania	grupowanie	łącznie	przygotowania	grupowanie	łącznie
birch	1000	0	31	31	0	0	0
	5000	0	765	765	0	188	188
	10000	0	2593	2593	31	765	796
	30000	31	9578	9609	62	8329	8391
	60000	63	66468	66531	140	34516	34656
covtype	1000	0	110	110	0	125	125
	5000	15	4188	4203	16	4469	4485
	10000	15	16016	16031	63	16797	16860
	30000	62	119625	119687	156	123953	124109
	60000	156	419578	419734	343	437422	437765
cup98	1000	0	266	266	0	250	250
	5000	0	6266	6266	31	5828	5859
	10000	16	26937	26953	47	23344	23391
	30000	62	291407	291469	172	232938	233110
	60000	172	1301170	1301340	343	961860	962203
sequoia	1000	0	32	32	0	31	31
	5000	0	844	844	0	640	640
	10000	16	3625	3641	31	2750	2781
	30000	31	55875	55906	79	25937	26016
	60000	62	336969	337031	141	112328	112469

Tabela 5.1 Porównanie wydajności implementacji algorytmu TI-DBSCAN wykorzystujących dodatkową listę wskaźników lub sortowanie zbioru danych dla przykładowych zbiorów danych. Etap przygotowań obejmuje obliczanie wszystkich odległości do punktów referencyjnych i utworzenie indeksu lub posortowanie oryginalnego zbioru danych, po czym następuje etap właściwego grupowania.



Wykres 5.2 Porównanie wydajności implementacji algorytmu TI-DBSCAN wykorzystujących dodatkową listę wskaźników lub sortowanie zbioru danych dla przykładowych zbiorów danych.

Uzasadnieniem większej wydajności implementacji wykorzystującej sortowanie jest wyjaśniona w rozdziale 4.1.1 optymalizacja odczytów pamięci w procesie wyznaczania epsilonowego sąsiedztwa dowolnego punktu. W związku z powyższymi wnioskami w dalszych testach stosowana była implementacja bez dodatkowej struktury ze wskaźnikami, lecz z bezpośrednim sortowaniem oryginalnego zbioru danych.

5.3.2. Strategie wyboru punktów referencyjnych

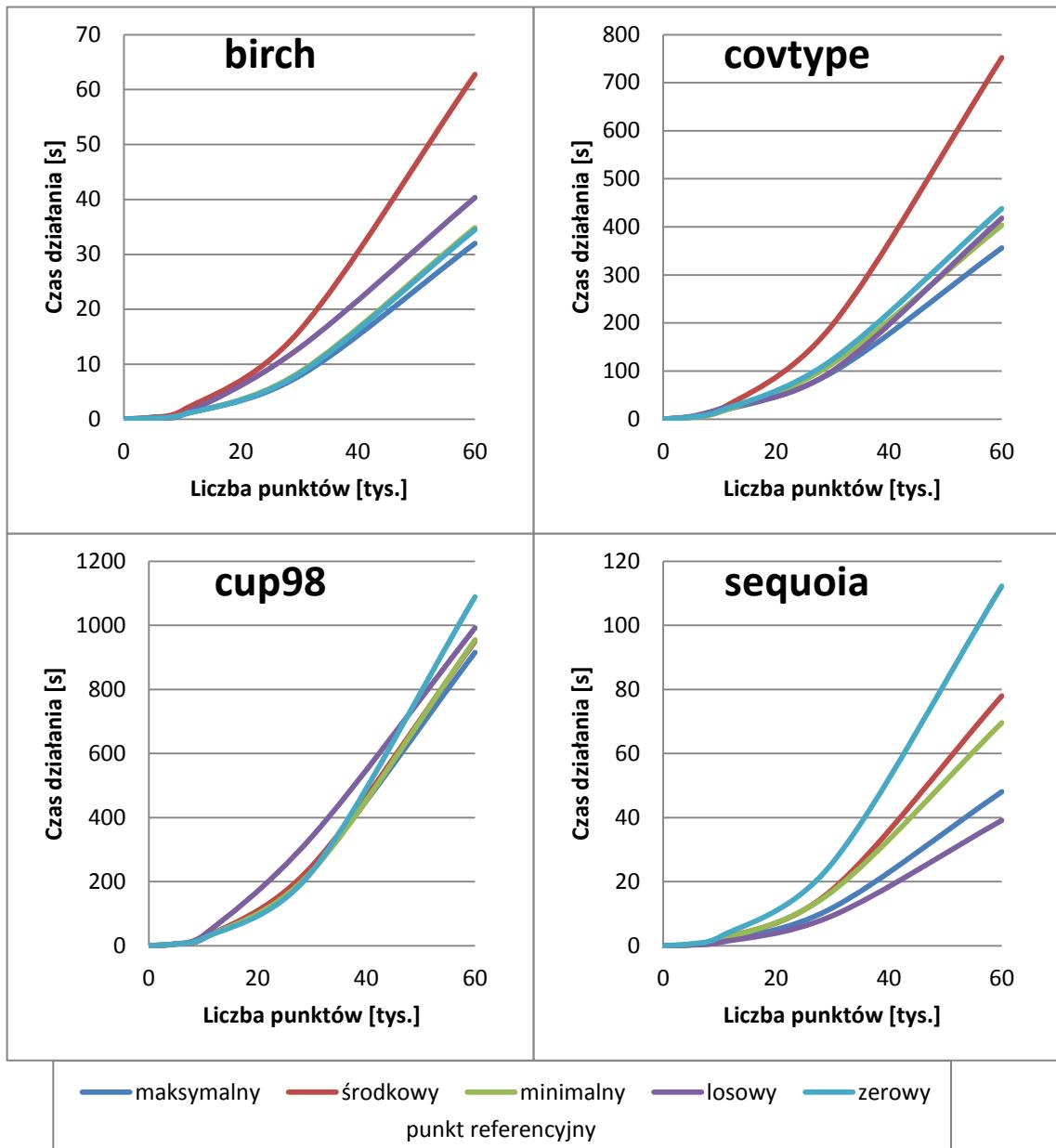
Niezależnie od konkretnej implementacji, zgodnie z opisem z rozdziału 3.1 zmodyfikowany algorytm może być parametryzowany za pomocą liczby i strategii wyboru punktów referencyjnych. Choć autorzy artykułu wprowadzającego TI-DBSCAN wspominają, że możliwe są takie modyfikacje, to w artykule [14] nie opisują ich wpływu na wydajność algorytmu. Jednym z celów niniejszej pracy było zbadanie wpływu powyższych parametrów na wydajność algorytmu. Wyniki testów i ich opis przedstawione zostały w kolejnych podrozdziałach.

5.3.2.1. Wybór jednego punktu referencyjnego

Podstawowym parametrem, który może być łatwo dobrany do zbioru danych jest strategia wyboru punktu referencyjnego, względem którego obliczane są wszystkie odległości referencyjne i sortowany jest zbiór danych. Zgodnie z opisem z rozdziału 3.1.1, możliwe strategie obejmują punkt zerowy, minimalny, środkowy i maksymalny oraz punkt losowy.

zbior	liczba punktów	strategia wyboru punktu referencyjnego				
		maksymalnego	środkowego	minimalnego	losowego	zerowego
birch	5000	172	360	203	313	187
	10000	891	1579	797	828	797
	30000	7860	16078	8532	12953	8313
	60000	32000	62765	34875	40343	34485
covtype	5000	4000	5047	3813	5766	4485
	10000	15797	19406	14781	21188	16813
	30000	97578	196250	115516	100453	124047
	60000	356281	752000	403704	418094	438015
cup98	5000	5532	5750	5875	5907	5875
	10000	21985	23047	23422	32188	23219
	30000	244875	248953	232906	340375	231359
	60000	915265	949859	954500	992032	1088860
sequoia	5000	297	407	344	234	625
	10000	1266	1969	1766	1031	2750
	30000	11828	17562	17000	9375	25875
	60000	48062	77969	69594	39125	112235

Tabela 5.2 Porównanie wpływu strategii wyboru punktu referencyjnego na wydajność algorytmu TI-DBSCAN dla przykładowych zbiorów danych.



Wykres 5.3 Porównanie wydajności strategii wyboru punktu referencyjnego w algorytmie TI-DBSCAN dla przykładowych zbiorów danych.

Zgodnie z wykresem 5.3, nie istnieje jedna najwydajniejsza strategia, gwarantująca największą wydajność niezależnie od wejściowego zbioru danych. Wymieniana w artykule [14] jako przykładowa, strategia punktu zerowego nie pozwala osiągnąć najlepszych wyników w przypadku żadnego z badanych zbiorów. Mimo to, może wydawać się to dość uniwersalny i bezpieczny wybór, ponieważ pozwala uzyskać wydajność zbliżoną do najlepszej, zarówno w przypadku wielowymiarowych zbiorów

cup98 i covtype, jak i dwuwymiarowego zbioru birch. Zdecydowanie gorzej wyglądają wyniki testów na zbiorze sequoia, dla którego punkt zerowy jest najgorszym wyborem, z powodu opisanej dalej charakterystyki tego zbioru.

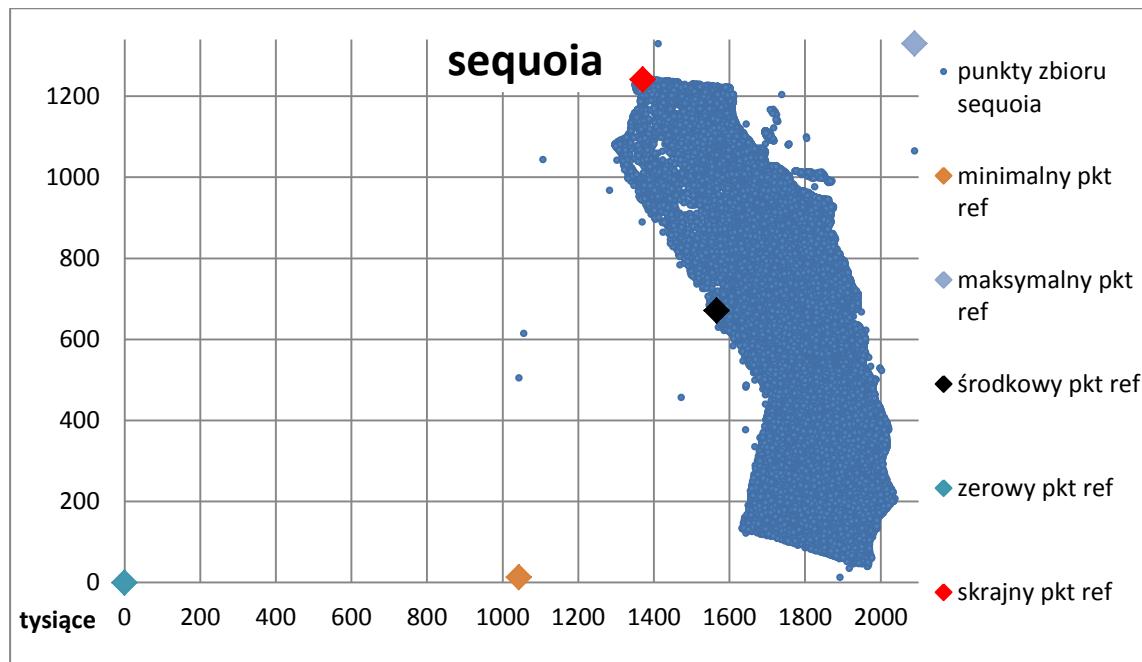
zbior	liczba punktów	maksymalny punkt ref.				minimalny punkt ref.				zerowy punkt ref.			
		przygotowa-wanie	sor-to-wa-nie	gru-pow-a-nie	łącznie	przygotowa-wanie	sor-to-wa-nie	gru-pow-a-nie	łącznie	przygotowa-wanie	sor-to-wa-nie	gru-po-wa-nie	łącznie
birch	30000	0	78	7782	7860	0	47	8469	8516	0	63	8250	8313
	60000	15	125	31860	32000	16	141	34734	34891	0	125	34360	34485
covtype	30000	47	125	97406	97578	47	109	115360	115516	47	109	123891	124047
	60000	109	266	355906	356281	110	250	403344	403704	93	250	437672	438015
cup98	30000	62	125	244688	244875	47	125	232734	232906	47	125	231187	231359
	60000	93	266	914906	915265	109	250	954141	954500	94	250	1088520	1088860
sequoia	30000	16	62	11750	11828	0	47	16938	16985	0	79	25796	25875
	60000	16	140	47906	48062	15	141	69453	69609	16	125	112094	112235

Tabela 5.3 Porównanie wpływu strategii wyboru punktu referencyjnego na wydajność poszczególnych etapów algorytmu TI-DBSCAN.

Głównym atutem wyboru punktu zerowego, mogłaby się wydawać łatwość obliczania Euklidesowej odległości od niego. W przypadku reprezentowanego w mojej implementacji przez *NULL* punktu zerowego, funkcja obliczania odległości do punktu referencyjnego pomija bowiem odejmowanie zer. Jak widać w tabeli 5.3, wynikająca z tego oszczędność czasu jest znikoma. Pierwszy etap przygotowań, obejmujący obliczanie wszystkich odległości referencyjnych jest nie tylko porównywalny niezależnie od wybranego punktu, ale co istotniejsze, jest praktycznie pomijalny w stosunku do, o rzędu wielkości dłuższego, etapu właściwego grupowania. Ten atut punktu zerowego nie powinien być więc brany pod uwagę przy wyborze strategii. W rzeczywistości wydajność zależy bowiem tylko od rozłożenia danych w stosunku do tego punktu. W przypadku równomiernie rozłożonego zbioru birch punkt zerowy pokrywa się z pewną dokładnością z punktem minimalnym, dlatego wydajność algorytmu dla tych punktów referencyjnych jest zbliżona. Natomiast w przypadku, przedstawionego

5. BADANIA EKSPERYMENTALNE

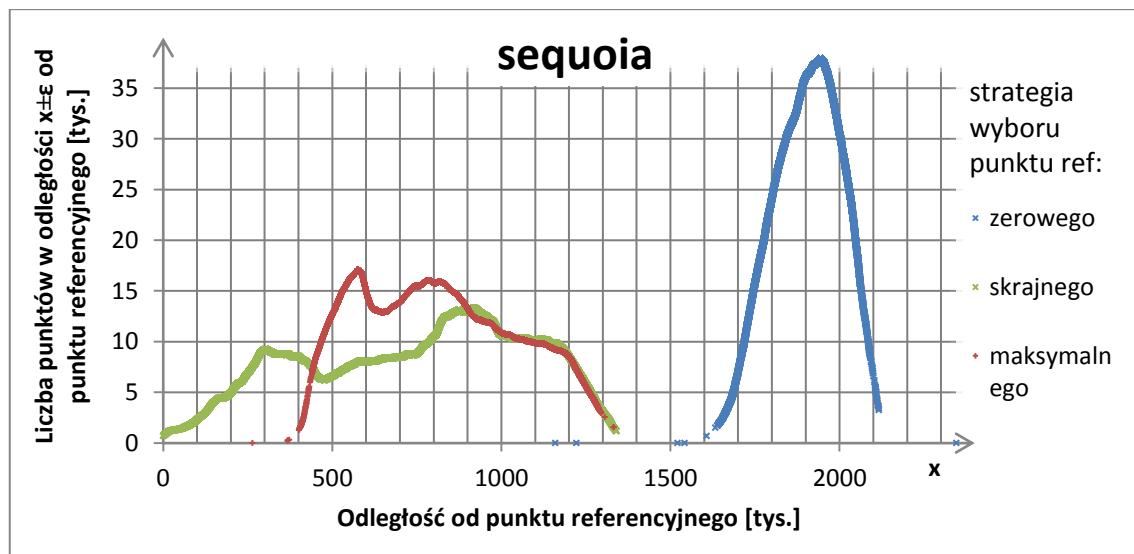
na wykresie 5.4, zbioru danych sequoia, bardzo słabe wyniki prosto można wy tłumaczyć nie tylko faktem, że punkt zerowy znajduje się daleko poza dziedzinami atrybutów, ale też punkty ułożone są w podobnej odległości od niego, co zmniejsza selektywność opartego na nim indeksu. Należy przy tym pamiętać, że wciąż nie jest to najgorszy z możliwych przypadków zbioru danych, kiedy to punkty rozłożone są na hiper-okręgu o środku w punkcie zerowym, a dokładniej w odległości nie większej niż epsilon od tak wyznaczonego okręgu.



