Raport

Bartłomiej Jańczak

# Opis zadania

1. Wyszukiwanie Euklidesowego otoczenia Eps i k sąsiadów dla wszystkich (lub zadanej liczby) wektorów i wszystkich (lub zadanej liczby) wymiarów:
   1. bez wykorzystania nierówności trójkąta
   2. z sortowaniem zbioru wektorów/indeksu na zbiór wektorów oraz:
      1. z wykorzystaniem nierówności trójkąta i 1 lub więcej wektorów referencyjnych (np. [0,0,0,1,0,0], [0,0,0,1/2,0,0], [0,0,0,1 1/2,0,0], [0,0,0,10,0,0]),
      2. z wykorzystaniem rzutu(ów) na os(ie),
      3. z wykorzystaniem wektorów referencyjnych i dodatkowych rzutów na osie,
2. Wyszukiwanie kosinusowego otoczenia Eps i k sąsiadów dla wszystkich (lub zadanej liczby) wektorów i wszystkich (lub zadanej liczby) wymiarów poprzez przekształcenie oryginalnego problemu do problemu z odległością Euklidesowa:
   1. bez wykorzystania nierówności trójkąta,
   2. z sortowaniem zbioru wektorów/indeksu na zbiór wektorów oraz:
      1. z wykorzystaniem nierówności trójkąta i 1 lub więcej wektorów referencyjnych (np. [0,0,0,1,0,0], [0,0,0,1/2,0,0], [0,0,0,1 1/2,0,0], [0,0,0,10,0,0]),
      2. z wykorzystaniem rzutu(ów) na os(ie),
      3. z wykorzystaniem wektorów referencyjnych i dodatkowych rzutów na osie
3. Klasyfikacja KNN z sortowaniem zbioru wektorów/indeksu na zbiór wektorów oraz:
   1. z uplasowaniem się liniowym/binarnym oraz:
      1. wykorzystaniem punktu/ów referencyjnych do szacowania odległości,
      2. wykorzystaniem rzutu/(ów) na os do szacowania odległości,
   2. bez uplasowania się oraz:
      1. z wykorzystaniem punktu/ów referencyjnych do szacowania odległości,
      2. z wykorzystaniem rzutu/ów na oś do szacowania odległości,
4. Wyszukiwanie euklidesowego/cosinusowego otoczenia Eps i k sąsiadów dla wszystkich (lub zadanej liczby) wektorów i wszystkich (lub zadanej liczby) wymiarów przy pomocy drzewa vantage point:
   1. z wykorzystaniem mediany jako kryterium podziału,
   2. z wykorzystaniem sąsiednich wartości mediany jako kryterium podziału.

# Założenia

//TODO?

# Opis formy danych wejściowych i wyjściowych

Na dane wejściowe składają się 3 rodzaje plików:

* Plik parametrów uruchomienia aplikacji – plik tekstowy zawierający rekordy postaci „*nazwa\_parametru*=*wartość\_parametru*”,
* Plik parametrów uruchomienia algorytmu – plik tekstowy zawierający rekordy postaci „*nazwa\_parametru*=*wartość\_parametru*”,,
* Plik danych wejściowych – plik tekstowy zawierający zbiór wektorów w formacie gęstym lub rzadkim.

Na dane wyjściowe składają się 3 rodzaje plików:

* Raport wykonania uruchomienia algorytmu – plik tekstowy zawierający:
  + wartości parametrów, z którymi uruchomiono algorytm,
  + czasy wykonania poszczególnych kroków algorytmu,
  + wyniki wykonania algorytmu.
* Raport wykonania uruchomienia aplikacji – plik \*.csv, zawierający dane raportów z wykonania uruchomień algorytmów z wyłączeniem wyników wykonania algorytmu. Każdy rekord pliku \*.csv odpowiada pewnemu raportowi z uruchomienia algorytmu w ramach jednego uruchomienia aplikacji.
* Oczyszczony raport wykonania uruchomienia aplikacji – plik \*.csv, zawierający dane raportów z wykonania uruchomień algorytmów z wyłączeniem wyników wykonania algorytmu. Każdy rekord pliku \*.csv odpowiada serii (ilość powtórzeń) uruchomień algorytmu. Uruchomienie algorytmu opisane jest plikiem parametrów uruchomienia algorytmu. Wielkość serii opisana jest parametrem **test\_repeats** w pliku parametrów uruchomienia aplikacji. W rekordzie pliku \*.csv każda wartość odpowiadająca czasowi wykonania pewnego kroku K algorytmu A w serii S jest tworzona w następujący sposób: spośród wszystkich wartości czasowych wykonania K w A w S usuwane są jedna maksymalna i jedna minimalna wartość czasowa, wartość średnia z pozostałych czasowych wartości stanowi wartość czasową wykonania pewnego kroku K algorytmu A w rekordzie pliku \*.csv.

