20 Luty 2010, Bielsko-Biała

Ulepszona klasteryzacja w eksploracji danych

Wybrane zagadnienia współczesnej algorytmiki
pod kierunkiem prof. dr hab. inż. Robert Schaefer
Informatyka III rok niestacjonarnie, semestr V

Łukasz Myśliński

1. Wstęp

Klastrowanie jest procesem odkrywczym w eksploracji danych. Grupuje zbiór danych w taki sposób, aby zmaksymalizować podobieństwo w obrębie klajstru i zminimalizowaniu podobieństwa pomiędzy dwoma różnymi klastrami. Z kolei odkryte klastry pomogą wyjaśnić charakter danego zbioru danych oraz służą jako baza do innych technik analizy. Klasteryzacja jest pożyteczna w celu scharakteryzowania grupy klientów bazując na zakupionych produktach, porządkowania zbioru dokumentów, grupowania genów o takich samych funkcjach itd.

Większość dostępnych algorytmów znajduje klastry, które są dopasowane do statycznych zbiorów danych. Z kolei efektywność tych algorytymów w pewnych przypadkach może zawieść. Na przykład z powodu źle wybranych statycznych parametrów lub algorytm nie będzie w stanie poprawnie sklastrować zbiór danych. Większość z nich ma problemy z danymi, które zawierają różne wymiary, rozmiary oraz kształty.

W moim projekcie skupiłem się na przykładowym zbiorze danych wektorów dwuwymiarowych, które zostały udostępnione przez pana Vipin Kumar na stronie http://www.cs.umn.edu/~kumar.

```
1   68.601997 102.491997
2   454.665985 264.808990
3   101.283997 169.285995
4   372.614990 263.140991
5   300.989014 46.555000
6   100.904999 205.776993
7   110.170998 55.647999
8   118.856003 47.445999
9   355.247009 76.431000
10  276.085999 182.048004
```

Rysunek 1 Przykładowy zbiór danych

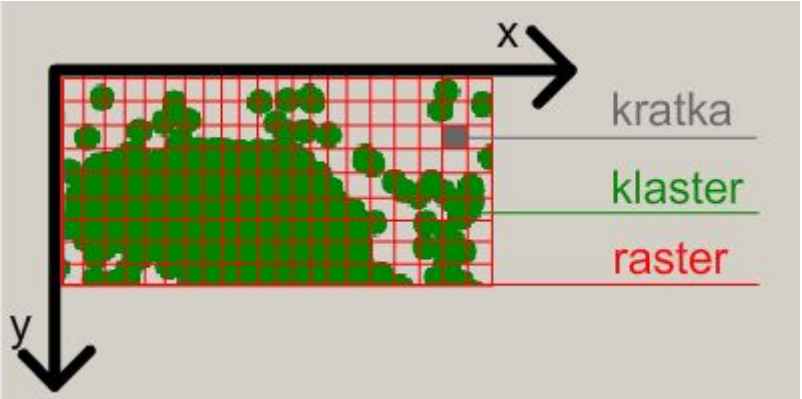
Zbiór tychże danych został następnie użyty w algorytmie hierarchicznym Chameleon pana Kumar'a. Do tego celu posłużyłem się gotowym programem autorstwa pana Weinan'a Wang'a, który po niewielkiej modyfikacji dawał na wyjściu zbiór danych uporządkowany według klastrów.

```
1   [ Klaster numer 0 ]
2   158.722000 211.529999
3   149.841003 204.709000
4   136.695999 213.328003
5   127.745003 218.070007
6   154.612000 215.951996
7   153.337997 174.563995
8   134.992996 212.268005
9   153.492004 180.065002
10  131.817993 196.412994
```

Rysunek 2 Przykładowy zbiór danych w klastrze

Zadaniem projektu była wizualizacja danych oraz ich klastrów, a następnie redukcja szumów. Do tego celu postanowiłem użyć języka c# i napisać stosowną aplikację. Aplikacja odczytuje przykładowy zbiór danych, a następnie przeprowadza ich wizualizację. Następnie odczytujemy zbiór danych, który został poddany algorytmowi Chameleon oraz

przeprowadzamy jego wizualizację. Na tym kroku możemy włączyć siatkę, która będzie ukazywać nam w jaki sposób klastry zostały podzielone. Kolejnym krokiem, który został wdrożony w aplikację było zastosowanie algorytmu usuwającego szum. Priorytetem była implementacja algorytmu używającego techniki rasteryzacji.