Wykres 5.4 Prezentacja zbioru danych sequoia z zaznaczeniem pięciu charakterystycznych punktów referencyjnych (minimalnego, maksymalnego, środkowego, zerowego i skrajnego).

W przypadku zbioru sequoia najlepszym wyborem okazał się punkt losowy, który z dużym prawdopodobieństwem znajdzie się w okolicy jednego z najkorzystniejszych w tym przypadku obszarów, na jednym z końców utworzonej przez większość punktów grupy o podłużnym kształcie. Przykładowy punkt z takiego obszaru zaznaczyłem na wykresie 5.4 jako *punkt skrajny*. Jak widać na wykresie 5.5, wartości obliczanych do niego odległości do punktów wejściowego zbioru danych mają największy zakres. Dodatkowo, liczności zbiorów punktów o różnicy odległości referencyjnej nie większej niż ϵ , czyli takich, dla których konieczne jest obliczenie rzeczywistej odległości

pomiędzy punktami, są stosunkowo małe. W przypadku testowanego zbioru 60000 punktów, średnia liczba punktów, których na podstawie odległości od punktu skrajnego nie można wykluczyć z epsilonowego sąsiedztwa wynosi 9198. Jest to nie tylko ponad trzykrotnie mniej niż w przypadku zerowego punktu referencyjnego, kiedy rzeczywiste odległości muszą być obliczane dla 29596 punktów, ale nawet o prawie 30% mniej niż w przypadku punktu maksymalnego. Należy jednak pamiętać, że w przypadku pozostałych zbiorów danych punkt losowy klasyfikował się jako jeden z gorszych punktów referencyjnych. Szczególnie niekorzystnym dla niego przypadkiem jest zbiór birch, który charakteryzuje się dość równomiernym rozłożeniem punktów w całej dziedzinie. W takim przypadku im punkt referencyjny znajduje się bliżej punktu środkowego, tym selektywność rozwiązania jest mniejsza.



Wykres 5.5 Porównanie liczności zbiorów punktów o różnicy odległości do punktu referencyjnego nie większej niż $\epsilon=84094$, które nie mogą zostać wykluczone z epsilonowego sąsiedztwa punktu z danej odległości od punktu referencyjnego na podstawie warunku nierówności trójkąta.

Pomijając opisany przypadek punktu losowego w zbiorze sequoia, strategia wyboru punktu maksymalnego okazuje się najwydajniejsza dla wszystkich testowych zbiorów danych. Mimo iż taka obserwacja nie pozwala na wyciąganie wniosków, że jest to uniwersalna strategia, gwarantująca najlepsze wyniki dla dowolnego zbioru danych, to wpisuje się ona w bardziej ogólny wniosek, że w przypadku, gdy nie jest znana

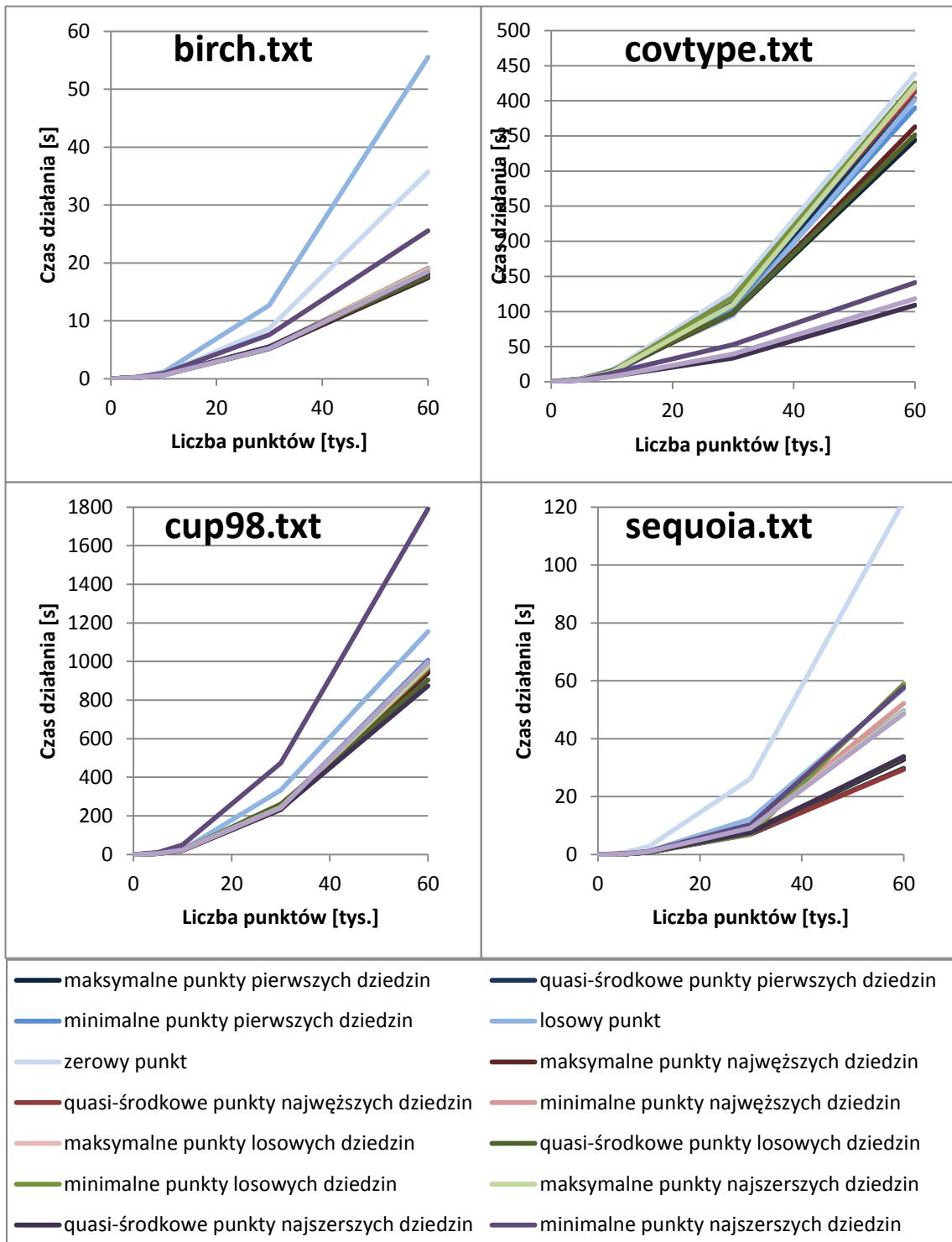
5. BADANIA EKSPERYMENTALNE

charakterystyka grupowanych danych, najbezpieczniejszą strategią jest wybór jednego ze skrajnych punków dziedziny zbioru danych.

5.3.2.2. Wybór wielu punktów referencyjnych

W przypadku algorytmu TI-DBSCAN-Ref z wieloma punktami referencyjnymi strategia wyboru punktów jest bardziej złożona. Zgodnie z opisem z rozdziału 3.1.2, wpływ na wydajność algorytmu ma tu nie jeden, ale aż trzy parametry, takie jak: liczba punktów referencyjnych, strategia wyboru punktów oraz strategia wyboru atrybutów. Liczba kombinacji utrudnia jednoczesną analizę wpływu wszystkich parametrów, dlatego dla zachowania czytelności wykresów i lepszego zrozumienia tematu, badanie przeprowadzone zostało w kilku etapach.

Poglądowe badanie wszystkich możliwych kombinacji strategii wyboru dwóch punktów referencyjnych, którego wyniki ilustruje wykres 5.6, pozwala wyciągnąć kilka podstawowych wniosków dotyczących zależności pomiędzy wyborem strategii. Testy potwierdziły, że tak jak w przypadku algorytmu TI-DBSCAN, niezależnie od zastosowanej strategii wyboru atrybutów, dla większości testowych zbiorów danych najlepszą strategią wyboru punktów referencyjnych jest strategia punktów maksymalnych. Ponownie jedynym wyjątkiem jest zbiór sequoia. Tym razem, wybrane zgodnie z tą strategią dwa maksymalne punkty referencyjne tworzą zestaw wydajniejszy od dwóch punktów losowych. Jednak w przypadku strategii wyboru atrybutów w kolejności rosnących zakresów dziedzin, najwydajniejszy zestaw tworzą punkty quasi-środkowe. Już pierwszy quasi-środkowy punkt referencyjny pozwala zoptymalizować wyszukiwanie sąsiadów, ponieważ położony jest w pobliżu szczególnie korzystnego obszaru na jednym z końców podłużnej grupy utworzonej przez większość punktów. Dodatkowe wsparcie dobrze położonego drugiego punktu referencyjnego pozwala uzyskać najwyższą wydajność. Co ważne, sukces strategii punktów quasi-środkowych nie powtarza się jednak w przypadku innych zbiorów danych, a kontrprzykładem jest wynik dla zbioru cup98 i punktów quasi-środkowych z najszerzej dziedzin, który jest prawie dwukrotnie gorszy od wyników pozostałych testowanych zestawów punktów.



Wykres 5.6 Porównanie wydajności strategii wyboru dwóch punktów referencyjnych w algorytmie TI-DBSCAN-Ref dla przykładowych zbiorów danych.

5. BADANIA EKSPERYMENTALNE

zbior	liczba punktów	pierwsze dziedziny			losowe dziedziny			zerowy punkt
		maksymalne punkty	quasi-środkowe punkty	minimalne punkty	maksymalne punkty	quasi-środkowe punkty	minimalne punkty	
birch	5000	203	219	203	203	204	203	219
	10000	688	671	672	687	657	656	828
	30000	5172	5219	5250	5531	5391	5484	8657
	60000	17437	17969	18625	17781	18328	18766	35704
covtype	5000	3813	3359	3156	3984	3735	3735	4360
	10000	14766	12328	11313	15672	14234	13765	16641
	30000	97718	111250	102313	98172	119797	109890	127906
	60000	344422	402828	390047	351672	425344	422032	438500
cup98	5000	5563	5860	5844	5562	5828	5875	5875
	10000	22406	24047	24031	22343	24109	24000	24109
	30000	233532	242156	245297	261968	247672	243704	244203
	60000	940890	969203	1007840	904672	965656	974843	966844
sequoia	5000	203	188	203	203	171	219	640
	10000	813	797	969	797	797	969	2781
	30000	7359	7016	9312	7453	7047	9188	26266
	60000	32797	29781	49657	33500	58922	49344	121985
		najwęższe dziedziny			najszersze dziedziny			losowe punkty
		maksymalne punkty	quasi-środkowe punkty	minimalne punkty	maksymalne punkty	quasi-środkowe punkty	minimalne punkty	
birch	5000	203	203	219	203	265	219	0
	10000	688	656	672	672	922	656	250
	30000	5282	5391	5422	5500	7579	5234	1156
	60000	17578	18172	19125	18296	25562	18531	12704
covtype	5000	4000	3719	3750	2172	3063	2171	55547
	10000	14391	13765	13672	7657	12297	7219	0
	30000	97703	119985	119906	33609	52921	39578	4766
	60000	362984	413235	418984	109000	141344	118078	16891
cup98	5000	5562	5844	5781	5547	11438	5859	94578
	10000	22375	23907	24062	22281	49954	24016	401687
	30000	233828	245797	242047	236610	474610	242281	0
	60000	945500	961719	995078	874422	1791330	994703	7594
sequoia	5000	188	172	219	203	266	204	23313
	10000	813	781	969	829	1109	969	333563
	30000	8187	7203	9281	7547	10328	9141	1154420
	60000	33235	29375	52265	33843	57781	48578	0

Tabela 5.4 Porównanie wydajności strategii wyboru dwóch punktów referencyjnych w algorytmie TI-DBSCAN-Ref dla przykładowych zbiorów danych.

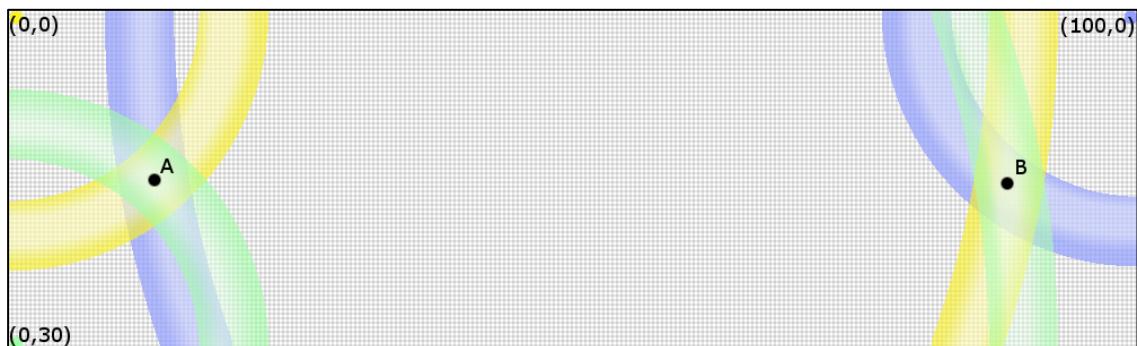
Strategia punktów losowych nie sprawdziła się nie tylko w zbiorze sequoia, w przypadku którego spadła z pierwszego na końcowe miejsce. Tak wybrany zestaw punktów referencyjnych nie sprawdzał się dobrze w żadnym przypadku. Zbiory punktów, których nie można wykluczyć z epsilonowego sąsiedztwa na podstawie odległości do dwóch losowych punktów są większe niż w przypadku zastosowania wybieranych zgodnie ze strategią dwóch sąsiednich punktów skrajnych. Tak jak dla TI-DBSCAN, najlepszym przykładem tej zależności jest najbardziej jednolity zbiór birch. Wyjaśnieniem, dlaczego jedynie niewiele lepiej 1la strategia wyboru punktu zerowego jest rozdział 3.1.2, w którym opisałem, dlaczego w jej przypadku zwiększenie liczby punktów referencyjnych nie jest sensowne, przez co TI-DBSCAN-Ref traci w tym przypadku swoją przewagę. Dobrze widać to podczas porównania ze strategią wyboru punktu minimalnego, która w TI-DBSCAN uzyskiwała bardzo zbliżone wyniki, a teraz jest dwukrotnie bardziej wydajna. Dokładniej przenalizowane zostało to w znajdującym się w dalszej części rozdziału porównaniu wydajności tych dwóch algorytmów.

Wcześniej, wykorzystując tabelę 5.4, warto przeanalizować wpływ drugiego parametru, czyli strategii wyboru atrybutów. Zgodnie z przypuszczeniami, różnice w wydajności widoczne są najlepiej w przypadku danych wielowymiarowych o atrybutach o różnym zakresie dziedzin, tak jak ma to miejsce w przypadku zbioru covtype, który zawiera zarówno atrybuty z wartościami zmiennych ilościowych, jak i binarnych. W jego przypadku strategia wyboru atrybutów o najszerzej dziedzinie jest ponad dwukrotnie wydajniejsza przy wyborze punktów quasi-środkowych, ponad trzykrotnie wydajniejsza przy wyborze punktów maksymalnych i jeszcze bardziej dla punktów minimalnych. Pozostałe strategie wyboru atrybutów dają porównywalne wyniki, z tylko nieznaczną przewagą strategii wybierania kolejnych atrybutów. Można to prosto wytlumaczyć przypominając, że to pierwszych 10 atrybutów charakteryzują szerskie dziedziny. Natomiast w przypadku losowych atrybutów czterokrotnie większe szanse na wylosowanie mają atrybuty o wartościach binarnych, których jest cztery razy więcej.