# Dokumentacja użytkowa

Rozdział ten zawiera opisy:

1. struktury plików, z których korzysta implementacja,
2. pliku parametrów uruchomienia aplikacji,
3. pliku parametrów uruchomienia algorytmu,
4. znaczenia czasów wykonania dla uruchomień poszczególnych algorytmów,
5. przygotowanych zbiorów danych,
6. uruchomienia aplikacji.

## Struktura plików

Do dokumentacji załączono plik *Implementacja.zip* o następującej strukturze folderów:

* Implementacja:
  + algorithms\_engine\_properties
  + code
    - hdr
    - src
  + datasets
  + logs
  + properties

Zawartość folderów opisuje Tabela 1 .

|  |  |
| --- | --- |
| Nazwa folderu | Zawartość folderu |
| implementacja | Foldery **algorithms\_engine\_properties**, code, **datasets**, **logs**, **properties** oraz plik wykonywalny implementacja.exe. |
| algorithms\_engine\_properties | Plik parametrów **algorithms\_engine\_properties.txt**. Opis pliku parametrów uruchomienia aplikacji znajduje się w rozdziale [Rozdział 4.2](#_Plik_parametrów_uruchomienia_1). |
| code | Foldery **hdr** i **src**. |
| hdr | Pliki nagłówkowe **\*.h**. |
| src | Pliki źródłowe **\*.cpp**. |
| datasets | Pliki ze zbiorami danych **\*.txt**. |
| logs | Pliki raportów wykonania algorytmu **\*.txt**, oraz pliki raportów wykonania aplikacji **\*.csv**. Krótki opis plików raportu znajduje się w rozdziale [Rozdział 3](#_Opis_formy_danych). |
| properties | Pliki parametrów uruchomienia algorytmu. Opis pliku parametrów uruchomienia algorytmu znajduje się w rozdziale [Rozdział 4.3](#_Plik_parametrów_uruchomienia). |

Tabela 1: Opis zawartości folderów pliku implementacja.zip

Plik wykonywalny *implementacja.exe* jest programem implementującym odmiany algorytmów, spełniającym wymagania wymienione w rozdziale [Rozdział 1](#_Opis_zadania). Program skompilowany jest dla środowiska WIN32 z podniesioną flagą LARGEADDRESSAWARE, która umożliwia aplikacji użycie do 3GB pamięci wirtualnej. Uruchomienia wykonywane są na środowisku Windows 7 x64 z procesorem Intel® i7™ 950 z dostępną pamięcią RAM równą 6GB.

## Plik parametrów uruchomienia aplikacji

W folderze **algorithms\_engine\_properties** znajduje się plik parametrów uruchomienia aplikacji *algorithms\_engine\_properties.txt*. Tabela 2 opisuje znaczenie poszczególnych parametrów pliku parametrów.

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

Tabela 2: Opis znaczenia parametrów pliku algorithms\_engine\_properties.txt

## Plik parametrów uruchomienia algorytmu

W folderze **properties** znajdują się pliki uruchomienia algorytmu. Każdy plik parametrów definiuje uruchomienie algorytmu. Program kolejno wczytuje pliki parametrów i dla każdego z nich wykonuje zdefiniowany algorytm. Tabela 3 opisuje znaczenie poszczególnych parametrów pliku parametrów.