Rysunek 3 Przykład rasteryzacji w układzie współrzędnym na wektorach

Dodatkowymi pomysłami usunięcia szumów było użycie promienia, w celu skanowania ilości elementów w obrębie koła. Kolejnym pomysłem, który zrodził się w czasie pracy było użycie metryki Euklidesa i wyznaczanie siatki połączeń między danymi elementami w klastrach. W ten sposób można byłoby usunąć elementy najdalej oddalone od swoich sąsiadów.

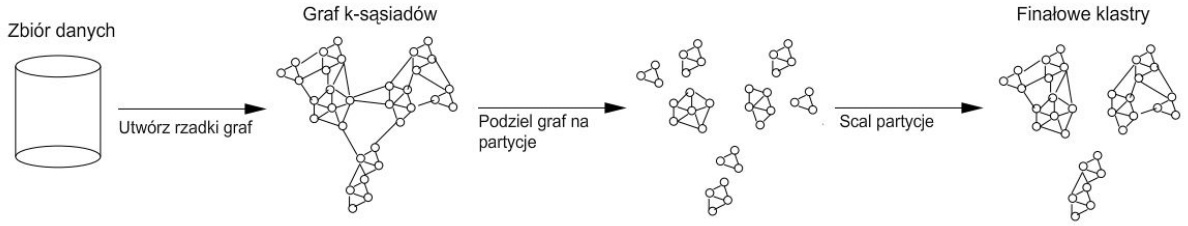
W projekcie skupiłem się na źródłowych danych, nie zaś na ich reprezentacji graficznej w układzie współrzędnych. W ten sposób starałem się zachować najbardziej możliwą precyzję kosztem pamięci. Starałem się ograniczać ilość pętli rozbudowywując klasy reprezentujące obiekty dodając do nich atrybuty. Poprzez relacje między klasami zaoszczędziłem na pamięci oraz kodzie. Aplikacja stała się bardziej czytelna i przyjemna w implementacji.

2. Programy

2.1 Chameleon

2.1.1. Algorytm

Chameleon jest nowym skupiająco hierarchicznie klastry algorytmem, który przezwycięża ograniczenia dotychczasowych klasteryzujących algorytmów.



Rysunek 4 Podsumowanie framework'a CHAMELEON

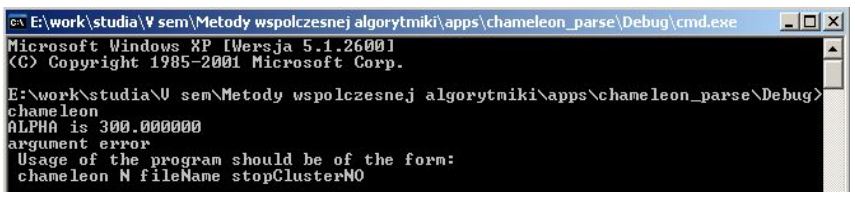
Kluczem tego algorytmu jest to, że tworzy relacje dla wewnętrznych połączeń oraz podobieństwa, które pozwalają identyfikować najbardziej zbliżone pary w klastrze. Omija ograniczenia omawiane we wstępie. Co więcej, Chameleon nie bazuje na statycznych parametrach i potrafi dostosowywać końcowy model bazując na charakterystyce scalanych klastrów.

Algorytm operuje na grafie z małą ilością subgrafów, za to bardzo rozproszoną. W tym grafie każdy liść reprezentuje element ze zbioru danych, zaś krawędzie reprezentują podobieństwo pomiędzy elementami. Znajduje klastry w zbiorze danych poprzez użycie dwu-fazowego algorytmu.

W czasie pierwszej fazy, Chameleon używa algorytmu do partycjonowania grafu w celu klasteryzacji elementów ze zbioru danych w nieliczne relatywnie małe subgrafy. Bazuje tutaj głównie na ilości elementów w danym klastrze. Znajduje początkowe subklastry używając hMetis (bardzo szybki i wydajny algorytm do partycjonowania grafów). hMetis partycjuje graf k-sąsiadów zbioru danych, a następnie zamienia go w rzadkie partycje, które minimalizują ucinanie krawędzi. Każda krawędź w grafie k-sąsiadów reprezentuje podobieństwo pomiędzy zbiorem danych, zaś partycjonowanie minimalizuje efektywnie relacje wśród punktów leżących po drugiej stronie partycji.

Z kolei w czasie drugiej fazy używa hierarchicznego algorytmu do znalezienia unikalnych klastrów poprzez powtarzanie procesu łączenia subklastrów.