W przypadku dwuwymiarowych zbiorów sequoia i birch dodatkowym czynnikiem wpływającym na niewielkie różnice wydajności pomiędzy strategiami wyboru atrybutów o najszerzych i najwęższych dziedzinach są niewielkie różnice zakresów dzie-

5. BADANIA EKSPERYMENTALNE

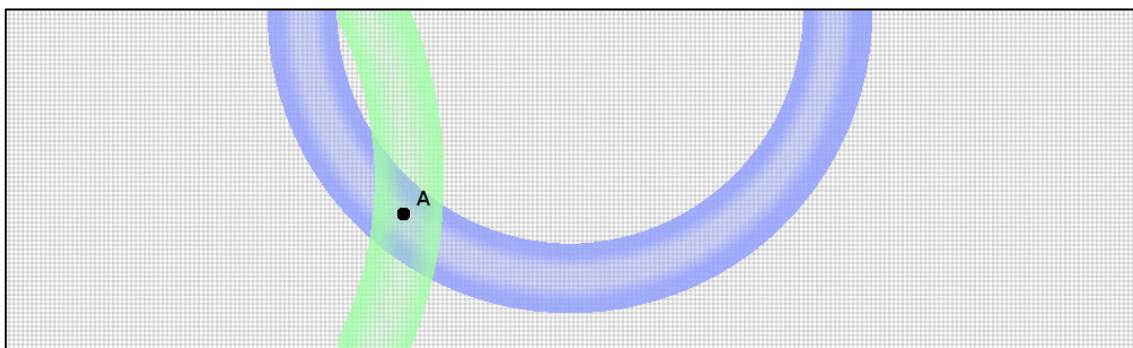
dzin atrybutów tych zbiorów. Strategie wybierania atrybutów są w związku z tym prawie symetryczne, dlatego jedyna różnica mogłaby wynikać z niesymetrycznego ułożenia danych. Porównanie strategii wyboru atrybutów dla punktów minimalnych w przypadku dwuwymiarowego zbioru o różnym zakresie dziedzin przedstawiłem na ilustracji 5.1. W przypadku punktu A, zaznaczony jako przecięcie obszaru żółtego i niebieskiego, zbiór punktów, których nie można wykluczyć z epsilonowego sąsiedztwa na podstawie odległości do punktów referencyjnych $(0,0)$ i $(100,0)$ jest nieznacznie większy od zbioru wyznaczonego na podstawie sprawdzania odległości do pary punktów $(0,0)$ i $(30,0)$, czyli przecięcia obszaru żółtego i zielonego. Jednak w przypadku znajdującego się dalej od punktu referencyjnego $(30,0)$ punktu B, zupełnie przeciwnie, przecięcie obszaru żółtego i zielonego jest zdecydowanie większe od przecięcia obszaru żółtego i niebieskiego. Prowadzi to do wniosku, że im bliżej siebie znajdują się takie skrajne punkty referencyjne tym więcej punktów spełnia dla obydwu z nich nierówność trójkąta.



Ilustracja 5.1. Porównanie sprawności optymalizacji opartej na dwóch minimalnych punktach referencyjnych wybieranych zgodnie ze strategią wyboru atrybutów o najwęższych (zakres żółty i zielony) i najszerzszych (zakres żółty i niebieski) dziedzinach dla przykładowych punktów A i B oraz wartości $\epsilon = 3$.

Zgodnie z wynikami z tabeli 5.4, strategia wyboru najszerzszych dziedzin sprawdza się dobrze zarówno przy wyborze minimalnych, jak i maksymalnych punktów referencyjnych. Jednak zdecydowanie większa różnica wydajności tych strategii, na niekorzyść strategii najszerzszych dziedzin, występuje w przypadku wyboru quasi-srodkowych punktów referencyjnych. Podobnie jak wcześniej, nie jest to bardzo widoczne w przypadku dwuwymiarowych zbiorów o porównywalnych dziedzinach. Naj-

lepszym przykładem jest wielowymiarowy zbiór cup98 o całkowitoliczbowych atrybutach o różnych dziedzinach, dla którego różnica wydajności jest prawie dwukrotna. Uproszczona sytuacja dla dwuwymiarowego zbioru o zróżnicowanych dziedzinach atrybutów przedstawiona została na ilustracji 5.2. Należy pamiętać, że dla dwuwymiarowego zbioru przy dwóch punktach referencyjnych ostatecznie prawdziwa odległość będzie obliczona dla takiej samej liczby punktów z przecięcia zielonego i niebieskiego obszaru. Jednak, nawet w takim przypadku znaczenie ma wybór głównego punktu referencyjnego, który decyduje o sortowaniu zbioru danych i liczbie punktów, które muszą być sprawdzone z użyciem pozostałych punktów referencyjnych. W przypadku, gdy liczba punktów referencyjnych jest mniejsza niż liczba atrybutów, wpływ ten jest jeszcze większy, ponieważ zależy od tego liczba najbardziej czasochłonnych operacji obliczania rzeczywistej odległości pomiędzy punktami. Z powyższymi wnioskami mogą nie zgadzać się zaprezentowane w tabeli 5.4 wyniki dwóch quasi-środkowych punktów referencyjnych dla wielowymiarowego zbioru covtype. Zgodnie z przedstawionym we wcześniejszym akapicie wyjaśnieniem, wynika to z nietypowej charakterystyki atrybutów, które obejmują zarówno atrybuty o szerokich dziedzinach, jak atrybuty binarne.



Ilustracja 5.2. Porównanie optymalizacji opartej na dwóch quasi-środkowych punktach referencyjnych wybieranych zgodnie ze strategią najwcześniej (zakres zielony jako podstawowy) i najpóźniej (zakres niebieski jako podstawowy) dziedzin dla przykładowego punktu A.

Powyższe analizy wykonywane były dla dwóch punktów referencyjnych, lecz TI-DBSCAN-Ref może wykorzystywać dowolną liczbę punktów. Jest to kolejny parametr wejściowy, który także może mieć istotne znaczenie na wydajność algorytmu. Jak zostało to już podkreślone, dodatkowe punkty to dodatkowy koszt, nie tylko

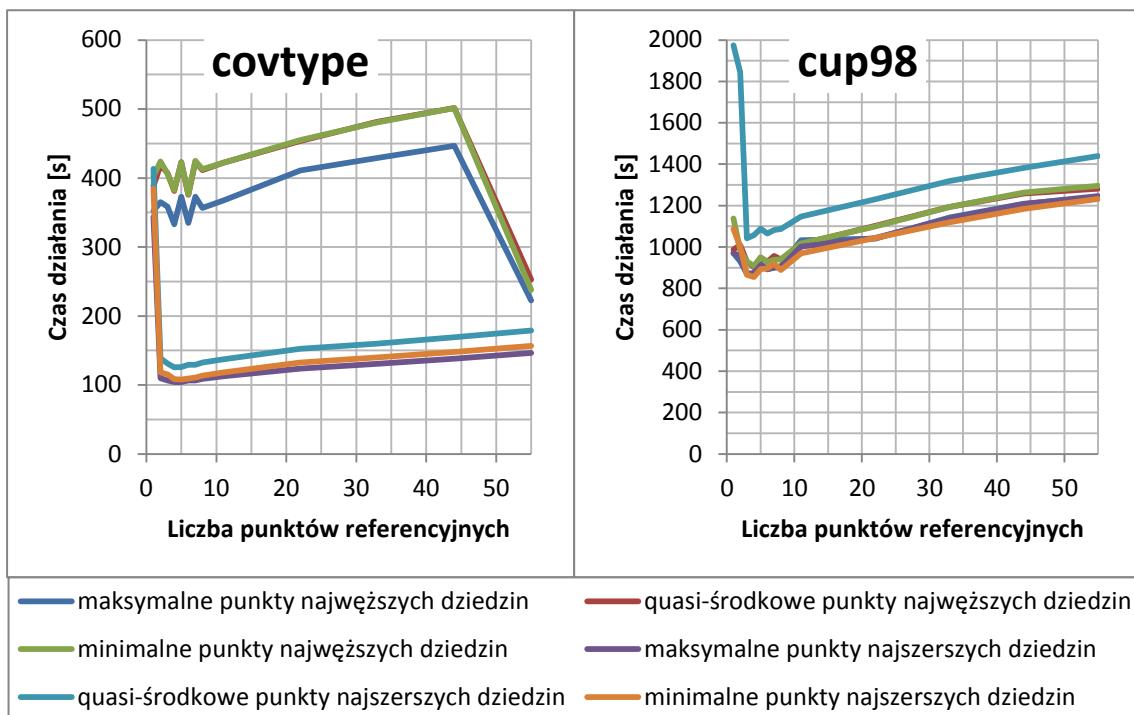
5. BADANIA EKSPERYMENTALNE

podczas fazy przygotowawczej, kiedy dla każdego punktu ze zbioru danych muszą zostać obliczone odległości do wszystkich punktów referencyjnych, ale także podczas fazy szukania epsilonowego otoczenia, kiedy musi być sprawdzone spełnienie warunku nierówności trójkąta dla wszystkich punktów referencyjnych. Z drugiej strony, każdy kolejny punkt może zawieźć liczbę punktów, dla których muszą być obliczane rzeczywiste odległości.

zbior	liczba punktów ref.	najwęższe dziedziny			najszerzsze dziedziny		
		maksymalne punkty	quasi-środkowe punkty	minimalne punkty	maksymalne punkty	quasi-środkowe punkty	minimalne punkty
covtype (60 tys. punktów)	1	352516	388141	402672	343546	413453	384281
	2	365078	419063	423797	109640	138390	118140
	3	358657	407094	406500	106625	131079	115406
	4	333094	381484	382422	104234	125453	108250
	5	372875	423000	422344	104266	126063	108171
	6	334703	375907	376454	106407	129453	109578
	7	373125	423828	425031	106547	129563	110657
	8	356454	411532	412688	108969	132719	113484
	11	367172	421953	422203	112609	137079	118375
	22	410812	453828	455141	123687	152578	132438
	33	429266	481328	480766	130750	159906	140219
	44	446844	501453	501390	138203	169015	148109
	55	222718	252907	237750	146422	178875	156781
cup98 (60 tys. punktów)	1	971109	987579	1137300	968797	1973830	1086500
	2	931344	1009170	986688	958265	1844670	1001080
	3	871187	928500	928703	871063	1042270	866422
	4	874078	903000	908360	871266	1058060	854766
	5	916766	947094	950172	911406	1088830	893813
	6	892312	928828	927312	893265	1065610	896875
	7	903500	956266	939125	899016	1082050	920515
	8	905547	939328	940328	902688	1087580	888750
	11	1033390	1013830	1017330	1001530	1147640	968531
	22	1040060	1101580	1099910	1043390	1231300	1043470
	33	1142920	1193020	1192770	1137520	1318980	1118770
	44	1209310	1258470	1263140	1206700	1381560	1183660
	55	1241590	1282080	1295220	1246480	1438750	1232500

Tabela 5.5 Porównanie wydajności algorytmu TI-DBSCAN-Ref na testowych zbiorach danych covtype i cup98 (po 60 tysięcy punktów) w zależności od liczby punktów referencyjnych i strategii ich wyboru.

W testach wpływu liczby punktów referencyjnych na wydajność algorytmu TI-DBSCAN-Ref ograniczyłem się do zbiorów danych, które posiadają więcej niż dwa wymiary, czyli covtype i cup98 (ograniczonych jak poprzednio do 60 tysięcy punktów). Skupiłem się na testach najbardziej charakterystycznych strategii najszerzszych i największych dziedzin i minimalnych i maksymalnych punktów, a także strategii punktów quasi-środkowych.



Wykres 5.7. Porównanie wydajności algorytmu TI-DBSCAN-Ref na testowych zbiorach danych covtype i cup98 (po 60 tysięcy punktów) w zależności od liczby punktów referencyjnych i strategii ich wyboru/

Wykres 5.7 wyraźnie pokazuje, że zwiększanie liczby punktów referencyjnych nie jest równoznaczne ze zwiększeniem wydajności algorytmu. W przypadku strategii najszerzszych dziedzin można zauważyc zgodny z oczekiwaniami trend, sugerujący, że istnieje optymalna liczba punktów, dla której korzyści, wynikające z wykluczania dodatkowych punktów ze zbioru, dla którego obliczane są rzeczywiste odległości, są większe niż koszty sprawdzania dla każdego punktu dodatkowych warunków. Jak widać w przypadku badanych zbiorów danych i użytych parametrów, optymalna liczba to zaledwie trzy lub cztery punkty referencyjne, choć największą różnicę wydajności widać pomiędzy jednym a dwoma punktami dla covtype i dwoma a trzema

punktami dla cup98. Generalnie, wykres z lewej strony opisanego minimum jest zdecydomniej bardziej stromy niż z prawej, co oznacza, że przeszacowanie parametru i użycie nieznacznie większej liczby punktów nie wpływa bardzo negatywnie na wydajność algorytmu. Warto jednak zwrócić uwagę, że w przypadku cup98 liczby większe od 10 powodowały, że wydajność TI-DBSCAN-Ref była niższa niż dla jednego punktu.

5.3.3. Ocena wydajności

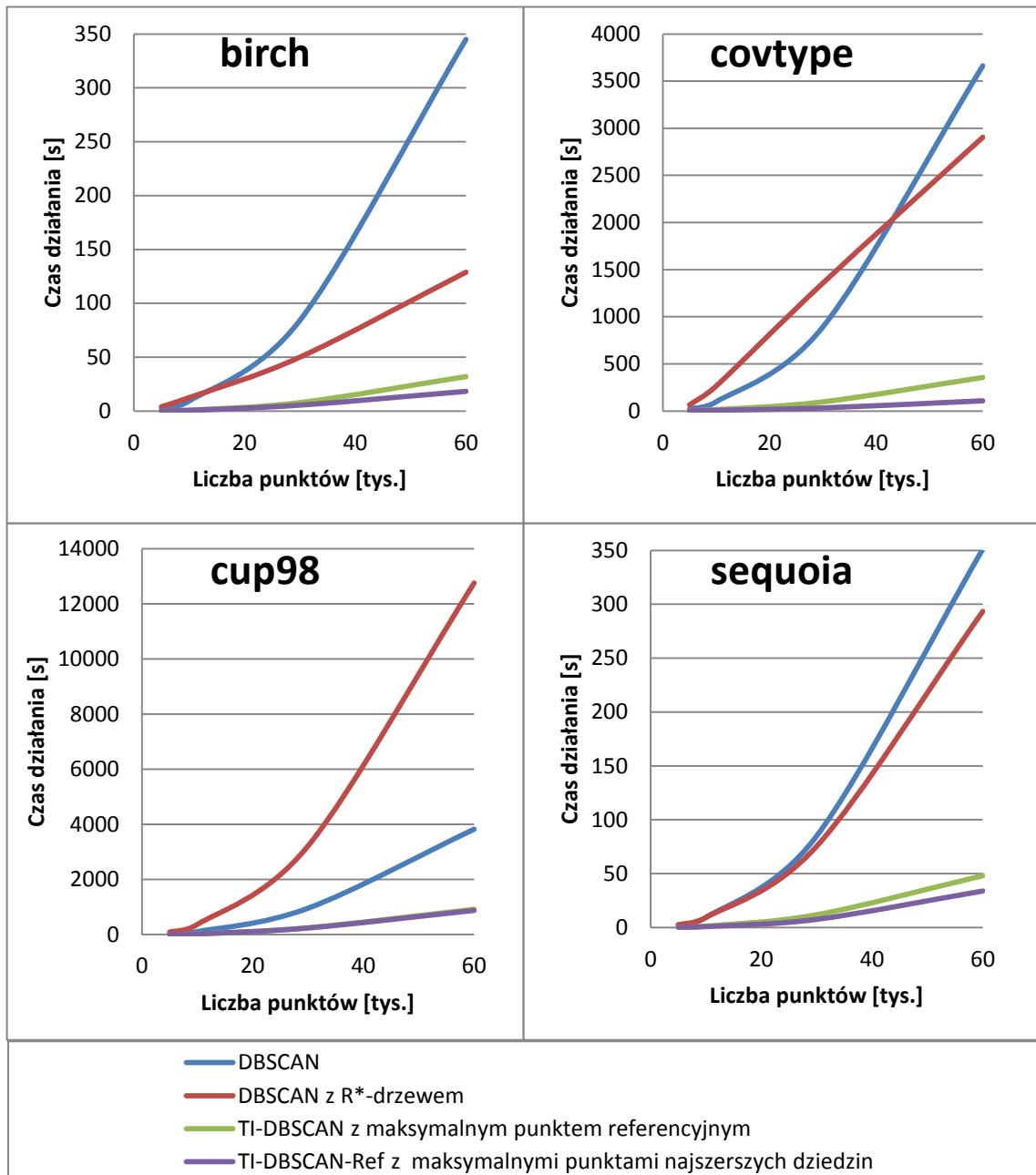
Przeprowadzone testy, których wyniki ilustruje wykres 5.8, jednoznacznie potwierdzają wyraźny wzrost wydajności TI-DBSCAN i TI-DBSCAN-Ref nie tylko w stosunku do algorytmu DBSCAN bez indeksu, ale także w stosunku do wersji wykorzystującej zalecane przez twórców oryginalnego algorytmu indeks R*-drzewo. Co ciekawe, zastosowanie R*-drzewa w niektórych przypadkach wręcz znaczco zmniejszało wydajność algorytmu. O ile przynosi ono najlepsze efekty w przypadku zbioru birch, o tyle cup98 jest zdecydowanym kontrprzykładem. Wynika to z zależności wydajności R*-drzewa od liczby wymiarów, która negatywnie wpływa zarówno na proces budowania indeksu, jak i jego późniejsze przeszukiwanie. Jest to mniej widoczne w przypadku zbioru covtype, który choć jest porównywalny z cup98 pod względem liczby atrybutów, to wyróżnia się wąskimi dziedzinami, z których większość ogranicza się do wartości binarnych.