|  |  |
| --- | --- |
| Nazwa parametru | Opis |
| algorithm\_name | Nazwa algorytmu przyjmująca wartości: *\****dbscan** – algorytm DBSCAN. Wymaga zdefiniowania parametrów: *eps*, *min\_pts*.  **\*dbscan\_points\_elimination** – algorytm DBSCAN z eliminacją rozpatrzonych punktów. Wymaga zdefiniowania parametrów: *eps*, *min\_pts*.  \***ti\_dbscan** – algorytm TI-DBSCAN. Wymaga zdefiniowania parametrów: *eps*, *min\_pts*,  *reference\_point*, *reference\_point\_format*, *use\_dataset\_index\_access*.  \***ti\_dbscan\_ref** – algorytm TI-DBSCAN wykorzystujący wiele punktów referencyjnych do szacowania odległości. Wymaga zdefiniowania parametrów: *eps*, *min\_pts*,  *reference\_point*, *reference\_point\_format, use\_dataset\_index\_access*.  \***ti\_dbscan\_ref\_projection** – algorytm TI-DBSCAN wykorzystujący wiele punktów referencyjnych oraz rzuty na osie do szacowania odległości.  Wymaga zdefiniowania parametrów: *eps*, *min\_pts*, *reference\_point*, *reference\_point\_format*, *use\_dataset\_index\_access,* *projection\_dimensions*, *projection\_source\_sequence*. \***k\_neighborhood** – algorytm K-NEIGHBORHOOD. Sąsiedzi wyznaczani metodą brute-force liczenia odległości na zasadzie ‘każdy z każdym’. Wymaga zdefiniowania parametrów: *k*, *use\_dataset\_index\_access, classification\_subset\_factor*, *use\_binary\_placement*.  \***ti\_ k\_neighborhood** – algorytm TI-K-NEIGHBORHOOD. Wymaga zdefiniowania parametrów: *k*, *reference\_point*, *reference\_point\_format*, *use\_dataset\_index\_access, classification\_subset\_factor*, *use\_binary\_placement*.  \***ti\_k\_neighborhood\_ref** – algorytm TI-K-NEIGHBORHOOD wykorzystujący wiele punktów referencyjnych do szacowania odległości. Wymaga zdefiniowania parametrów: *k*, *reference\_point*, *reference\_point\_format, use\_dataset\_index\_access, classification\_subset\_factor*, *use\_binary\_placement*. \***ti\_k\_neighborhood\_ref\_projection** – algorytm TI-K-NEIGHBORHOOD wykorzystujący wiele punktów referencyjnych oraz rzuty na osie do szacowania odległości. Wymaga zdefiniowania parametrów: *k*, *reference\_point*, *reference\_point\_format*, *use\_dataset\_index\_access, classification\_subset\_factor*, *use\_binary\_placement*, *projection\_dimensions*, *projection\_source\_sequence*. \***vp\_tree** – algorytm wyszukujący k najbliższych sąsiadów. Buduje a następnie przeszukuje VANTAGE POINT TREE. Wymaga zdefiniowania parametrów: *k*, *p\_sample\_index*, *s\_sample\_index*, *classification\_subset\_factor*, *use\_binary\_placement,*  *search\_method,* use\_boundaries. |
| eps | Promień sąsiedztwa. |
| min\_pts | Minimalna liczba punktów w grupie. |
| k | Liczba sąsiadów. |
| use\_cosine\_similarity | Flaga definiująca rodzaj wyszukiwanego otoczenia przyjmująca wartości:  *\****true** – jeśli wyszukiwanie kosinusowego otoczenia, *\****false** – jeśli wyszukiwanie euklidesowego otoczenia. |
| dataset\_file\_format | Flaga definiująca format wczytywanych danych przyjmująca wartości:  *\****dense** – jeśli wczytywane wektory zapisane są w formacie gęstym,  *\****sparse** – jeśli wczytywane wektory zapisane są w formacie rzadkim. |
| dataset\_file\_path | Ścieżka dostępu do pliku ze zbiorem punktów, plik powinien się znajdować w katalogu **datasets**. |
| dataset\_dimension | Liczba definiująca największy wymiar każdego punktu jaki zostanie wczytany do pamięci programu. Wymiary punktu większe niż *dataset\_dimension* nie zostaną wczytane do pamięci programu. |
| dataset\_dimension\_value\_treshold | Współczynnik wyznaczający poziom istotności wartości wymiaru punktu. Jeśli wartość dowolnego wymiaru punktu będzie niższa od *dimension\_value\_treshold* to temu wymiarowi zostanie przypisana wartość 0. |
| dataset\_elements\_number | Liczba punktów zbioru *dataset\_file\_path*, które zostaną wczytane do pamięci programu jako zbiór punktów. |
| dataset\_internal\_format | Flaga definiująca wewnętrzny sposób przechowywania punktów w programie przyjmująca wartości: \***dense** – punkt przechowywany jest w formacie gęstym,  \***sparse** – punkt przechowywany jest w formacie rzadkim |