2.1.2. Interface



Rysunek 5 Sposób użycia programu Chameleon

Program należy wywołać w następujący sposób: ***chameleon ilość_wektorów nazwa_pliku ilość_klastrów***. Źródło programu znajduje się w dodatku

2.2 Chameleon Filtering – usuwanie szumu

2.2.1. Algorytmy

2.2.1.1. Główny

Diagram 1 Formy

Diagram 2 Operacje na plikach

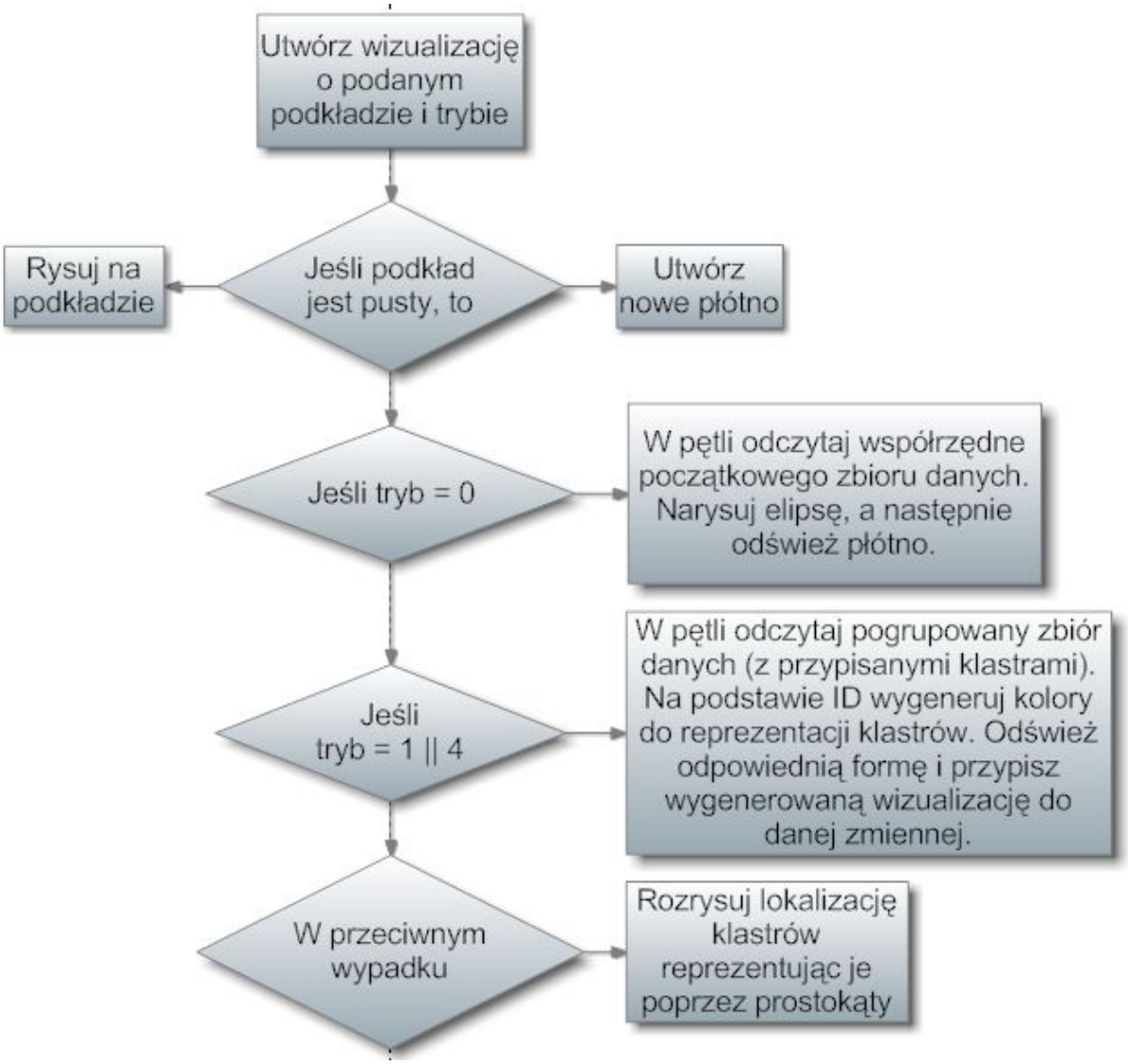


Diagram 3 Wizualizacja

2.2.1.2. Metody usuwania szumu

1. Raster