Jak widać w tabeli 5.6, wersja oparta na nierówności trójkąta jest zdecydowanie mniej zależna od liczby wymiarów. Ma ona liniowy wpływ na proces obliczania odległości referencyjnych, który stanowi mniej istotny od sortowania etap przygotowań do właściwego grupowania. Samo przeszukiwanie posortowanego zbioru danych w celu znalezienia potencjalnych sąsiadów, dla których obliczone muszą być rzeczywiste odległości, jest już niezależne od liczby wymiarów, ponieważ jedynym branym pod uwagę atrybutem jest różnica wspomnianych odległości referencyjnych.

zbior	liczba punktów	DBSCAN	DBSCAN z R*-drzewem		TI-DBSCAN z maksymalnym punktem referencyjnym		TI-DBSCAN-Ref z dwoma maksymalnymi punktami najszerzych dziedzin	
			łącznie	przygotowania	łącznie	przygotowania	łącznie	przygotowania
birch	5000	2391	828	4031	0	172	16	203
	10000	9359	2328	12500	16	891	31	672
	30000	84156	8047	50000	78	7860	125	5500
	60000	350032	16891	128906	140	32000	265	18296
covtype	5000	24312	13109	66359	15	4000	31	2172
	10000	99000	27906	266062	47	15797	79	7657
	30000	892484	87938	1357670	172	97578	250	33609
	60000	3662830	183625	2904600	375	356281	531	109000
cup98	5000	26111	13844	91281	32	5532	31	5547
	10000	107144	30515	366388	47	21985	78	22281
	30000	954133	96469	3214410	187	244875	250	236610
	60000	3822100	209075	12756000	359	915265	532	874422
sequoia	5000	2375	422	2500	16	297	16	203
	10000	9391	1016	9328	16	1266	47	829
	30000	84968	3656	75422	78	11828	125	7547
	60000	351313	7860	293408	156	48062	265	33843

Tabela 5.6 Porównanie wydajności podstawowej wersji algorytmu DBSCAN bez optymalizacji z wersją wykorzystującą R*-drzewo, a także wersjami TI-DBSCAN z maksymalnym punktem referencyjnym i TI-DBSCAN-Ref z dwoma maksymalnymi punktami referencyjnymi z najszerzych dziedzin. W przypadku wersji wykorzystujących indeks dodatkowo wskazane zostały czasy etapu przygotowań (budowania indeksu) do właściwego grupowania.

Mimo, iż w porównaniach ograniczyłem się tylko do jednego wariantu TI-DBSCAN i jednego wariantu TI-DBSCAN-Ref, a nie dostosowywałem strategii wyboru punktów referencyjnych do zbioru danych, algorytmy te we wszystkich przypadkach, bez wyjątku, okazały się zdecydowanie wydajniejsze od wersji bez indeksu oraz z R*-drzewem. Różnica wydajności była tym większa, im większy był zbiór danych. W przypadku wielowymiarowego zbioru covtype cały proces grupowania algorytmem TI-DBSCAN-Ref trwał znacznie krócej niż samo zbudowanie R*-drzewa. Jednak nawet dla dwuwymiarowego zbioru birch czasy trwania tych procesów są porównywalne. Co istotne, testy nie wykazały żadnego zbioru danych, dla którego TI-DBSCAN czy TI-DBSCAN-Ref okazywał się mniej wydajny od porównywanych algorytmów DBSCAN w wersji bez optymalizacji i wykorzystującej R*-drzewo.



Wykres 5.8 Porównanie wydajności podstawowej wersji algorytmu DBSCAN bez optymalizacji z wersją wykorzystującą R*-drzewo, a także wersjami TI-DBSCAN i TI-DBSCAN-Ref.

W przypadku przyjętych, zgodnie z wnioskami z poprzednich podrozdziałów, najbardziej uniwersalnych strategii punktów maksymalnych dodatkowy punkt referencyjny zawsze decydował o przewadze TI-DBSCAN-Ref nad opartym na jednym punkcie TI-DBSCAN. W tym porównaniu różnica nie jest już jednak tak znacząca i mocno zależy od danych testowych. O ile w przypadku 60-tysięcznego zbioru covty-

pe różnica jest ponad trzykrotna, o tyle dla drugiego wielowymiarowego zbioru cup98, o mniej zróżnicowanych dziedzinach atrybutów, wydajność obydwu algorytmów jest porównywalna.

Zbiór (liczba punktów)	wartość ϵ	DBSCAN	DBSCAN z R*-drzewem	TI-DBSCAN z maksymalnym punktem referencyjnym	TI-DBSCAN-Ref z dwoma maksymalnymi punktami najszerzych dziedzin
Birch (60 tys.)	0	348406	21937	157	266
	6244	349250	25938	2891	1235
	31220	349641	61968	16156	7125
	62440*	350032	128906	32000	18296
	1248810	418844	4800980	307172	355563
covtype (60 tys.)	0	3782160	724734	390	578
	52	3699630	1415560	35938	2484
	261	3640050	2113660	178469	29641
	521*	3662830	2904600	356281	109000
	10422	4388580	14273000	2582980	2517730
cup98 (60 tys.)	0	3829630	5447840	578	578
	43	3822550	9816000	89656	64547
	214	3787020	11445900	477437	479313
	428*	3822100	12756000	915265	874422
	8566	4512550	15033600	2629860	2675590
sequoia (60 tys.)	0	348953	12531	156	281
	8409	349547	22281	4844	1984
	42047	350781	111016	25515	12453
	84094*	351313	293408	48062	33843
	1681880	423969	4019500	313890	367937

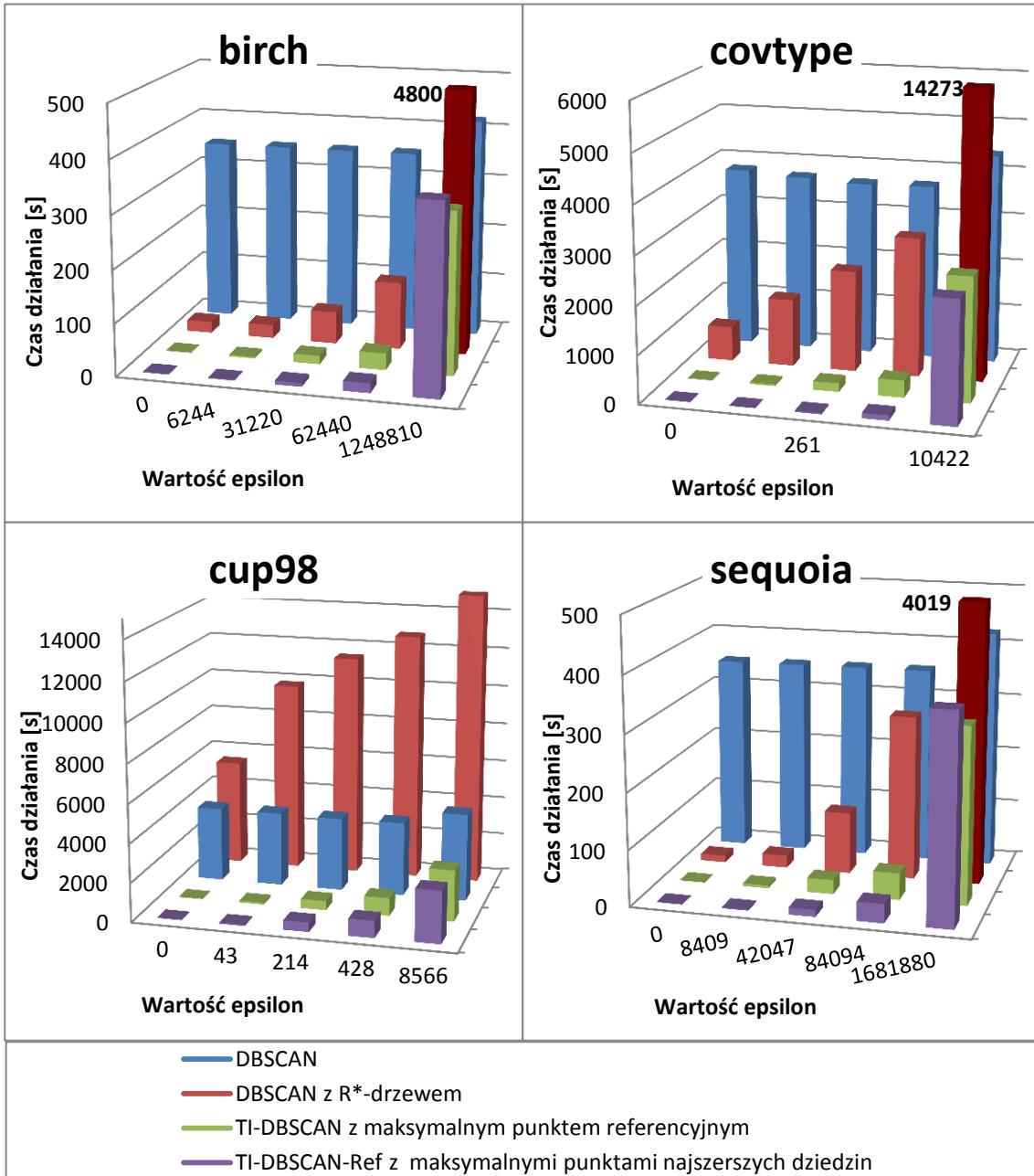
Tabela 5.7 Porównanie wydajności podstawowej wersji algorytmu DBSCAN bez optymalizacji z wersją wykorzystującą R*-drzewo, a także wersjami TI-DBSCAN i TI-DBSCAN-Ref w zależności od przyjętej wartości epsilon (od zera do długości przekątnej dziedziny zbioru danych). Wartości oznaczone symbolem * to domyślne wartości używane w pozostałych testach na 60 tysiącach punktów przykładowych zbiorów danych.

Przedstawione w tabeli 5.7 wyniki testów potwierdzają, że wzrost wydajności wynikający z zastosowania nierówności trójkąta zależy w dużym stopniu także od przyjętej wartości parametru ϵ . Największa korzyść z zastosowania sortowania i szacowania odległości z wykorzystaniem nierówności trójkąta występuje w przypadku małych wartości ϵ , które pozwalają bez liczenia rzeczywistych odległości wykluczyć ze zbioru potencjalnych sąsiadów największą liczbę punktów. Co więcej, właśnie w przypad-

5. BADANIA EKSPERYMENTALNE

ku małych wartości ϵ , widoczna jest największa korzyść z wykorzystania dodatkowego punktu referencyjnego. TI-DBSCAN-Ref z dwoma punktami referencyjnymi jest mniej wydajny od TI-DBSCAN jedynie w przypadku skrajnie małych i skrajnie dużych wartości ϵ , kiedy koszt obliczania odległości do dodatkowych punktów referencyjnych i sprawdzania ich podczas wyszukiwania sąsiedztwa nie jest rekompensowany przez zmniejszenie zbioru punktów, dla których obliczane muszą być rzeczywiste odległości. Co istotne, nawet w przypadku wartości epsilon równej długości przekątnej dziedziny zbioru danych, kiedy nie ma możliwości wykluczenia żadnego punktu na podstawie nierówności trójkąta, algorytm TI DBSCAN wciąż uzyskuje lepsze wyniki od oryginalnego DBSCAN. Wynika to z zastosowania dodatkowej modyfikacji, wykluczającej ze zbioru przeszukiwanych punktów, te już zbadane, co zapobiega wielokrotnemu liczeniu odległości pomiędzy tymi samymi punktami.

Uzyskanie pozytywnych wyników TI-DBCAN i TI-DBSCAN-Ref, także w najbardziej pesymistycznym przypadku największej wartości ϵ robi jeszcze większe wrażenie po wizualizacji rezultatów testów na wykresie 5.9. Jak widać, w przypadku największych z testowanych wartości ϵ , użycie indeksu R*-drzewa powoduje wzrost czasu grupowania nawet o rzad wielkości. W przypadku algorytmów wykorzystujących nierówność trójkąta, brak dodatkowej struktury danych, zwiększającej liczbę niesekwencyjnych odczytów pamięci, pozwala uniknąć tego, typowego dla większości indeksów, problemu. W połączeniu, ze znaczco mniejszą niż w przypadku R*-drzewa zależnością wydajności od liczby wymiarów, pozwala to wykorzystywać analizowaną modyfikację w zdecydowanie szerszym zakresie sytuacji.



Wykres 5.9 Porównanie wydajności podstawowej wersji algorytmu DBSCAN bez optymalizacji z wersją wykorzystującą R*-drzewo, a także wersjami TI-DBSCAN z maksymalnym punktem referencyjnym i TI-DBSCAN-Ref z dwoma maksymalnymi punktami referencyjnymi z najszerzszymi dziedzin na testowych zbiorach danych z 60000 punktów, w zależności od przyjętej wartości epsilon. Uwaga: Dla lepszego zilustrowania wyników, pionowa oszona została ograniczona, a słupki o wartościach wykraczających poza limit oznaczone zostały ciemniejszym odcieniem i opisane prawdziwą wartością, która nie została odwzorowana na wykresie.

5.4. TESTY ALGORYTMU NBC

Opisane w poprzednim podrozdziale, bardzo zadowalające wyniki zastosowania indeksu opartego na nierówności trójkąta w algorytmie TI-DBSCAN pozwalają przy-puszczać, że podobne efekty powinny zostać uzyskane w przypadku TI-NBC. Dodatkową niepewność wprowadza tu jednak, opisany w rozdziale 3.2.2, odmienny sposób wykorzystania nierówności trójkąta, która nie służy w tym przypadku do znajdowania epsilonowego otoczenia, lecz k-sąsiedztwa dowolnego punktu. Inaczej wygląda też podział złożoności poszczególnych etapów algorytmu, ponieważ samo grupowanie stanowiące najdłuższy etap TI-DBSCAN, podczas którego wyznaczane były wszystkie epsilonowe sąsiedztwa, w przypadku TI-NBC traci na znaczeniu na rzecz poprzedzającego go etapu wyznaczania wartości NDF.