| reference\_point | Definicja punktów referencyjnych. Punkty referencyjne mogą być definiowane zarówno w formacie gęstym jak i rzadkim. Format według, którego *reference\_point*  będzie parsowany przechowywany jest w parametrze *reference\_point\_format.* Punkty referencyjne przechowywane są w programie w formacie zgodnym z parametrem *internal\_representation.*  Funkcje:  **[max]** – punkt referencyjny o maksymalnych wartościach poszczególnych wymiarów w danym zbiorze danych. ([max, max, …, max])  **[min]** – punkt referencyjny o minimalnych wartościach poszczególnych wymiarów w danym zbiorze danych. ([min, min, …, min])  **[n]** – punkt referencyjny postaci [n,n,…,n]  **[(a,b)\*]** – punkt referencyjny, tworzony na zasadzie powtarzania wzorca aż do maksymalnego wymiaru. Przykładowo dla: [(a,b)\*] punkt referencyjny byłby postaci [a,b,a,b,a,b,a,b,a,b,…],  a dla [(a,b,c,d)\*] punkt referencyjny byłby postaci [a,b,c,d,a,b,c,d,a,b,c,d,a,b,c,d,…]. **[rand]** – punkt referencyjny o losowych i nie większych niż maksymalne wartościach poszczególnych wymiarów w danym zbiorze danych.  **max** – maksymalna wartość wymiaru w danym zbiorze danych.  Przykładowe użycie dla przestrzeni dwuwymiarowej:  format gęsty:  [max,1] ,  [max,max] ⬄ [max].  format rzadki: (1 odpowiada pierwszemu wymiarowi)  [1,max,2,1],  [1,max,2,max] ⬄ [max].  **min** - minimalna wartość wymiaru w danym zbiorze danych.  Przykładowe użycie dla przestrzeni dwuwymiarowej: (analogicznie do max).  *reference\_point* może definiować więcej niż jeden punkt referencyjny, przykładowo: reference\_point = [0][min][max] |
| reference\_point\_format | Flaga definiująca format definicji *reference\_point* przyjmująca wartości: **\*dense** – definicja *reference\_point* w formacie gęstym,  **\*sparse** – definicja *reference\_point* w formacie rzadkim. |
| use\_dataset\_index\_access | Flaga definiująca sposób dostępu do zbioru wektorów przyjmująca wartości: **\*true** – dostęp do zbioru wektorów przez indeks. Jeśli wymagane jest sortowanie zbioru to sortowany jest indeks,  **\*false** – bezpośredni dostęp do zbioru wektorów lub dostęp do zbioru przez indeks jeśli wymaga tego implementacja (algorytmy DBSCAN z nierównością trójkąta). Jeśli wymagane jest sortowanie zbioru to sortowany jest bezpośrednio zbiór wektorów. |
| projection\_dimension | Numery wymiarów oddzielone przecinkami, na które mogą być rzutowane punkty w celu szacowania odległości (1 odpowiada pierwszemu wymiarowi, brak wymiaru zerowego 0). |
| projection\_source\_sequence | Kolejność kryteriów, według których realizowane będzie przycinanie (pruning) punktów w celu szacowania odległości.  Przykładowo, mając dane:  *reference\_point*=[0][max][min]  *projection\_dimensions*=1,2  *projection\_source\_sequence*=d1,r2,r3,r1,d2  Zbiór punktów zostanie posortowany względem wartości kryterium.  Kryterium dla każdego punktu wyznaczane jest na podstawie *projection\_source\_sequence*.  Sposób obliczania kryterium:  Kryterium stanowi wektor wartości: \*(d1) wartość pierwszego wymiaru punktu,  \*(r2) odległość punktu do drugiego punktu referencyjnego, \*(r3) odległość punktu do trzeciego punktu referencyjnego, \*(r1) odległość punktu do pierwszego punktu referencyjnego, \*(d2) wartość drugiego wymiaru punktu.  W czasie przycinania kolejnymi kryteriami branymi pod uwagę będą:   * wartość pierwszego wymiaru zdefiniowanego w liście *projection\_dimensions* (d1=1) * odległość do drugiego punktu referencyjnego zdefiniowanego w liście *reference\_point* (r2=[max]), * odległość do trzeciego punktu referencyjnego zdefiniowanego w liście *reference\_point* (r3=[min]), * odległość do pierwszego punktu referencyjnego zdefiniowanego w liście *reference\_point* (r1=[0]), * wartość drugiego wymiaru zdefiniowanego w liście *projection\_dimensions* (d2=2) |
| classification\_subset\_factor | Definiuje, co który rekord zbioru danych ma zostać wybrany do zbioru, które będzie podlegał klasyfikacji. |
| use\_placement | Flaga definiująca sposób wybrania najlepszego odpowiednika punktu klasyfikowanego w zbiorze punktów *dataset\_file\_path* przyjmująca wartości:  **\*true** – z zastosowaniem wyszukiwania binarnego,  **\*false** – z zastosowaniem wyszukiwania liniowego. |
| p\_sample\_index | Maksymalna liczba elementów zbioru randomSampleP wykorzystywanego do znalezienia vantage point. |
| s\_sample\_index | Maksymalna liczba elementów zbioru randomSampleD wykorzystywanego do znalezienia vantage point. |
| search\_method | Zmienna definiująca metodę poszukiwania sąsiadów w drzewie vantage point. Dostępne metody to: **\*range** – wyszkiwanie zakresowe (wymaga zdefiniowanie parametr *eps*), **\*k\_neighborhood** – wyszukiwanie k sąsiadów (wymaga zdefiniowania parametru (*k\_neighborhood*). |
| use\_boundaries | Flaga definiująca sposób wyszukiwania sąsiadów w drzewie VP przyjmująca wartości: **\*true** – wyszukiwanie z wykorzystaniem wartości sąsiednich mediany (największej wartości mniejszej od mediany i najmniejszej wartości większej od mediany),  **\*false** – wyszukiwanie z wykorzystaniem mediany. |