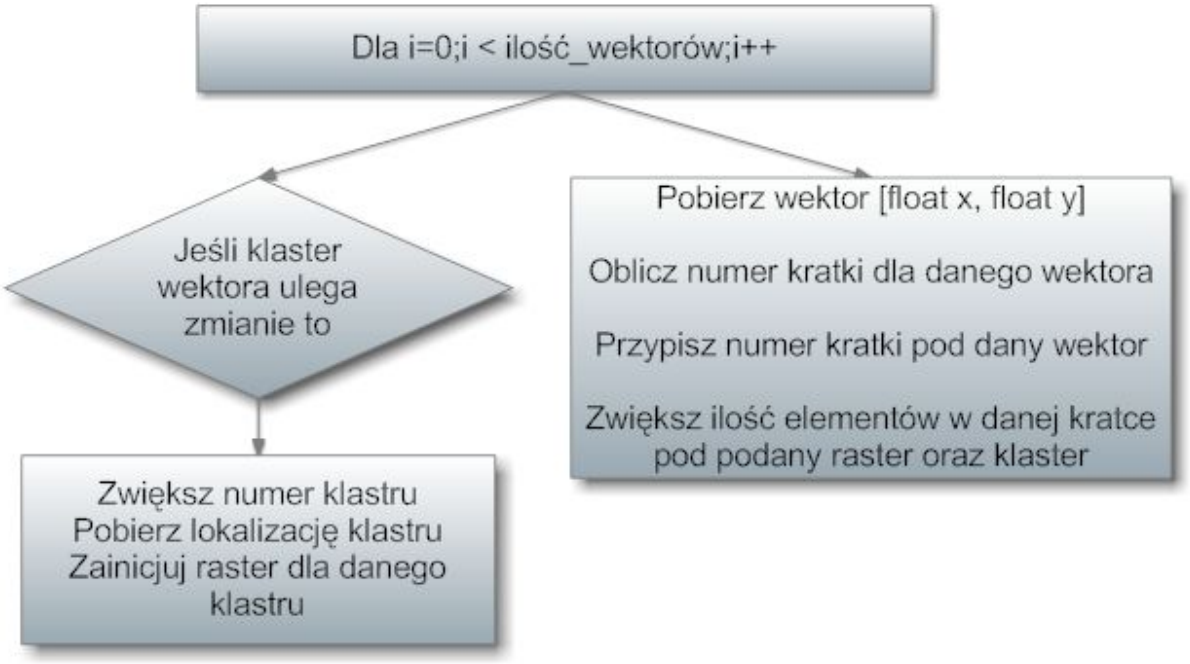


Diagram 4 Raster inicjalizacja

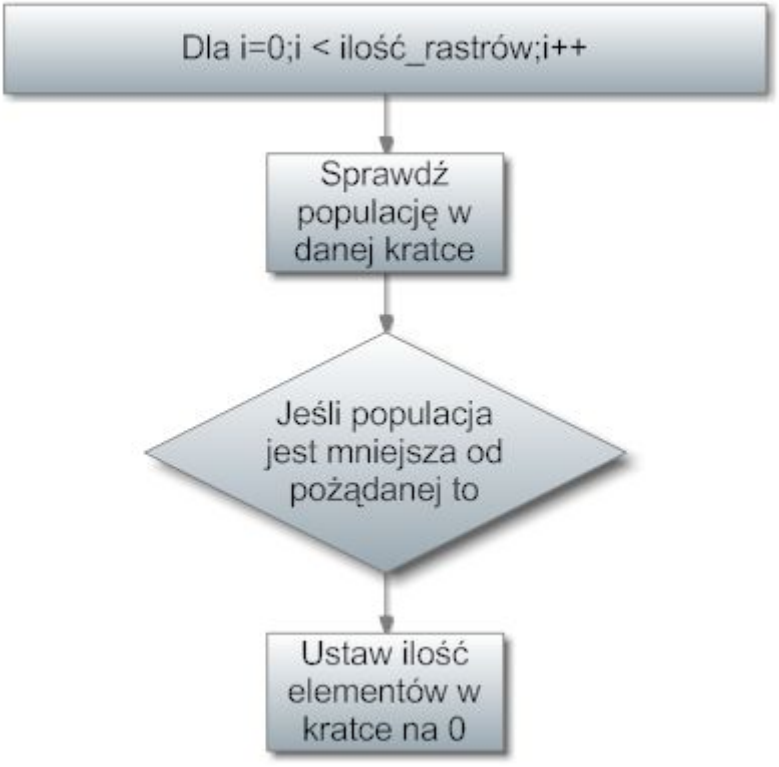


Diagram 5 Raster sprawdzanie kratek

Diagram 6 Usuwanie szumu

2. Kołowa

Algorytm kołowy do redukcji szumów polegałby na generowaniu kół o podanym promieniu w danym klastrze. Następnie sprawdzanie ile obiektów znajduje się w kole ze wzoru: $(x_0 - x)^2 + (y_0 - y)^2 - (r)^2 < 0$. Jeśli jest mała to usuwamy obiekty.