5.4.1. Implementacja TI-dNBC

Przed właściwymi testami wydajności zoptymalizowanej wersji algorytmu sprawdziłem, jaki wpływ na wyniki ma wybór implementacji algorytmu. Zgodnie z informacjami z rozdziału 2.2 poza standardową wersją NBC zgodną z pseudokodami przedstawionymi przez autorów algorytmu [13], zaimplementowałem dodatkowo wersję dNBC.

Zaprezentowane w tabeli 5.8 wyniki testów dla standardowych parametrów wejściowych nie wykazały znaczących różnic wydajności pomiędzy algorytmami. Tłumaczyć można to tym, że dla badanych zbiorów danych i parametrów nie występuje problem niedeterminizmu podstawowej wersji algorytmu NBC, która wykrywa wszystkie grupy. Wyniki pokazują jednak, że czasy etapu grupowania są rzędu wielkości mniejsze niż czasy etapu wyznaczania wartości NDF, w którym zawiera się wyznaczanie k-sąsiedztwa wszystkich punktów. Biorąc pod uwagę fakt, że modyfikacja w dNBC dotyczy jedynie etapu grupowania, wywnioskować można, że nawet kilkukrotne zwiększenie czasu jego trwania miałoby pomijalny wpływ na ogólną wydajność algorytmu.

zbior	liczba punktów	TI-dNBC			TI-NBC		
		obliczanie NDF	grupowanie	łącznie	obliczanie NDF	grupowanie	łącznie
birch	5000	875	0	875	860	0	875
	10000	5937	16	5969	5907	15	5937
	30000	83156	141	83360	82000	156	82219
	60000	423875	656	424672	436078	625	436843
covtype	5000	11453	0	11469	11453	0	11484
	10000	49188	15	49250	49125	16	49187
	30000	459984	156	460296	461515	157	461828
	60000	1985640	656	1986630	1913980	625	1914950
cup98	5000	4985	15	5031	5000	0	5016
	10000	21828	16	21891	20969	15	21031
	30000	205078	157	205407	205235	156	205563
	60000	893937	625	894906	891656	594	892594
Sequoia	5000	3140	16	3156	3156	0	3171
	10000	13890	32	13953	14281	16	14313
	30000	141985	156	142203	143125	140	143328
	60000	609688	781	610625	600985	734	601859

Tabela 5.8 Porównanie wydajności algorytmów TI-NBC i deterministycznej wersji TI-dNBC dla przykładowych zbiorów danych i strategii zerowego punktu referencyjnego.

W dalszej analizie, z racji testowania wpływu wyboru punktu referencyjnego, od którego zależy kolejność sortowania zbioru danych, chciałem uniknąć sytuacji, gdy wyniki zależałyby od niedeterministyczności algorytmu NBC. W związku z tym do badania wydajności optymalizacji wykorzystującej nierówność trójkąta wykorzystana została deterministyczna wersja algorytmu dNBC.

5.4.2. Strategie wyboru punktów referencyjnych

Podobnie jak w przypadku opisywanego wcześniej TI-DBSCAN szybkość działania algorytmu zależy nie tylko od wydajnej implementacji, ponieważ niektóre parametry, takie jak strategia wyboru punktów referencyjnych, czy liczba tych punktów, powinny być dobrane do przetwarzanego zbioru danych. W kolejnych podrozdziałach przedstawiłem wyniki testów i wnioski dotyczące strategii wyboru parametrów oraz ich wpływu na wydajność algorytmów TI-dNBC i TI-dNBC-Ref.

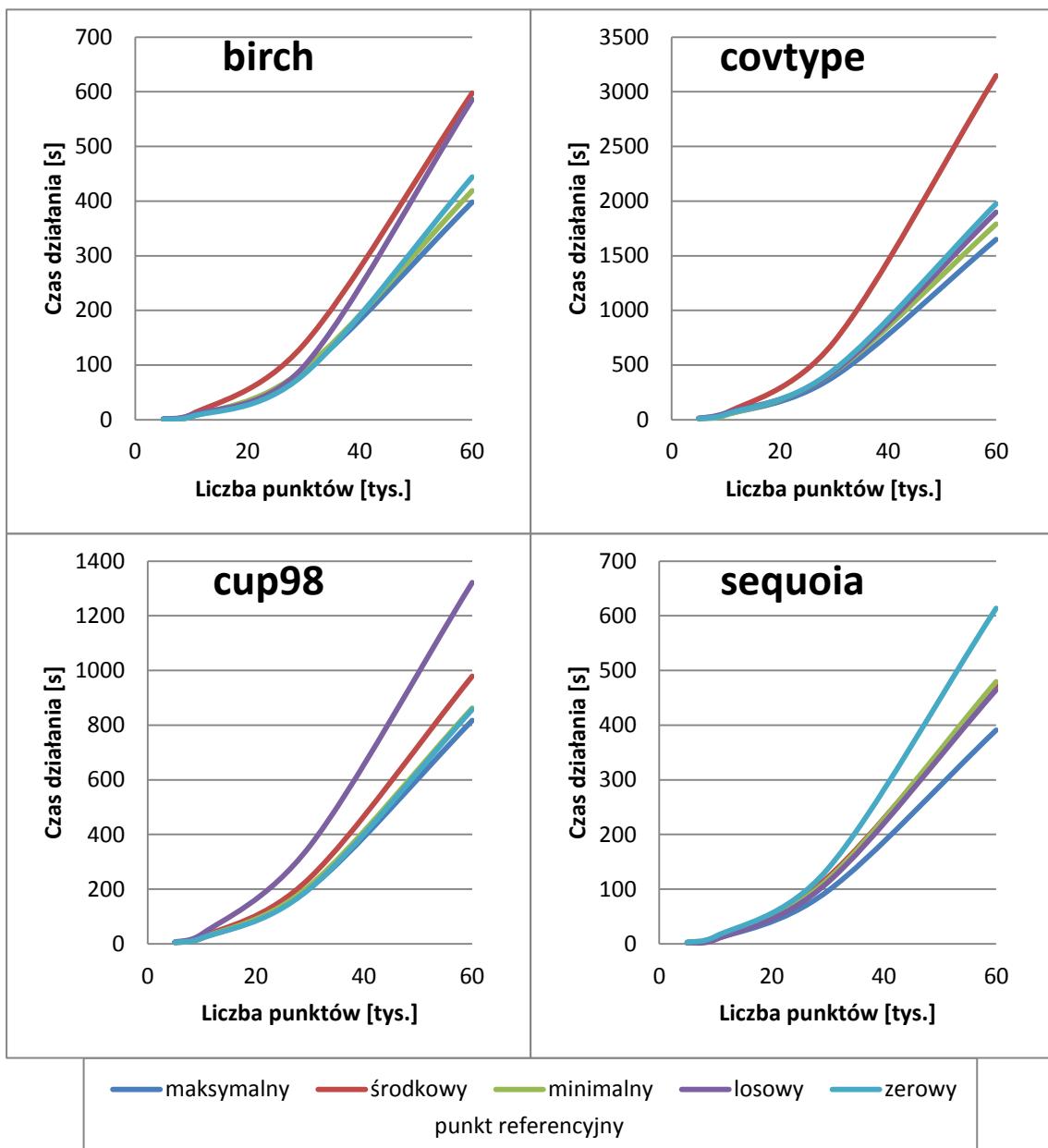
5. BADANIA EKSPERYMENTALNE

5.4.2.1. Wybór jednego punktu referencyjnego

Mimo opisanych różnic w wykorzystaniu nierówności trójkąta, prezentujący wydajność algorytmu TI-dNBC w zależności od strategii wyboru punktu referencyjnego wykres 5.10 przypomina wykres 5.3, który przedstawia analogiczną charakterystykę algorytmu TI-DBSCAN. Jedną z głównych różnic jest spadek wydajności strategii losowego punktu referencyjnego. Choć jest on najbardziej zauważalny w przypadku zbioru sequoia, gdzie strategia ta przestała być już najwydajniejsza, to spadek jej wydajności zauważalny jest także w przypadku zbiorów birch i covtype. Wciąż najlepszą strategią (w tym przypadku już bez wyjątku jakim był wcześniej zbiór sequoia) pozostaje wybór maksymalnego punktu referencyjnego, a najsłabiej wypada strategii punktów środkowych. Jednak w większości przypadków różnice pomiędzy wydajnościami poszczególnych strategii nieznacznie się zmniejszyły.

zbior	liczba punktów	strategia wyboru punktu referencyjnego				
		maksymalnego	środkowego	minimalnego	losowego	zerowego
birch	5000	860	1703	875	1250	875
	10000	5984	9907	5906	8734	5859
	30000	88500	137469	92375	97344	82640
	60000	398078	597734	419063	585672	444140
covtype	5000	10406	14375	9609	12328	11328
	10000	46031	59656	42359	60890	48313
	30000	393563	711531	435844	441094	457375
	60000	1648920	3147220	1789940	1898360	1975640
cup98	5000	4562	5594	4906	6234	4922
	10000	20156	24782	21437	35454	21282
	30000	203109	241703	213484	358469	202515
	60000	817219	979032	862343	1321590	857063
sequoia	5000	1829	2297	1937	2063	3016
	10000	8485	11015	10219	7609	13375
	30000	96297	121234	117156	112078	136797
	60000	390797	472297	479485	464375	613672

Tabela 5.9 Porównanie wydajności strategii wyboru punktu referencyjnego w algorytmie TI-dNBC dla przykładowych zbiorów danych.



Wykres 5.10 Porównanie wydajności strategii wyboru punktu referencyjnego w algorytmie TI-dNBC dla przykładowych zbiorów danych.

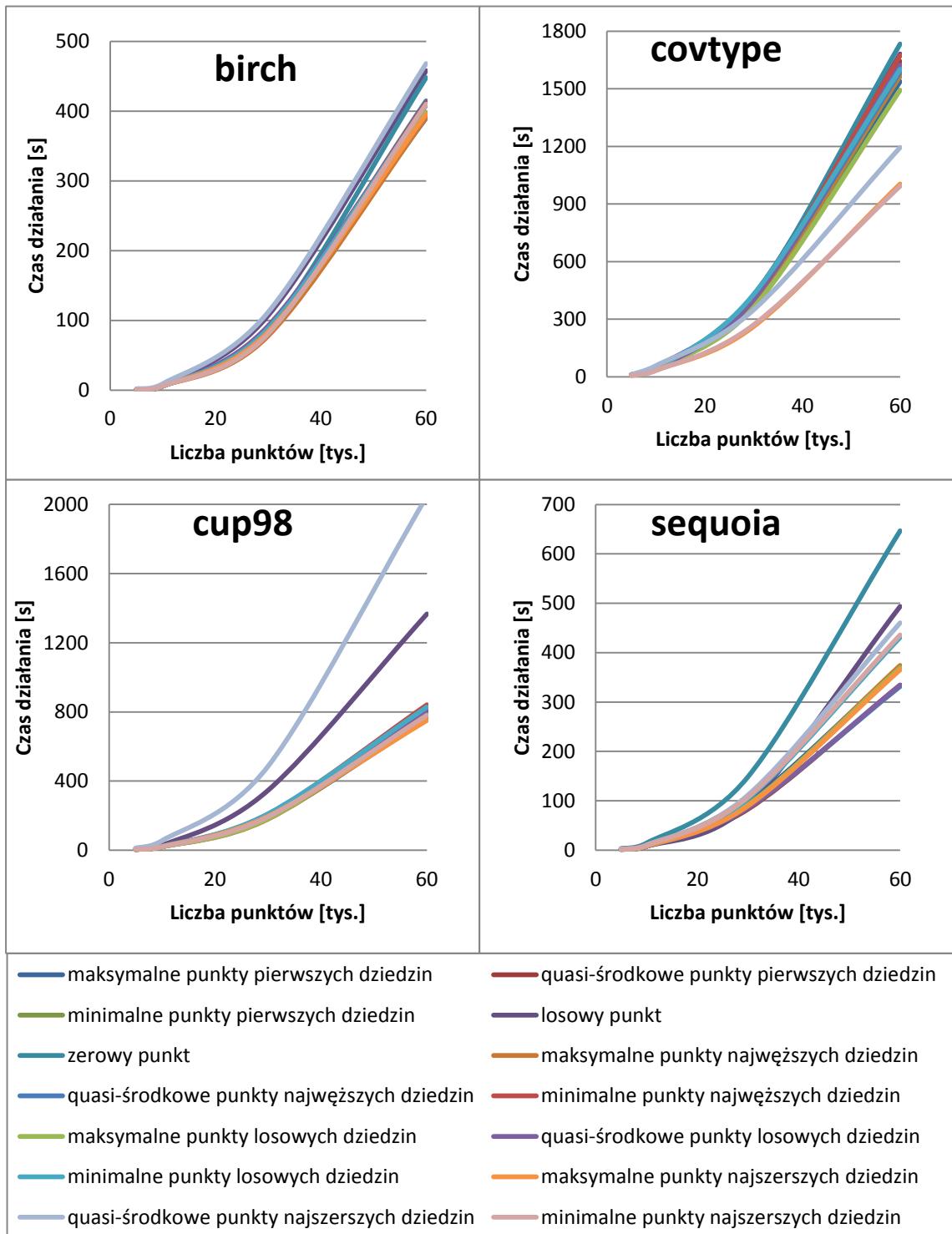
5.4.2.1. Wybór wielu punktów referencyjnych

Analogiczne testy jak w przypadku algorytmu TI-DBSCAN-Ref przeprowadziłem na TI-dNBC-Ref. Pozwoliło to porównać aktualne wyniki z zaobserwowanymi wcześniej tendencjami i wnioskami, które stanowią punkt wyjściowy analizy.

5. BADANIA EKSPERYMENTALNE

zbior	liczba punktów	pierwsze dziedziny			losowe dziedziny			zerowy punkt
		maksymalne punkty	quasi-środkowe punkty	minimalne punkty	maksymalne punkty	quasi-środkowe punkty	minimalne punkty	
birch	5000	890	891	906	876	1703	891	922
	10000	6000	5828	5922	5985	5829	5953	6015
	30000	86722	83237	90753	86627	92065	88315	86346
	60000	390091	408903	410466	398262	409591	409810	448015
covtype	5000	9688	8765	8454	9813	9407	9563	10547
	10000	44423	41502	40470	44923	41142	41627	47221
	30000	368074	423936	375184	361792	400997	428685	412138
	60000	1537580	1642300	1565220	1492530	1622390	1601630	1733140
cup98	5000	4468	4812	4812	4532	4797	4875	4859
	10000	19579	21078	20563	19641	21437	21188	20046
	30000	183063	200828	192469	181985	199515	210312	198328
	60000	763688	824141	803609	780125	801297	828406	787328
sequoia	5000	1742	1524	1821	1742	1883	1813	3266
	10000	7969	7617	9399	8204	10281	9266	14164
	30000	95518	82831	105722	91972	85347	104362	148271
	60000	371293	334214	433592	368965	334386	430998	646708
		najwęższe dziedziny			najszersze dziedziny			losowe punkty
		maksymalne punkty	quasi-środkowe punkty	minimalne punkty	maksymalne punkty	quasi-środkowe punkty	minimalne punkty	
birch	5000	890	891	907	891	1719	907	1063
	10000	6016	5813	5907	6001	9266	6032	7610
	30000	77862	84503	89768	83769	111816	80924	107019
	60000	390809	414779	412263	395716	468172	409982	457890
covtype	5000	10063	9188	9344	7453	11767	7125	11609
	10000	43689	42611	42892	34814	57767	31782	50080
	30000	386482	407076	405341	261556	348339	270134	422967
	60000	1571500	1587420	1672770	1004050	1194450	993297	1681340
cup98	5000	4469	4891	4891	4453	12219	4641	7812
	10000	19750	20828	20828	20016	55391	20109	27203
	30000	179813	207719	206297	196407	485063	192734	345875
	60000	778906	815969	841125	751078	2048630	781265	1366660
sequoia	5000	1758	1531	1812	1750	2227	1813	1649
	10000	8258	7633	9423	8047	10243	9563	7539
	30000	89941	85893	109035	89487	111363	107175	90738
	60000	374043	331495	431545	365340	460249	435795	493515

Tabela 5.10 Porównanie wydajności strategii wyboru dwóch punktów referencyjnych w algorytmie TI-dNBC-Ref dla przykładowych zbiorów danych.