Tabela 3: Opis znaczenia parametrów pliku properties.txt

## Wyniki uruchomień programu

Wyniki uruchomień programu znajdują się w folderze **logs**. Każdemu poprawnemu uruchomieniu algorytmu odpowiada plik raportu o nazwie *run\_report%\_data\_i\_czas\_uruchomienia%.txt*. Raport zawiera:

* wartości parametrów programu,
* czasy wykonania poszczególnych etapów algorytmów,
* wynik wykonania algorytmu.

Tabela 4 opisuje znaczenie rekordów raportu dotyczących czasów wykonania w zależności od użytego algorytmu.

|  |  |  |
| --- | --- | --- |
| Algorytm | Nazwa czasu | Opis |
|  | Dataset read | Czas wczytywania zbioru punktów z pliku zbioru punktów. |
|  | Reference point calculation | Czas obliczania punktów referencyjnych. |
|  | Normalization | Czas normalizacji punktów. |
| DBSCAN | Algorithm execution | Czas wykonania algorytmu. |
| DBSCAN\_POINTS\_ELIMINATION | Algorithm execution | Czas wykonania algorytmu. |
| TI-DBSCAN | Algorithm execution | Czas wykonania algorytmu. Algorithm execution = Clustering + Distance + Sorting( + Building index); |
|  | Clustering | Czas wykonania grupowania. |
|  | Distance | Czas obliczania odległości punktów zbioru punktów do punktu referencyjnego. |
|  | Building index | Tworzenie indeksu dostępu do danych. |
|  | Sorting | Czas sortowania zbioru punktów według kryterium odległości do punktu referencyjnego. |
| TI-DBSCAN-REF | Algorithm execution | Czas wykonania algorytmu. Algorithm execution = Clustering + Distance + Sorting; |
|  | Clustering | Czas wykonania grupowania. |
|  | Distance | Czas obliczania odległości punktów zbioru punktów do każdego z punktów referencyjnych. |
|  | Sorting | Czas sortowania zbioru punktów według kryteriów odległości do punktów referencyjnych. |
| TI-DBSCAN-REF-PROJECTION | Algorithm execution | Czas wykonania algorytmu. Algorithm execution = Clustering + Distance + Sorting; |
|  | Clustering | Czas wykonania grupowania. |
|  | Distance | Czas obliczania odległości punktów zbioru punktów do każdego z punktów referencyjnych oraz wykonania projekcji. |
|  | Sorting | Czas sortowania zbioru punktów według kryteriów odległości do punktów referencyjnych oraz projekcji. |
| K-NEIGHBORHOOD | Algorithm execution | Czas wykonania algorytmu. |
| TI-K-NEIGHBORHOOD | Algorithm execution | Czas wykonania algorytmu. Algorithm execution = Clustering + Distance + Sorting + (Building index); |
|  | Clustering | Czas znajdowania k sąsiadów. |
|  | Distance | Czas obliczania odległości punktów zbioru punktów do punktu referencyjnego oraz, w przypadku klasyfikacji, odległości punktów zbioru punktów klasyfikowanych do punktu referencyjnego. |
|  | Building index | Tworzenie indeksu dostępu do danych. (Wartość zliczana jeśli algorytm wykonywany jest z opcją dostępu do danych przez indeks) |
|  | Sorting | Czas sortowania zbioru punktów według kryterium odległości do punktu referencyjnego. |
|  | Positioning | Czas pozycjonowania. |
| TI-K-NEIGHBORHOOD-REF | Algorithm execution | Czas wykonania algorytmu. Algorithm execution = Clustering + Distance + Sorting( + Building index); |
|  | Clustering | Czas znajdowania k sąsiadów. |
|  | Distance | Czas obliczania odległości punktów zbioru punktów do każdego z punktów referencyjnych oraz, w przypadku klasyfikacji, odległości punktów zbioru punktów klasyfikowanych do każdego z punktów referencyjnych. |
|  | Building index | Tworzenie indeksu dostępu do danych. (Wartość zliczana jeśli algorytm wykonywany jest z opcją dostępu do danych przez indeks) |
|  | Sorting | Czas sortowania zbioru punktów według kryteriów odległości do punktów referencyjnych. |
|  | Positioning | Czas pozycjonowania . |
| TI-K-NEIGHBORHOOD-REF-PROJECTION | Algorithm execution | Czas wykonania algorytmu. Algorithm execution = Clustering + Distance + Sorting( + Building index); |
|  | Clustering | Czas znajdowania k sąsiadów. |
|  | Distance | Czas obliczania odległości punktów zbioru punktów do każdego z punktów referencyjnych oraz wykonania projekcji a także, w przypadku klasyfikacji, odległości punktów zbioru punktów klasyfikowanych do każdego z punktów referencyjnych oraz wykonania projekcji. |
|  | Building index | Tworzenie indeksu dostępu do danych. (Wartość zliczana jeśli algorytm wykonywany jest z opcją dostępu do danych przez indeks) |
|  | Sorting | Czas sortowania zbioru punktów według kryteriów odległości do punktów referencyjnych oraz projekcji. |
|  | Positioning | Czas pozycjonowania (Pozycjonowanie wykonywane jest w przypadku klasyfikacji). |
| VP-TREE | Algorithm execution | Czas wykonania algorytmu. Algorithm execution = Clustering + Index building; |
|  | Clustering | Czas znajdowania k sąsiadów lub otoczenia eps. |
|  | Building index | Czas budowania drzewa vantage point z punktów zbioru punktów. |