3. Dystansowa

Algorytm polegałby na obliczaniu odległości danego wektora od środka klastru według wzoru na tzw. odległość Euklidesową: $\sqrt{(x_0 - x)}^2 + \sqrt{(y_0 - y)^2}$. Następnie sprawdzanie, który obiekt nie spełnia pożądanej odległości. Tutaj należałoby się zastanowić nad obliczeniem średniej odległości i podawać tylko margines (przyrost odległości).

2.2.2. Klasy obiektów

Aplikacja posiada 3 klasy, które na zasadzie relacji komunikują się ze sobą.

Pierwsza z nich to **Pobj**, która reprezentuje wektor. Posiada współrzędne typu float, numer klastru oraz kratki w której się znajduje. Metody standardowo ustawiają współrzędne i zwracają ich wartości.

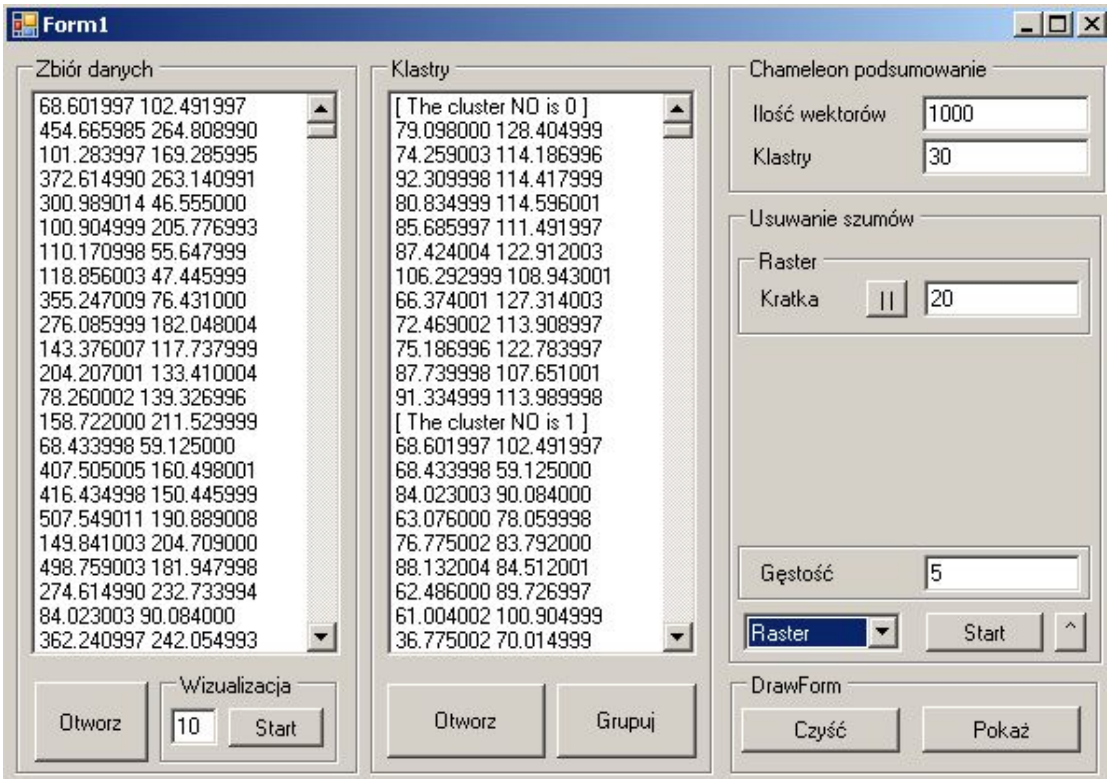
Druga z nich to **Cluster**, która posiada lokalizację (minX, minY, maxX, maxY), a także swoje ID.

Ostatnia z nich to **Raster**, która posiada rozmiar, szerokość oraz wysokość siatki, a także siatkę z ustawionymi ilościami elementów. Klasa ta ma takie metody jak

- inicjalizacja siatki dla danego rastera,
- zwracanie numeru kratki,
- zwiększanie liczby populacji w danej kratce,
- sprawdzanie siatki pod kątem ilości populacji, w wyniku czego eliminowane są niepożądane kratki
- sprawdzanie czy kratka o podanym ID zawiera pożądaną ilość elementów