Wykres 5.11 Porównanie wydajności strategii wyboru dwóch punktów referencyjnych w algorytmie TI-dNBC-Ref dla przykładowych zbiorów danych. Wizualizacja danych z tabeli 5.10.

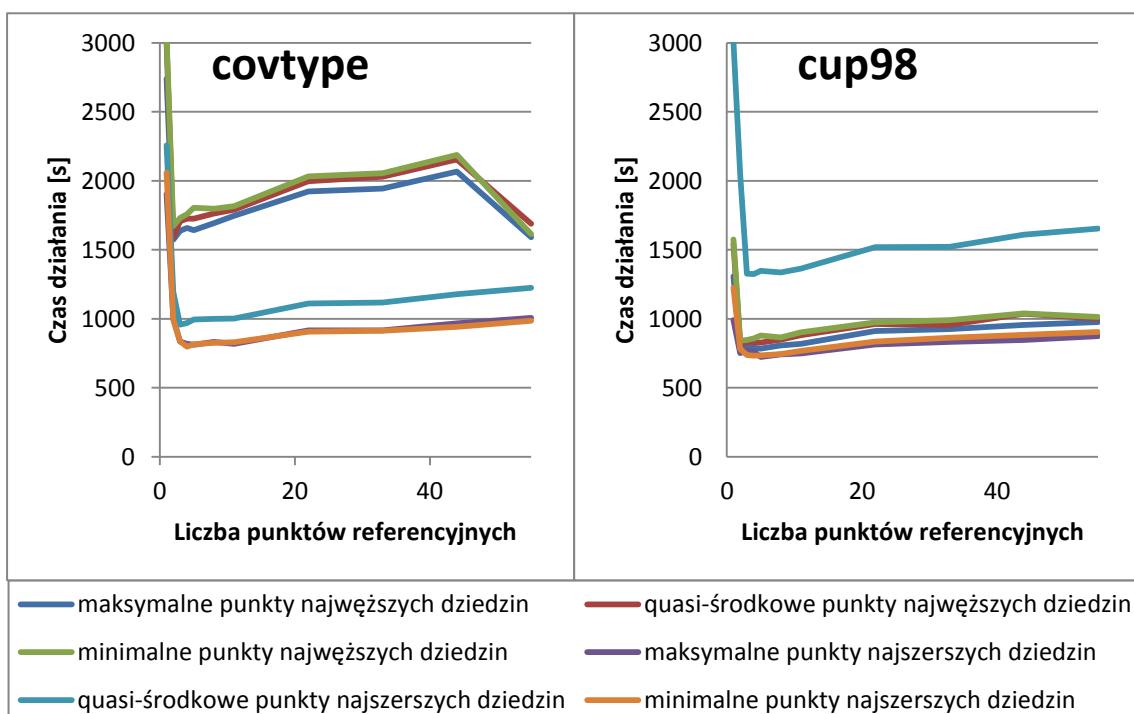
5. BADANIA EKSPERYMENTALNE

Podobnie jak w rozdziale 5.3.2.2, analizę rozpoczęłem od porównania wydajności wszystkich kombinacji strategii wyboru dwóch punktów referencyjnych, która pozwala na wykrycie najważniejszych zależności. Podstawowym wnioskiem płynącym z analizy wykresu 5.11 jest znacząca zbieżność wyników testów wydajności, które w małym stopniu zależą od przyjętej strategii wyboru punku czy atrybutu. Wyjątkiem jest tu strategia punktu zerowego, która okazała się zdecydowanie najmniej wydajna w przypadku wszystkich zbiorów danych. Prostym wyjaśnieniem tej sytuacji jest operowanie zawsze tylko na jednym punkcie referencyjnym, co okazało się mniej wydajne od wykorzystywania par punktów wybieranych w testach pozostałych strategii.

zbior	liczba punktów ref.	najwęższe dziedziny			najszerze dziedziny		
		maksymalne punkty	quasi-środkowe punkty	minimalne punkty	maksymalne punkty	quasi-środkowe punkty	minimalne punkty
covtype (60 tys. punktów)	1	2739383	3002338	3048498	1903741	2257441	2059925
	2	1571500	1587420	1672770	1004050	1194450	993297
	3	1636508	1707643	1729537	835124	958013	836267
	4	1659058	1724558	1756694	817283	968913	797536
	5	1641693	1724861	1805285	812508	994004	816778
	8	1692766	1762228	1797520	832933	998623	825874
	11	1745372	1790664	1815893	818215	1000544	831042
	22	1923340	1995728	2031808	915148	1111745	906404
	33	1943538	2028887	2054822	915521	1117034	913362
	44	2066857	2155523	2186597	966462	1176992	940599
cup98 (60 tys. punktów)	55	1590600	1688376	1613253	1007353	1225178	985884
	1	1308733	1541835	1573685	989763	3005410	1224063
	2	778906	815969	841125	751078	2048630	781265
	3	797005	829467	845921	770236	1325531	737653
	4	790593	829241	856835	761621	1325132	732769
	5	783310	826990	879122	722085	1346638	735151
	8	806381	847947	865537	742562	1335765	744388
	11	817575	880489	901494	749591	1363991	767599
	22	910947	963098	973238	813632	1517945	833959
	33	924656	954153	988972	832509	1520732	862679
	44	955550	1035838	1038334	846828	1610104	884358
	55	976931	1002040	1012270	873786	1653233	905150

Tabela 5.11 Porównanie wydajności algorytmu TI-dNBC-Ref na zbiorach danych covtype i cup98 (po 60 tys. punktów) w zależności od liczby i strategii wyboru punktów referencyjnych.

Przy założeniu, że z racji powyższego faktu strategię punktu zerowego pomijamy w analizie, jako najmniej wydajne strategie kwalifikują się: strategia punktów losowych i punktów quasi-środkowych z najszerzej dziedzin. Jest to trend znacznie klarowniejszy niż w przypadku TI-DBSCAN-Ref, ale zgodny z wnioskami z poprzedniego podrozdziału opisującego wydajność TI-dNBC. Natomiast analogicznie, jak w powyższych analizach, jedną z najlepszych strategii dla wszystkich zbiorów danych jest wybór punktów maksymalnych z atrybutów o najszerzej dziedzinach.



Wykres 5.12 Porównanie wydajności algorytmu TI-dNBC-Ref na testowych zbiorach danych covtype i cup98 (po 60 tysięcy punktów) w zależności od liczby punktów referencyjnych i strategii ich wyboru. Wizualizacja danych z tabeli 5.11.

Jak można zauważyć w tabeli 5.11, przedstawione powyżej wnioski dotyczące wyboru najwydajniejszej strategii pozostają aktualne niezależnie od liczby punktów referencyjnych. W szczególności, najbardziej wydajną strategią pozostaje wybór punktów maksymalnych z najszerzej dziedzin. Główny wniosek z porównania wykresu 5.12 do analogicznych wykresów, obrazujących wyniki dla algorytmu TI-DBSCAN-Ref, to zdecydowany spadek wydajności strategii punktów quasi-środkowych z najszerzej dziedzin. W przypadku zbioru covtype wydajność strategii preferujących atrybuty z wąską dziedziną jest dużo niższa z racji przewagi mało zróżnicowanych atrybu-

tów binarnych. Jednak nawet przy takiej strategii, przy odpowiednio dużej liczbie punktów referencyjnych (by na koniec brane pod uwagę były także atrybuty o szerszych dziedzinach) wydajność jest większa niż w przypadku omawianej strategii punktów quasi-środkowych z maksymalnych dziedzin. Oznacza to, że sprawdzenie nawet 55 odległości referencyjnych jest wydajniejsze od obliczania zbędnych rzeczywistych odległości pomiędzy punktami. Potwierdza to umiarkowany spadek wydajności, następujący przy zwiększaniu liczby punktów referencyjnych, powyżej liczby pięciu punktów.

5.4.3. Ocena wydajności

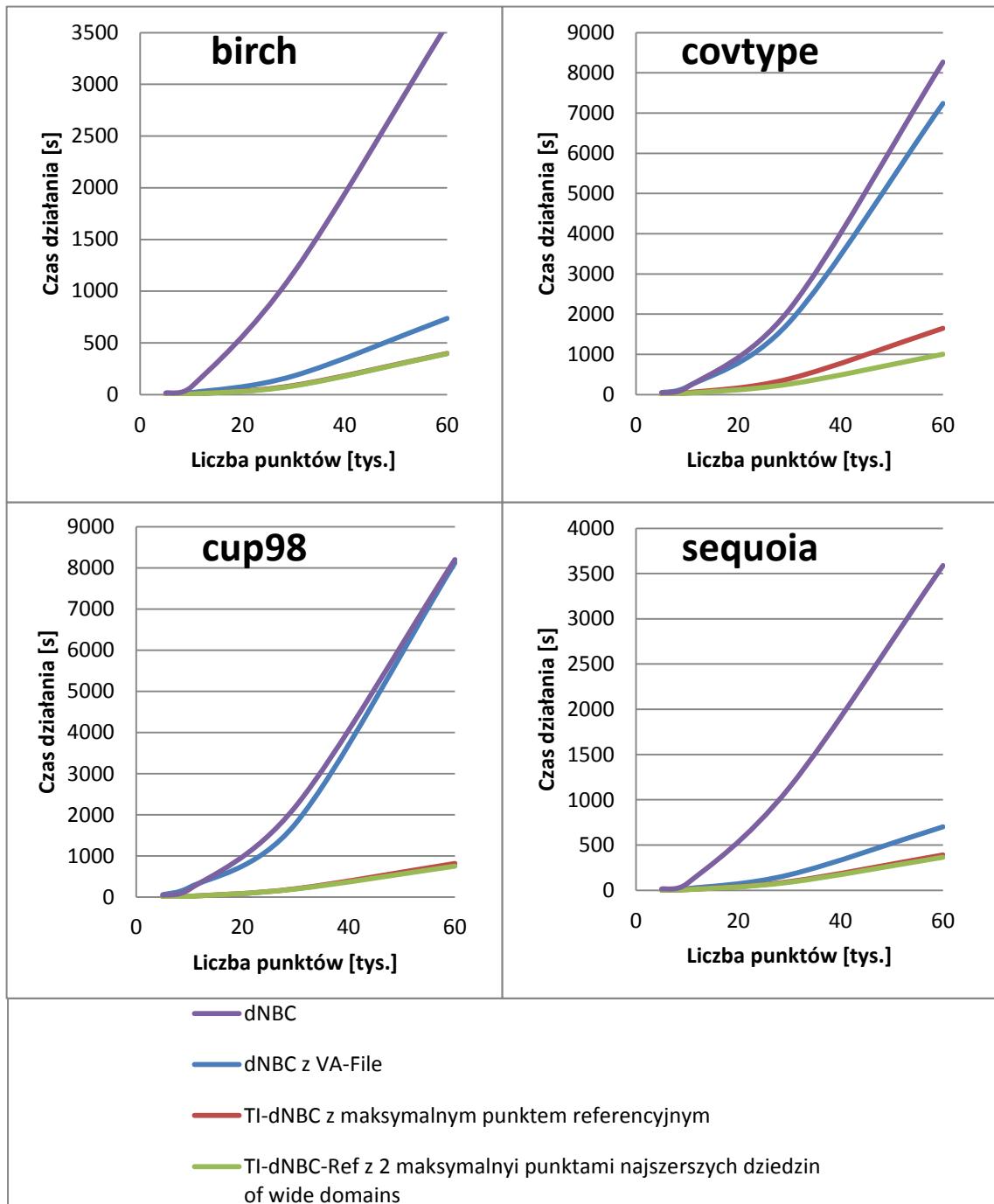
Najbardziej interesujące, bezpośrednie porównanie wydajności algorytmu dNBC ze zmodyfikowanymi wersjami wykorzystującymi VA-File i nierówność trójkąta przedstawia wykres 5.13. Podobnie jak w przypadku analogicznego porównania dla DBSCAN, wyraźna jest przewaga algorytmów wykorzystujących nierówność trójkąta. Mimo iż, zgodnie z tabelą 5.12, etap przygotowywania i sortowania danych w TI-dNBC może być kilkukrotnie dłuższy od budowania struktur VA-File, to nie ma to wpływu na ostateczny czas działania. Większa wydajność metody opartej na nierówności trójkąta nie tylko rekompensuje dłuższy czas przygotowań, ale co więcej, pozwala przyspieszyć etap właściwego grupowania. W przypadku dwuwymiarowych zbiorów birch i sequoia, znaczący wzrost wydajności osiągany jest już po zastosowaniu VA-File. TI-dNBC (oparty na maksymalnych punktach referencyjnych) wciąż pozwala dwukrotnie zwiększyć wydajność. Największy wzrost wydajności widoczny jest jednak w przypadku wielowymiarowych zbiorów danych covtype i cup98, dla których zastosowanie VA-File jedynie nieznacznie przyspiesza proces grupowania. Natomiast praktycznie niezależny od liczby wymiarów warunek na odległości referencyjne pozwala kilkukrotnie przyspieszyć działanie algorytmu. Warto jednak odnotować, że w stosunku do analogicznego porównania dla algorytmów TI-DBSCAN i TI-DBSCAN-Ref, w tym przypadku zastosowanie więcej niż jednego punktu referencyjnego ma mniejszy wpływ na wzrost wydajności. W większości przypadków, zastosowanie dodatkowych punktów referencyjnych nieznacznie tylko

skraca czas działania algorytmu. Oznacza to, że w przypadku wyznaczania k-sąsiedztwa dodatkowy narzut na obliczenie i sprawdzenie kolejnych odległości referencyjnych nie powoduje znacznego ograniczenia liczby punktów, dla których obliczane są rzeczywiste odległości.

zbior	liczba punktów	DBSCAN	DBSCAN z R*-drzewem		TI-DBSCAN z maksymalnym punktem referencyjnym		TI-DBSCAN-Ref z dwoma maksymalnymi punktami najszerzych dziedzin	
			łącznie	przygotowania	łącznie	przygotowania	łącznie	przygotowania
birch	5000	16015	16	4500	0	860	16	891
	10000	75172	16	18594	15	5984	47	6891
	30000	1179020	15	180906	63	88500	125	105859
	60000	3579800	15	736687	140	398078	250	489375
covtype	5000	43704	15	49484	15	10406	46	9687
	10000	182203	31	195031	47	46031	78	50297
	30000	2122160	62	1805330	172	393563	250	359890
	60000	8267910	125	7238700	359	1648920	547	1507630
cup98	5000	44297	16	58579	31	4562	32	6000
	10000	187844	16	235047	47	20156	78	27672
	30000	2200880	78	1784910	172	203109	265	269293
	60000	8202380	141	8124110	375	817219	547	1083890
sequoia	5000	15125	0	4453	16	1829	15	1968
	10000	75547	0	18391	16	8485	31	9266
	30000	1138060	0	171875	62	96297	125	105406
	60000	3589050	15	701984	141	390797	266	440094

Tabela 5.12 Porównanie wydajności podstawowej wersji algorytmu dNBC bez optymalizacji z wersją wykorzystującą VA-File, a także wersjami TI-dNBC z maksymalnym punktem referencyjnym i TI-dNBC-Ref z dwoma maksymalnymi punktami referencyjnymi z najszerzych dziedzin.

5. BADANIA EKSPERYMENTALNE



Wykres 5.13 Porównanie wydajności podstawowej wersji algorytmu dNBC bez optymalizacji z wersją wykorzystującą VA-File, a także wersjami TI-dNBC z maksymalnym punktem referencyjnym i TI-dNBC-Ref z dwoma maksymalnymi punktami referencyjnymi z najszerzszymi dziedzinach.