Tabela 4: Opis rekordów raportu dotyczących czasów wykonania w zależności od algorytmu

Wartości czasów podawane są w sekundach.

Każdemu poprawnemu uruchomieniu aplikacji odpowiada plik raportu o nazwie: *ultimate\_run\_report%\_data\_i\_czas\_uruchomienia%.csv*. Raport zawiera:

* wartości parametrów programu,
* czasy wykonania poszczególnych etapów algorytmów,

dla wszystkich uruchomień algorytmów w danym uruchomieniu programu (tj. dla wszystkich plików parametrów w folderze **properties**).

Raport ten jest podsumowaniem uruchomienia aplikacji i zbiera najważniejsze dane z uruchomień algorytmów. Raport można uruchomić w arkuszu kalkulacyjny i dzięki dostępnym narzędziom swobodnie porównywać dane raportów uruchomień algorytmów.

## Zbiory danych

W folderze **datasets** znajdują się pliki ze zbiorami wektorów. Tabela 5 opisuje właściwości przykładowych przygotowanych zbiorów wektorów.

|  |  |  |  |
| --- | --- | --- | --- |
| Nazwa zbioru | Liczba wektorów zbioru | Liczba grup zbioru | Liczba wymiarów |
| dense\_d2\_r3000.txt | 3000 | 20 | 2 |
| dense\_d2\_r62556\_sequoia.txt | 62556 |  | 2 |
| dense\_d2\_r100000\_birch1.txt | 100000 |  | 2 |
| dense\_d15\_r10126.txt | 10126 |  | 15 |
| dense\_d56\_r100000\_cup98.txt | 100000 |  | 56 |
| dense\_d55\_r500000\_cup98.txt | 500000 |  | 55 |
| sparse\_d126373\_r8580\_karypis\_sport.txt | 8580 |  | 126373 |
| sparse\_d126373\_r4069\_karypis\_reviews.txt | 4069 |  | 126373 |
| sparse\_d126373\_r2301\_karypis\_hitech.txt | 2301 |  | 126373 |

Tabela 5: Opis przykładowych przygotowanych zbiorów punktów

Pliki nazywane są według konwencji *FormatPliku****\_d****LiczbaWymiarów****\_r****LiczbaRekordów\_NazwaZbiowu****.txt***.