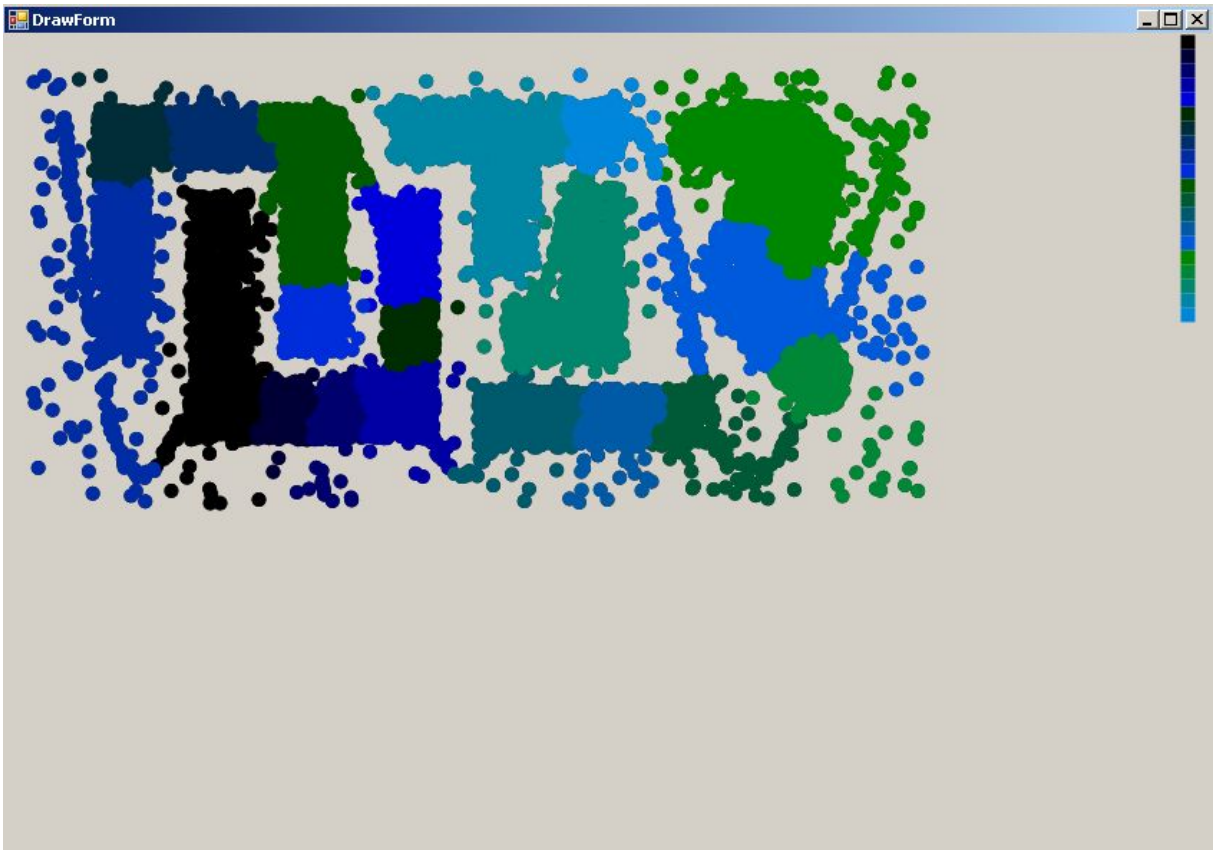
2.2.3. Interface

Na głównej formie widzimy 2 listbox'y, które służą do ładowania danych. Na prawo znajduje się oryginalny zbiór danych, zaś na lewo porządkowy zbiór z numerami klastrów. Po prawej stronie znajduje się podsumowanie ładowania plików z algorytmu Chameleon. Następnie poniżej znajduje się grupa przycisków do usuwania szumów. Klikając na przycisk koło „Kratka" włączymy rozrysowanie lokalizacji klastrów za pomocą prostokątów. Obok można ustawić jakich rozmiarów ma być kratka, na dole zaś gęstość kratki. Po czym należy uruchomić sprawdzanie wyborem metody oraz przyciskiem „Start". Przycisk „^" służy do przywołania okna, na którym aplikacja będzie usuwać szum.



Rysunek 6 Główna forma

Rysunek 7 Przykładowa wizualizacja zbioru danych



Rysunek 8 Przykładowa wizualizacja klastrów

Rysunek 9 Przykładowa wizualizacja klastrów po usunięciu szumu metodą rasteryzacji

3. Wnioski

Eksploracja danych jest dziedziną interdyscyplinarną. Jej dalszy rozwój wymaga połączenia technik i metod wypracowanych w różnych dziedzinach nauki: statystyce, grafice komputerowej, technologii baz danych, systemów równoległych, teorii optymalizacji, programowania matematycznego, uczenia maszynowego, itd. Część osób porównuje aktualny stan rozwoju tej dziedziny do stanu, w jakim systemy baz danych znajdowały się na początku swojej drogi. Brakuje nam standardu języka, w którym użytkownicy mogliby definiować swoje „zapytania", mechanizmów optymalizacji wykonywania takich zapytań, zarządzania efektywnym, współbieżnym wykonywaniem zapytań, narzędzi do budowy aplikacji, itd. Olbrzymie bazy danych przypominają dzisiaj czasami wielkie grobowce pełne danych, ale bez życia. Eksploracja danych jest próbą, być może kolejną, wniesienia do tych grobowców trochę światła. Dzięki takim algorytmom jak Chameleon, a następnie redukcji szumów można bardzo efektywnie i szybko wyodrębnić charakterystyczne grupy ze zbioru danych.