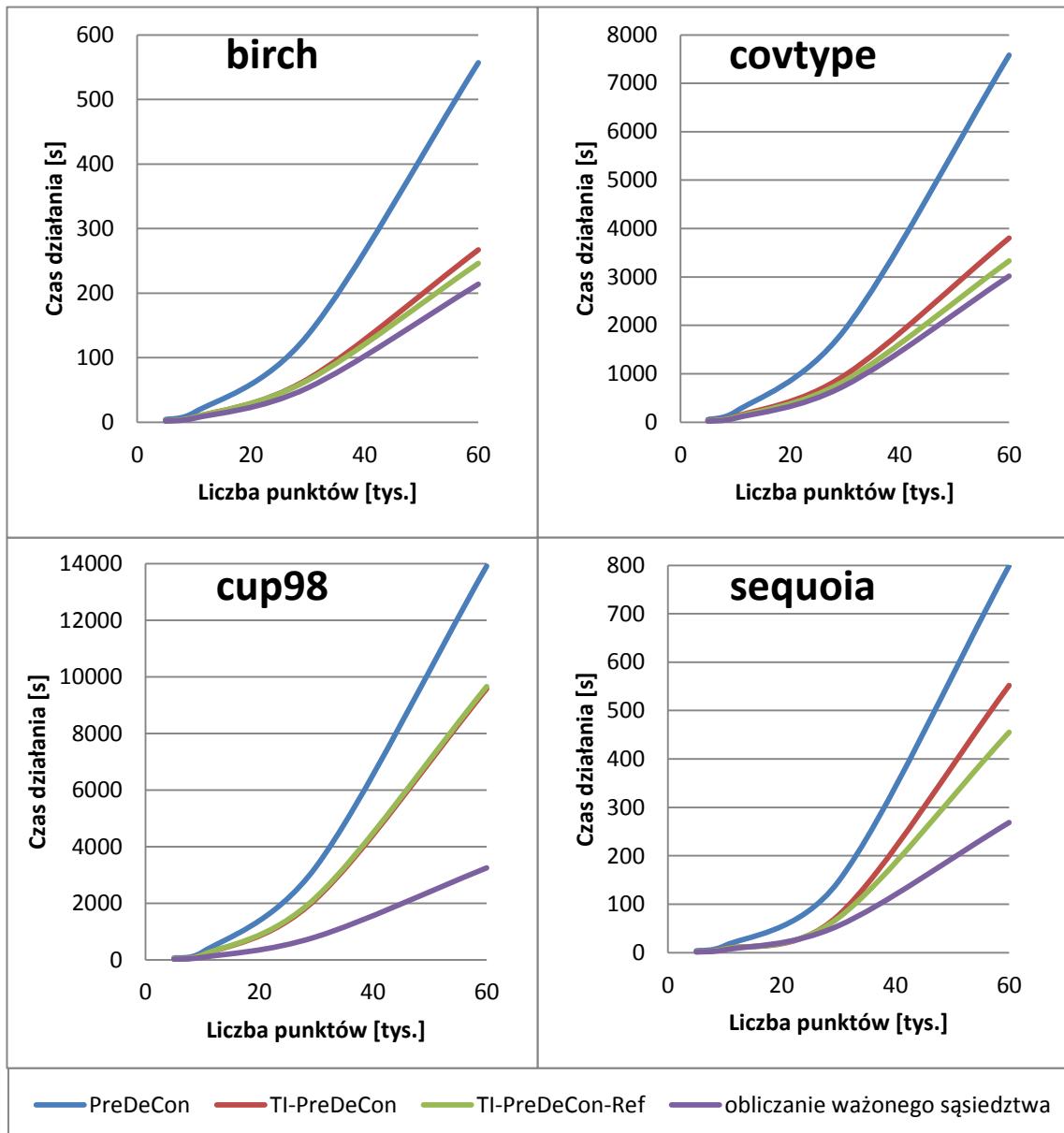
5.5. TESTY ALGORYTMU PREDECON

Już w teoretycznej analizie algorytmu TI-PreDeCon przedstawionej w rozdziale 3.2.3 zauważyłem, że z racji specyficznie obliczanych ważonych odległości możliwość wykorzystania nierówności trójkąta w celu optymalizacji algorytmu PreDeCon jest ograniczona. Wprowadzony przez autorów algorytm wzór na podobieństwo, w którym poszczególne atrybuty ważone są wartościami wektora W_0 wyznaczanego niezależnie dla danego punktu, uniemożliwia wykorzystanie nierówności trójkąta do usprawnienia wyszukiwania ważonego sąsiedztwa. Usprawnienie można jednak zastosować w pierwszym etapie algorytmu, w którym podczas wyznaczania wektorów W_0 , analogicznie jak w przypadku DBSCAN wyszukiwane są epsilonowe sąsiedztwa.

zbior	liczba punktów	PreDeCon			TI-PreDeCon z maksymalnym punktem referencyjnym			TI-PreDeCon-Ref z dwoma maksymalnymi punktami najszerzych dziedzin		
		oblicza-nie W_0	obli-czanie $N_\epsilon^{w_0}$	łącznie	oblicza-nie W_0	oblicza-nie $N_\epsilon^{w_0}$	łącznie	oblicza-nie W_0	oblicza-nie $N_\epsilon^{w_0}$	łącznie
birch	5000	2297	1781	4093	375	1719	2125	437	1656	2141
	10000	8703	6156	14922	1516	6078	7687	1328	6172	7609
	30000	81750	53250	135000	11688	54953	66766	8109	56531	64812
	60000	343140	213797	556937	48828	217969	267047	25468	220328	246171
covtype	5000	32969	21375	54344	8953	21469	30438	5922	21500	27469
	10000	130688	84594	215282	32734	84875	117687	18828	84797	103719
	30000	1158690	755875	1914560	205328	764813	970375	94578	761344	856234
	60000	4563590	3018280	7581880	753578	3050140	3804220	284375	3048030	3333080
cup98	5000	44078	22485	66563	20828	22640	43500	21313	22687	44047
	10000	182641	90437	273078	85610	91172	176860	85406	91031	176531
	30000	2477700	814657	3292360	1337060	819157	2156450	1395130	817266	2212700
	60000	10660400	3250920	13911300	6308860	3267450	9576800	6389050	3265405	9655110
sequoia	5000	2312	1438	3750	468	1500	1984	313	1500	1828
	10000	8515	5828	14406	2062	6000	8140	1407	6031	7531
	30000	91078	55578	147203	19906	56703	77156	13328	58015	71953
	60000	372531	268859	799093	84265	239001	551906	54547	240859	455156

Tabela 5.13 Porównanie wydajności podstawowej wersji algorytmu PreDeCon z zoptymalizowanymi wersjami TI-PreDeCon i TI-PreDeCon-Ref. Poza czasem działania pełnego procesu zostały zaprezentowane czasy dwóch głównych etapów: pierwszy etap obejmuje wyznaczanie standardowego epsilonowego sąsiedztwa w celu wyznaczenia wektora W_0 , który wykorzystywany jest w drugim etapie obliczania ważonego sąsiedztwa.

5. BADANIA EKSPERYMENTALNE



Wykres 5.14 Porównanie wydajności podstawowej wersji algorytmu PreDeCon z opartą na maksymalnym punkcie referencyjnym wersją TI-PreDeCon i TI-PreDeCon-Ref z dwoma maksymalnymi punktami referencyjnymi z najszerzej dziedzin. Kolorem fioletowym zaznaczony został, stały dla wszystkich wersji algorytmu, czas etapu wyznaczania ważonego sąsiedztwa

W przypadku algorytmu PreDeCon skupiłem się tylko na analizie wpływu zastosowania nierówności trójkąta. Modyfikacja pierwszego etapu algorytmu PreDeCon jest na tyle analogiczna, że zrezygnowałem z przedstawiania tu wyników testów dla wszystkich strategii wybierania punktów referencyjnych. Aktualne pozostają wszystkie wnioski z rozdziału 5.3. Zgodnie z nimi, do testów porównawczych wybrałem strategię maksymalnych punktów z najszerzej dziedzin. Wyniki testów zilustrowa-

ne na wykresie 5.14 pokazują wzrost wydajności, związany z zastosowaniem nierówności trójkąta, lecz jak można zauważyć, jest on zdecydowanie mniejszy niż ten obserwowany w przypadku analizowanych wcześniej algorytmów DBSCAN czy NBC. Przedstawiona w tabeli 5.13, dokładniejsza analiza czasów trwania głównych etapów algorytmów pokazuje, że znaczącą część czasu działania algorytmu stanowi etap obliczania ważonego sąsiedztwa, na który nie mają wpływu analizowane modyfikacje. Po naniesieniu czasów trwania tego etapu na wykresie 5.14 można zauważyć, że zastosowanie optymalizacji znaczaco wpływa na wydajność pierwszego etapu. Wzrost wydajności nie jest taki sam dla wszystkich testowych zbiorów danych. Różnica w przypadku zbiorów birch i covtype jest zdecydowanie bardziej zauważalna niż w przypadku cup98, co potwierdza analogię do przedstawionych w rozdziale 5.3.3 wyników wydajności indeksu w algorytmie DBSCAN.

6. PODSUMOWANIE

Podsumowując, przeprowadzona analiza wykorzystującej nierówność trójkąta opptymalizacji algorytmów gęstościowego grupowania danych potwierdza znaczące zwiększenie ich wydajności. Jak zaprezentowałem w pracy dyplomowej, wykorzystanie nierówności trójkąta pozwala znaczaco ograniczyć liczbę kosztownych operacji obliczania rzeczywistych odległości pomiędzy punktami. Zysk, nie tylko w pełni rekompensuje koszt sortowania zbioru danych według odległości do punktu referencyjnego, ale także zdecydowanie wpływa na zmniejszenie czasu trwania pełnego procesu grupowania. Jak potwierdzają testy, nie chodzi tu o zmniejszenie czasu o połowę, lecz nawet o rzędy wielkości. Co istotne, różnica w wydajności algorytmów jest tym większa, im większy jest zbiór danych. Jeszcze bardziej niż od liczby obiektów, jest ona zależna od liczby atrybutów. Duża liczba wymiarów, która stanowi wyzwanie dla porównywanych w testach indeksów przestrzennych, w wersjach wykorzystujących nierówność trójkąta nie wpływa na proces szacowania odległości, a jedynie na czas operacji obliczania rzeczywistych odległości do punktów referencyjnych i pomiędzy punktami, których algorytm nie wykluczył ze zbioru kandydatów na sąsiadów. Wywoływane dla każdego punktu funkcje wyznaczania sąsiedztwa, wykorzystując obliczone odległości do punktów referencyjnych ograniczają zbiór kandydatów, a tym samym ograniczają zależność algorytmu od liczby wymiarów bardziej od porównywanych indeksów przestrzennych R*-drzewo i VA-File.

Jak potwierdziła przeprowadzona analiza, wykorzystanie nierówności trójkąta pozwala przyspieszyć działanie zarówno, stanowiącego podstawę DBSCAN, procesu wyszukiwania epsilonowego sąsiedztwa, jak i bardziej złożonego procesu wyszukiwania k-sąsiedztwa z algorytmu NBC. W drugim przypadku, nieznajomość wartości epsilon a`priori komplikuje sposób wykorzystania nierówności trójkąta, przez co osiągany wzrost jest mniejszy, lecz wciąż jest on znaczący, w szczególności dla wielowymiarowych danych. Natomiast modyfikacja opartego na podprzestrzeniach algo-

6. PODSUMOWANIE

rytmu PreDeCon, przynosi ograniczony wzrost wydajności. Przyspieszony może być bowiem tylko pierwszy etap, obejmujący wyznaczania standardowych epsilonowych sąsiedztw każdego punktu. Nierówność trójkąta nie może być w prosty sposób wykorzystana w, stanowiącym podstawę drugiego etapu algorytmu, procesie wyszukiwania ważonego sąsiedztwa, w którym wagi atrybutów podczas obliczania odległości różnią się dla każdego punktu. Wciąż stanowi to problem czekający na rozwiązanie.

Kolejny istotnym celem przeprowadzonych testów była analiza wydajności zaproponowanych strategii wyboru punktów referencyjnych i ich liczby. W tym celu, jako dane testowe wybrałem cztery zbiory danych o różnych charakterystykach. Są to powszechnie stosowane w badaniach algorytmów eksploracji danych zbiory testowe: KKD-cup98, Sequoia2000, Covtype i Birch. Pozwoliły one potwierdzić przypuszczenia, że wydajność poszczególnych strategii zależy od charakterystyki zbioru danych, w tym szerokości dziedzin atrybutów i rozłożenia ich wartości. Najbardziej uniwersalnymi i wydajnymi punktami referencyjnymi okazały skrajne punkty dziedziny zbioru danych, w szczególności punkt maksymalny. W przypadku wyboru wielu punktów referencyjnych, najwydajniejszą strategią okazał się wybór punktów o maksymalnych wartościach atrybutów o najszerzych dziedzinach. Warto jednak zauważyć, że o ile wykorzystanie drugiego punktu referencyjnego zawsze oznacza zwiększenie wydajności, o tyle dalsze zwiększanie liczby punktów referencyjnych może negatywnie wpływać na wydajność. W przypadku obydwu algorytmów wykorzystujących wiele punktów referencyjnych, TI-DBSCAN-Ref i TI-NBC-Ref, optymalna liczba punktów referencyjnych, dla zbiorów danych o ponad 50 wymiarach, oscylowała wokół niewielkiej liczby od trzech do pięciu punktów. Jednak, co istotne, nawet w przypadku wyboru najmniej wydajnej strategii czy liczby punktów referencyjnych, algorytmy wykorzystujące nierówność trójkąta były wciąż wydajniejsze od wersji wykorzystujących indeksy przestrzenne.

Tak optymistyczne wnioski z przeprowadzonych testów zachęcają do dalszej analizy możliwości wykorzystania nierówności trójkąta zarówno w wymienionych w pracy, jak i innych algorytmach gęstościowego grupowania danych. Ciekawym polem do dalszych badań wydaje się w szczególności algorytm wyszukiwania k-sąsiedztwa, a jednym z pomysłów jest wykorzystanie technik heurystycznych do wstępnego sza-

cowania wartości epsilon. Podobne techniki, opracowywane na podstawie wyciągniętych w pracy dyplomowej wniosków, zastosowane mogą być w celu wyboru najlepszych punktów referencyjnych, dopasowanych do charakterystyki zbioru danych.

Bez względu na możliwości dalszego rozwoju, wykorzystanie szacowania opartego na nierówności trójkąta gwarantuje znaczący wzrost wydajności, praktycznie bez dodatkowych kosztów, takich jak np. znaczaco zwiększone zużycie pamięci. Ograniczeniem zastosowań analizowanej modyfikacji może okazać się konieczność sortowania wejściowego zbioru danych. Niestety, jak wykazały testy, zastosowanie dodatkowej struktury indeksu obniża wydajność całego rozwiązania. W dalszych badaniach warto rozważyć i poddać analizie także inne implementacje, badając ich wydajność także na zbiorach danych na tyle dużych, by uwzględnić czasy dostępu do pamięci masowej.

Podsumowując, uzyskane w pracy dyplomowej wyniki analizy wykorzystania nierówności trójkąta do usprawnienia algorytmów gęstościowego grupowania danych potwierdzają znaczące zwiększenie wydajności, w szczególności w przypadku danych wielowymiarowych, które wciąż stanowią wyzwanie dla większości algorytmów gęstościowego grupowania danych i indeksów przestrzennych.

BIBLIOGRAFIA

- [1] Aggarwal C.C., Procopiuc C., "Fast Algorithms for Projected Clustering", Materiały z *ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'99)*, Philadelphia, 1999.
- [2] Agrawal R., Gehrke J., Gunopulos D., P Raghavan., "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications", Materiały z *ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'98)*, Seattle, 1998.
- [3] Beckmann N., Kriegel H.N., Schneider R., Seeger B., "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", Materiały z *ACM SIGMOD Int. Conference on Management of Data*, Atlantic City, 1990, pp. 322-331.
- [4] Blackard J.A. (1999, lipiec) The Forest CoverType dataset. [Online].
<http://ftp.ics.uci.edu/pub/machine-learning-databases/covtype/covtype.info>
- [5] Bohm C., Railing K., Kriegel H.P., Kroger P., "Density connected clustering with local subspace preferences", Materiały z *ICDM '04*, 2004, pp. 27-34.
- [6] Elkan C., "Using the triangle inequality to accelerate k-Means", Materiały z *ICML 2003*, 2003, pp. 147-153.
- [7] 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, pp. 226-231.