## Uruchomienie

Aby program poprawnie zapisywał i odczytywał pliki, powinny one być umieszczone w następujący sposób:

Obok pliku **implementacja.exe**, na tym samym poziomie drzewa katalogów, powinny znajdować się następujące katalogi:

* **datasets** – zawierający pliki zbiorów danych,  
  Żadna linia pliku zbioru punktów nie powinna rozpoczynać się białym znakiem.  
  Kolejne wartości opisujące punkt powinny być oddzielone dokładnie jedną spacją.
* **logs** – zawierający pliki raportów wykonania (ścieżka względem pliku *implementacja.exe* \logs\),
* **properties** – zawierający pliki parametrów uruchomienia algorytmu(ścieżka względem pliku *implementacja.exe* \properties\).  
  W każdym pliku parametrów po nazwie parametru powinien znajdować się dokładnie znak ‘=’ a następnie wartość parametru.
* **algorithms\_engine\_properties** – zawierający plik parametrów uruchomienia aplikacji (ścieżka względem pliku *implementacja.exe* \algorithms\_engine\_properties\)  
  W pliku parametrów po nazwie parametru powinien znajdować się dokładnie znak ‘=’ a następnie wartość parametru.

Jeżeli pliki parametrów nie będą znajdowały się w odpowiednim katalogu (properties\) program będzie działał niepoprawnie.

Jeżeli katalog **logs** nie zostanie utworzony, pliki logów/raportów przebiegu wykonania programu nie zostaną zapisane.

Istnienie katalogu **datasets** nie jest wymagane, ponieważ w pliku parametrów podawana jest ścieżka do pliku ze zbiorem danych. Jednakże dla spójności rozwiązania, wszystkie pliki z danymi umieszczane są w folderze **datasets**.

Aby uruchomić implementację należy:

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

Po poprawnym uruchomieniu implementacji zostanie wydrukowany raport do folderu **logs**. W celu zapoznania się z wynikami uruchomienia należy przejrzeć raport wykonania.

# Wnioski

Przycinanie (pruning) zdecydowanie lepiej sprawdza się, jeśli pierwsze kryterium stanowi odległość do punktu referencyjnego.

//TODO

# Badania eksperymentalne

W przeprowadzonych przeze mnie eksperymentach badałem wydajność wyszukiwania k-sąsiedztwa oraz epsilonowego sąsiedztwa z zastosowaniem nierówności trójkąta oraz wpływ sposobu ich zaimplementowania. Implementacja algorytmów została wykonana w języku C++ w oparciu o pseudokody z artykułów [1], [4], [5].

Eksperymenty przeprowadziłem na środowisku Windows 7 x64 z procesorem Intel® i7™ 950 z dostępną pamięcią RAM równą 6GB. Aplikacja, którą posłużyłem się do zebrania wyników eksperymentów została skompilowana dla środowiska WIN32 z podniesioną flagą LARGEADDRESSAWARE, która umożliwia użycie do 3GB pamięci wirtualnej.

Eksperymenty uruchamiałem jako proces o priorytecie ‘Wysoki’ aby uniknąć wywłaszczenia procesu z rdzenia zmniejszając tym samym ryzyko losowego zakłócenia wyników.

Zauważyłem, że kolejne uruchomienia danego algorytmu z tymi samymi parametrami, nazwijmy je serią, wykonują się w różnym czasie. Ze względu na charakter wspomnianych różnic wyszczególniłem dwa przypadki. Pierwszy, w którym różnice czasów wykonań algorytmów w danej serii są niewielkie oraz drugi, w którym owe różnice są znaczące. W pierwszym przypadku powodem odchyleń są losowe zakłócenia. W drugim przypadku zauważyłem pewną prawidłowość, według której pierwsze uruchomienie algorytmu w serii trwa najdłużej z wszystkich, kolejne trochę krócej a różnice czasów trwania następnych uruchomień odpowiadają tym z przypadku pierwszego. Dzieje się tak ponieważ cache procesora (w moim przypadku 8MB, L3) z każdym wykonaniem algorytmu w serii działa sprawniej do chwili osiągnięcia pewnej sprawności granicznej dla danej serii. W efekcie czas wykonania algorytmu skraca się aż do momentu gdy różnice czasów kolejnych wykonań przypominają te z przypadku pierwszego. Aby umniejszyć wpływ wyżej opisanych zjawisk na zbieranie wyników czasowych poszczególnych kroków algorytmów ich pomiar dokonywany jest w następujący sposób. Spośród wszystkich wykonań w serii odrzucane są największe i najmniejsze wartości czasowe a z pozostałych obliczana jest wartość średnia.