4. Źródła

- „Chameleon: Hierarchical Clustering Rusing Dynamic Modeling" - George Karypis, Eui-Hong (sam) Han, Viking Kumar
- "Study and Implementation of CHAMELEON algorithm for Gene Clustering" – Saurav Sahay
- "Introduction to Data Mining - Data Mining Cluster Analysis: Basic Concepts and Algorithms" – Tan, Steinbech, Kumar

5. Dodatki

5.1. Implementacja algorytmu CHAMELEON w języku

```
/*****************************************************
 * This is my CHAMELEON clustering program           *
 * Author: Weinan Wang                               *
 *****************************************************/

#include <stdio.h> /* for sscanf, FILE* */
#include <stdlib.h> /* for calloc, malloc, recalloc, free */
#include <math.h> /* for sqrt */
#include <string.h> /* for strlen */
#include <time.h>
#include <WINSOCK2.H>
//#include <sys/time.h>

#if defined(_MSC_VER) || defined(_MSC_EXTENSIONS)
  #define DELTA_EPOCH_IN_MICROSECS  11644473600000000Ui64
#else
  #define DELTA_EPOCH_IN_MICROSECS  11644473600000000ULL
#endif


/* !!!!!! */
/* Define the maximum size of the data points */
#define MAXSIZE 10000

/* maxmum length of the file name */
#define MAX_fileName 20

/* !!!!!! */
/* Define the K_nearest as 10,
 * this is according to the CHAMELEON paper p.13.
 */
/* #define K_nearest 10 */
#define K_nearest 15
/*#define K5_nearest 50 *//* 5*K_nearest */
#define K5_nearest 50

/* !!!!!! */
/* this is the balance constraint to be used in hMETIS.
 * this selection is according to page 10 of the CHAMELEON paper
 */
#define BC 25

/*!!!!!!*/
/* This is the MINSIZE over all the points, to be used in hMETIS.
 * this selection is according to page 13 of the CHAMELEON paper
 */
/* #define MINSIZE 0.025 */
#define MINSIZE 0.04
```

```
/* !!!!!! */
/* This is the domain width and height of the data set */
/* This set of WIDTH and HEIGHT is for t7.10k.dat */
#define WIDTH 700
#define HEIGHT 500
/* this is the diagonal distance */
#define DIAG 860.0

/* this is used for the maxmum number of groups
 * after phase 1's partition.
 */
#define MAXGROUPS 100

/* this alpha is used for the criterian function (p11 of the CHAMELEON paper)
 * for merging in phase 2.
 * This selection is according to p13 of the CHAMELEON paper.
 */
#define ALPHA 300.0

/*
 * This data type represent a hyper edge for a point.
 */
typedef struct
{
  int pointNO; /* the other point number */
  double similarity; /* similarity (DIAG-distance)/DIAG */
} Hyperedge;


/*
 * This data type contains information of
 * one point.
 */
typedef struct
{
  float x;
  float y;
  Hyperedge edges[K5_nearest];
  /* length cannot be bigger than K5_nearest */
  int length; /* current number of hyperedges */
} Point;


/*
 * This structure represent
 * a node in the binary tree of
 * graph partition.
 */
struct node {
  int * points; /* list of point numbers */
  int numberPoints; /* number of points belongs to this node */
```

```
  struct node *left; /* left subtree */
  struct node *right; /* right subtree */
};

// gettime

struct timezone
{
  int  tz_minuteswest; /* minutes W of Greenwich */
  int  tz_dsttime;     /* type of dst correction */
};

/************************************************
 * Declare static functions                     *
 ************************************************/
static int parsingInput(int argc, char *argv[]);
static int readData();
static int initialize();
static int establish_hyperGraph();
static double computeSimilarity(int i, int j);
static int cutNode(struct node * source, struct node * left, struct node * right, int ub);
static int partition(struct node* source, struct node * left, struct node * right);
static int belongs(struct node *, int point);
static int phase2();
static int merge(int group0, int group1);
static float computeGoodness(int group0, int group1);
static int computeAIC_AC(struct node * node0, struct node * node1, float * aic, float * ac);
static int clusterResult();
static long compute_time(struct timeval start_time, struct timeval end_time);
int gettimeofday(struct timeval *tv, struct timezone *tz);

/************************************************
 * Declare global variables                     *
 ************************************************/
/* data points */
Point points[MAXSIZE];

FILE * fp; /* file pointer pointing to the data file */
FILE * out_fp; /* file pointer pointing to the output file */

/* N is the total number of data points in the data file */