BIBLIOGRAFIA

- [8] Hadjieleftheriou M. (2010, grudzień) SpatialIndex Library (libspatialindex). [Online]. <http://libspatialindex.github.com/>
- [9] Han J., Kamber M., *Data Mining: Concepts and Techniques*.: Morgan Kaufman, 2001.
- [10] Hinneburg A., Aggarwal C., Keim D.A., "What is the Nearest Neighbor in High Dimensional Spaces", Materiały z *26th Int. Conf. on Very Large Databases (VLDB'00)*, Cairo, 2000, pp. 506-516.
- [11] Kailing K., Kriegel H.P., Kröger P., "Density-Connected Subspace Clustering for High-Dimensional Data", Materiały z *SIAM Int. Conf. on Data Mining (SDM'04)*, Lake Buena Vista, 2004.
- [12] Kaufman L., Rousseeuw P.J., *Finding Groups in Data: an Introduction to Cluster Analysis*.: John Wiley & Sons, 1990.
- [13] Kryszkiewicz M., Lasek P., "A Neighborhood-Based Clustering by Means of the Triangle Inequality", Materiały z *IDEAL 2010, Lecture Notes in Computer Science Volume 6283*, 2010, pp. 284-291.
- [14] Kryszkiewicz M., Lasek P., "TI-DBSCAN: Clustering with DBSCAN by Means of the Triangle Inequality", Materiały z *RSCTC 2010, Lecture Notes in Computer Science Volume 6086*, 2010, pp. 60-69.
- [15] Kryszkiewicz M., Skonieczny Ł., "Faster Clustering with DBSCAN", Materiały z *IIPWM'05*, Gdańsk, 2005, pp. 605-614.
- [16] Moore A.W., "The Anchors Hierarchy: Using the Triangle Inequality to Survive High Dimensional Data", Materiały z *UAI*, Stanford, 2000, pp. 397–405.
- [17] Nokia Corporation. (2012) Qt - cross-platform application and UI framework. [Online]. <http://qt.nokia.com/>

- [18] Parsa I. (1999, luty) The UCI KDD Archive: KDD Cup 1998 Data. [Online].
<http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html>
- [19] (2012) SQLite Webpage. [Online]. <http://www.sqlite.org/>
- [20] Stonebraker M., Frew J., Gardels K., Meredith J., "The SEQUOIA 2000 Storage Benchmark", Materiały z *ACM SIGMOD*, Washington, 1993, pp. 2-11.
- [21] Weber R., Schek H.J., Blott S., "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces", Materiały z *VLDB'98*, New York City, 1998, pp. 194-205.
- [22] Zhang T., Ramakrishnan R., Livny M., "BIRCH: A New Data Clustering Algorithm and its Applications", Materiały z *Data Mining and Knowledge Discovery*, 1 (2), 1997, pp. 141-182.
- [23] Zhou S., Zhoa Y., Guan J., Huang J.Z., "A Neighborhood-Based Clustering Algorithm", Materiały z *PAKDD*, 2005, pp. 361-371.

DODATEK A. OPIS PROGRAMU

Projekt zaimplementowany został w języku C++. Obejmuje on 17 wersji analizowanych w pracy algorytmów grupowania danych: DBSCAN, NBC i PreDeCon:

- *DBSCAN* – podstawowa wersja algorytmu DBSCAN zaimplementowana zgodnie z opisem z rozdziału 2.1;
- *DBSCAN with R*-tree* – wersja DBSCAN wykorzystująca indeks R*-drzewo zaimplementowany zgodnie z opisem z rozdziału 4.2.1;
- *TI-DBSCAN with sorting* – wykorzystująca nierówność trójkąta z jednym punktem referencyjnym wersja TI-DBSCAN oparta na sortowaniu oryginalnego zbioru danych, zgodnie z opisem z rozdziału 5.3.1;
- *TI-DBSCAN with index* – wykorzystująca nierówność trójkąta z jednym punktem referencyjnym wersja TI-DBSCAN oparta na dodatkowej liście posortowanych wskaźników, zgodnie z opisem z rozdziału 5.3.1;
- *TI-DBSCAN-Ref* – wykorzystująca nierówność trójkąta z wieloma punktami referencyjnymi wersja TI-DBSCAN-Ref zaimplementowana zgodnie z opisem z rozdziału 3.2.1;
- *NBC* - podstawowa wersja algorytmu NBC zaimplementowana zgodnie z opisem z rozdziału 2.2;
- *dNBC* - deterministyczna wersja algorytmu NBC zaimplementowana zgodnie z opisem z rozdziału 2.2;
- *NBC with VA-File* – wersja NBC wykorzystująca indeks VA-File zaimplementowany zgodnie z opisem z rozdziału 4.2.2;
- *dNBC with VA-File* – deterministyczna wersja dNBC wykorzystująca indeks VA-File zaimplementowany zgodnie z opisem z rozdziału 4.2.2;
- *TI-NBC* - wykorzystująca nierówność trójkąta z jednym punktem referencyjnym wersja TI-NBC zaimplementowana zgodnie z opisem z 3.2.2;

- *TI-dNBC* - wykorzystująca nierówność trójkąta z jednym punktem referencyjnym deterministyczna wersja TI-dNBC zaimplementowana zgodnie z opisem z rozdziału 3.2.2;
- *TI-NBC-Ref* - wykorzystująca nierówność trójkąta z wieloma punktami referencyjnymi wersja TI-NBC-Ref zaimplementowana zgodnie z opisem z rozdziału 3.2.2;
- *TI-dNBC-Ref* - wykorzystująca nierówność trójkąta z wieloma punktami referencyjnymi deterministyczna wersja TI-dNBC-Ref zaimplementowana zgodnie z opisem z rozdziału 3.2.2;
- *PreDeCon* - podstawowa wersja algorytmu PreDeCon zaimplementowana zgodnie z opisem z rozdziału 2.3;
- *PreDeCon with R*-tree* - wersja PreDeCon wykorzystująca indeks R*-drzewo zaimplementowany zgodnie z opisem z rozdziału 4.2.1;
- *TI-PreDeCon* - wykorzystująca nierówność trójkąta z jednym punktem referencyjnym wersja TI-PreDeCon zaimplementowana zgodnie z opisem z rozdziału 3.2.3;
- *TI-PreDeCon-Ref* - wykorzystująca nierówność trójkąta z wieloma punktami referencyjnymi wersja TI-PreDeCon-Ref zaimplementowana zgodnie z opisem z rozdziału 3.2.3.

W celu wygodnego wywoływania poszczególnych algorytmów, różniących się parametrami wejściowymi oraz udostępnienia wizualizacji wyników grupowania, stworzony został także interfejs graficzny wykorzystujący bibliotekę Qt [17]. Jego opisowi poświęcony został kolejny rozdział.

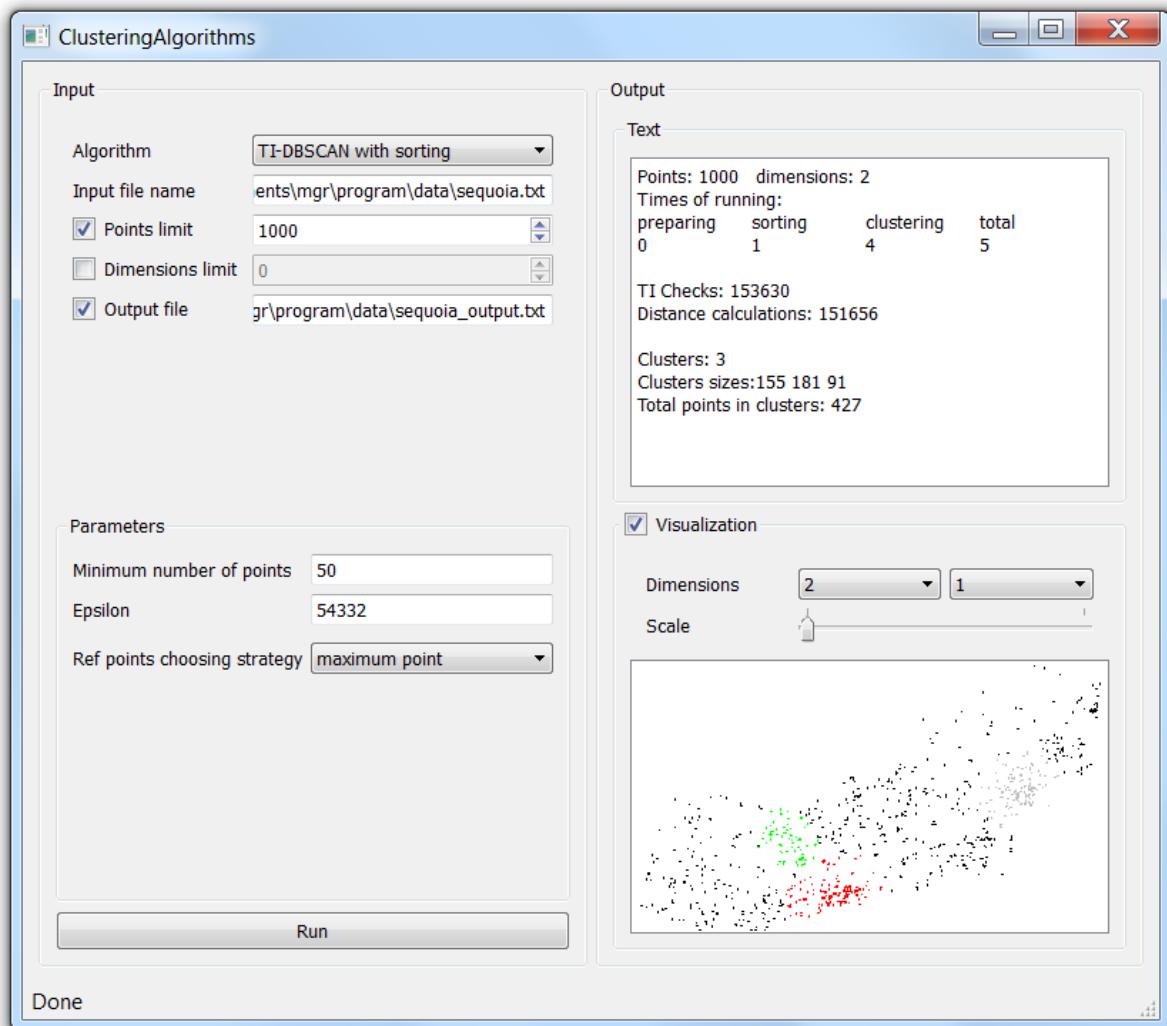
I. INTERFEJS UŻYTKOWNIKA

Przedstawiony na ilustracji A.1 interfejs użytkownika podzielić można na dwie główne części. Lewa strona okna umożliwia użytkownikowi wybór algorytmu oraz wszystkich parametrów wywołania. Użytkownik podać musi ścieżkę do pliku

ze zbiorem danych do grupowania, a opcjonalnie może także ograniczyć liczbę wczytywanych punktów lub atrybutów oraz określić czy generowany ma być plik z wynikiem, a jeśli tak to w jakiej lokalizacji (domyślnie jest to lokalizacja pliku wejściowe, a nazwa różni się sufiksem *_output*). Dodatkowo, w zależności od wybranego algorytmu dostosowywany jest formularz parametrów wywołania, który może zawierać:

- minimalną liczbę punktów;
- epsilon;
- liczbę sąsiadów *k*;
- liczbę bajtów wektorów indeksu VA-File;
- maksymalną liczbę gęstych wymiarów;
- maksymalną wariancję niegęstego wymiaru;
- wagę gęstych wymiarów w wektorze w_0 ;
- liczbę punktów referencyjnych;
- strategię wyboru punktów referencyjnych:
 - *zerowych* – zerowe wartości wszystkich atrybutów;
 - *minimalnych* – minimalne wartości dziedzin wszystkich lub wybranych atrybutów;
 - *środkowych* – środkowe wartości dziedzin wszystkich lub wybranych atrybutów;
 - *maksymalnych* – maksymalne wartości dziedzin wszystkich lub wybranych atrybutów;
 - *losowych* – losowo wybieranych ze zbioru danych;
- strategię wyboru znaczących atrybutów:
 - pierwszych, zgodnie z kolejnością w oryginalnym zbiorze danych;
 - wybieranych losowych;
 - o najwięzszych dziedzinach wartości;
 - o najszerznych dziedzinach wartości.

Dokładne opisy poszczególnych parametrów zamieszczono w rozdziałach 2 i 3.



Ilustracja A.1 Interfejs użytkownika dostępny w aplikacji clusteringAlgorithms.

Prawa część okna umożliwia prezentację wyników działania algorytmów. W części tekstowej, poza statystykami zbioru danych (liczba wczytanych punktów i wymiarów) i najbardziej interesującymi podczas testów czasami trwania najważniejszych etapów algorytmów, prezentowane są także statystyki wyniku grupowania, takie jak liczba grup i ich rozmiary. Wynik grupowania może zostać zwizualizowany, w dwóch dowolnie wybranych wymiarach, w postaci wykresu, na którym punkty należące do tej samej grupy oznaczone są tym samym kolorem, a szum reprezentowany jest przez kolor czarny.

Dodatkowo, w celu wygodnego i sprawnego przeprowadzania testów zaimplementowanych wersji algorytmów, stworzyłem alternatywną w stosunku do interfejsu graficznego klasę *ClusteringAlgorithmsTests*. Umożliwia ona iteracyjne wywoływanie algorytmów z łatwo konfigurowalnymi parametrami. Poza logowaniem postępu na standardowym wyjściu, zapisuje ona wszystkie parametry wywołania algorytmów oraz czasy poszczególnych etapów grupowania w prostej bazie danych SQLite [19], skąd mogą ona zostać pobrane w dowolnej postaci do dalszych analiz. Klasa zawiera scenariusze testów wykorzystane w pracy dyplomowej, które mogą być dowolnie rozbudowywane.

II. DANE WEJŚCIOWE I WYJŚCIOWE

Program przyjmuje na wejściu pliki tekstowe, w których każdy punkt zapisany jest w nowej linii, a poszczególne wartości atrybutów oddzielone są znakiem tabulatora. Przykładowy czterowymiarowy zbiór zawierający siedem punktów przedstawiony zostanie jako:

1942954	573516	1895163	245383
1351711	1141088	1530655	810049
1704930	925709	1772615	895403
1718240	730885	1952223	539742
1622121	1060107	1762224	950693
2004891	312864	1691503	1066231
1988456	133962	1694123	1110700

Wynik działania aplikacji może zostać zapisany do analogicznego pliku tekściego⁴ z wartościami rozdzielonymi tabulatorami, rozszerzonego o dodatkową, umieszczoną jako pierwsza, kolumnę z identyfikatorami grup, do których przydzielone zostały dane punktu. W szczególności, wartość -1 oznacza, że jest to szum, czyli punkt nienależący do żadnej grupy. Przykładowy wynik grupowania powyższego zbioru byłby zapisany jako:

⁴ Warto przy tym zaznaczyć, że kolejność punktów w pliku wejściowym i wyjściowym może ulec zmianie.

DODATEK A. OPIS PROGRAMU

2	1942954	573516	1895163	245383
2	1351711	1141088	1530655	810049
1	1704930	925709	1772615	895403
2	1718240	730885	1952223	539742
1	1622121	1060107	1762224	950693
-1	2004891	312864	1691503	1066231
-1	1988456	133962	1694123	1110700

Funkcja odczytu i zapisu danych jest częścią interfejsu, a nie poszczególnych algorytmów. Oznacza to, że aplikacja może zostać w prosty sposób dostosowana do obsługi innych formatów plików.