# Dane testowe

W eksperymentach wykorzystałem zbiory danych stosowane w dziedzinowej literaturze, których odmienne charakterystyki pozwoliły na sprawdzenie działania algorytmów w różnych warunkach. Zamieszczone poniżej opisy dotyczą zbiorów, które wykorzystałem do przeprowadzenia eksperymentów, na których oparłem zaprezentowane w kolejnych podrozdziałach wnioski.

Covtype [7] jest zbiorem udostępnianym przez US Forest Service. Jego pełna nazwa to Forest Cover Type. Baza ta zawiera informacje na temat gatunków drzew w amerykańskich lasach. Zbiór posiada 581012 rekordów w formacie gęstym o 55 atrybutach. 10 pierwszych atrybutów to wartości zmiennych ilościowych, kolejne 44 atrybuty posiadają wartości 0 lub 1, z kolei ostatni determinuje jeden z siedmiu gatunków drzew. Należy zwrócić uwagę, że każdy z rekordów spośród atrybutów binarnych przyjmuje wartość 1 dokładnie dwa razy, a wartość zero 42 razy. W przypadku 10 pierwszych atrybutów zróżnicowanie wartości jest znacznie większe, z których pierwsze dwa posiadają najszerszą dziedziną (ponad 7000), a pozostałe charakteryzuje znacznie mniejsza różnorodność i zakres.

KDD-Cup98 [8] jest zbiorem opublikowanym podczas Drugiego Międzynarodowego Konkursu Odkrywania Wiedzy i Eksploracji Danych, który miał miejsce podczas KDD-98, Czwartej Międzynarodowej Konferencji Odkrywania Wiedzy i Eksploracji Danych w 1998 roku. Zbiór posiada 96367 rekordów w formacie gęstym o 56 atrybutach o wartościach będącymi liczbami naturalnymi. Najszerszą dziedziną [0, 6000] posiadają dwa atrybuty, najwęższą dziedzinę [0, 13] również tylko dwa atrybuty, natomiast aż 44 atrybuty posiadają taką samą dziedzinę [0, 99].

Karypis\_sport jest zbiorem danych wchodzącymi w skład bazy zbiorów [9] udostępnianych na stronie profesora George Karypis z Department of Computer Science & Engineering , University of Minesota. Zbiór karypis\_sport zawiera 8580 rekordów w formacie rzadkim o 126373 atrybutach przyjmujących wartości będące liczbami naturalnymi. Baza ta zawiera informacje o dokumentach dotyczących sportu.

Karypis\_review jest zbiorem danych wchodzącym w ten sam skład bazy zbiorów co karypis\_sport [9]. Zbiór ten zawiera 4069 rekordów w formacie rzadkim o 126373 atrybutach przyjmujących wartości będące liczbami naturalnymi. Baza ta zawiera informacje o dokumentach dotyczących recenzji.

# Dane testowe

# Bibliografia

1. Kryszkiewicz, M., Lasek, P.: TI-DBSCAN: Clustering with DBSCAN by Means of the Triangle Inequality, ICS Research Report, Warsaw University of Technology, April (2010)
2. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A Density-Based Algorithm of Discovering Clusters in Large Spatial Database with Noise. In: Proc. Of KDD’96, Portland (1996) 226-231
3. Kryszkiewicz, M.: Efficient Determination of Neighborhoods Defined in Terms of Cosine Similarity Measure, ICS Research Report, Warsaw University of Technology, April (2011)
4. Kryszkiewicz, M., Lasek, P.:A Neighborhood-Based Clustering by Means of the Triangle Inequality and Reference Points, ICS Research Report, Warsaw University of Technology
5. Yianilos, P. N.: Data Structures and Algorithms for Nearest Neighbor Search in General Metrics Spaces, The NEC Research Institute
6. Bozkaya, T., Ozsoyoglu M.: Distance-based indexing for high-dimensional metric spaces, Case Western Reserve University
7. Blackard J.A (1999, lipiec) The Forest CovType dataset. [Online ]. <http://ftp.ics.uci.edu/pub/machine-learning-databases/covtype/covtype.info>
8. Parsa I. (1999, luty) The UCI KDD Archive: KDD Cup 1988 Data. [Online]. <http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html>
9. Karypis G. (2002, sierpień) The various datasets used in evaluating the performance of CLUTO's clustering algorithms. [Online]. <http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download>