int N;
/* fileName is the data file's name */
char fileName[MAX_fileName + 1];
/* stopClusterNO is the number of remainning clusters after
 * the whole cluster process.
 */
int stopClusterNO;

/* partition tree root */
struct node *root = NULL;
```

```
/* threshold to decide whether stop partition */
int threshold;

/* this variable is for output matlab file */
int index_matlab=0;

/* this variable is used for indexing point groups
 * produced in phase 1.
 */
int groupIndex;
int * groups[MAXGROUPS];
int groupLength[MAXGROUPS];

/* following varaibles are for merging phase in phase 2 */
float bestGoodness; /* best goodness */
int mergeGroups[2] = {0, 0}; /* the two groups' number should be merged */

/* for timing */
long time_period;
struct timeval time_start, time_end;


/*************************************************************
 * Name    : main                                            *
 * Function: main function                                   *
 * Input   : argc, argv                                      *
 * Output  : void                                            *
 *          The usage of the rock program should be of the    *
 *          form:                                            *
 *            chameleon N fileName stopClusterNO             *
 *          Each parameter's meanning is :                   *
 *          chameleon -- program's name                      *
 *          int N    -- how many data points does the file has *
 *          string fileName -- indicate the data file        *
 *                          name which                       *
 *                           contains all the data vectors. *
 *          int stopClusterNO -- number of remaining clusters  *
 *                           at the end of clustering.      *
 *************************************************************/
int main(int argc, char * argv[]) {
 struct node * left;
 struct node * right;

 int i;

 if(gettimeofday(&time_start, NULL)<0)
   {
    printf("time error \n");
    return -1;
   }

 printf("ALPHA is %f \n", ALPHA);
```

```
 /* parse the command line */
if (parsingInput(argc, argv)<0)
  {
   return -1;
  }

 /* read in data file */
if ( readData()<0 )
  {
    fclose(fp);
    return -1;
  }

/* close read in data file */
fclose(fp);
/* for debugging purpose */
/* printData(); */

/* initialize data structure */
if ( initialize()<0 )
  {
   return -1;
  }

/* establish hyperGraph */
if (establish_hyperGraph()<0)
  {
   return -1;
  }

/* Now begin partition */
left = (struct node *)calloc(1, sizeof(struct node));
if(left == NULL)
  {
   printf("cannot allocate memory for left! \n");
   return -1;
  }
right = (struct node *)calloc(1, sizeof(struct node));
if(right == NULL)
  {
   printf("cannot allocate memory for right! \n");
   return -1;
  }

/* begin partitionning, this is a recursive program */
if(partition(root, left, right)<0)
  {
   printf("partition error! \n");
   return -1;
  }
```

```
if(phase2()<0)
  {
   printf("phase2 error! \n");
   return -1;
  }

if(clusterResult()<0)
  {
   fclose(out_fp);
   printf("clusterResult error! \n");
   return -1;
  }
fclose(out_fp);

for(i=0; i<groupIndex; i++)
  {
   free(groups[i]);
   groups[i] = NULL;
  }

if(gettimeofday(&time_end, NULL)<0){
  printf("time error \n");
  return -1;
}

 /* Now compute the time for sequential sorting */
time_period = compute_time(time_start, time_end);

printf("time spend is : %d ms \n", time_period);

return 1;

}/* end main */
// time

int gettimeofday(struct timeval *tv, struct timezone *tz)
{
 FILETIME ft;
 unsigned __int64 tmpres = 0;
 static int tzflag;

 if (NULL != tv)
 {
  GetSystemTimeAsFileTime(&ft);

  tmpres |= ft.dwHighDateTime;
  tmpres <<= 32;
  tmpres |= ft.dwLowDateTime;

  /*converting file time to unix epoch*/
  tmpres /= 10;  /*convert into microseconds*/
  tmpres -= DELTA_EPOCH_IN_MICROSECS;
```

```
    tv->tv_sec = (long)(tmpres / 1000000UL);
    tv->tv_usec = (long)(tmpres % 1000000UL);
  }

  if (NULL != tz)
  {
    if (!tzflag)
    {
      _tzset();
      tzflag++;
    }
    tz->tz_minuteswest = _timezone / 60;
    tz->tz_dsttime = _daylight;
  }

  return 0;
}


/*************************************************************
 * Name    : compute_time                                    *
 * Function: compute the time interval between two           *
 *           given time structures.                          *
 * Input   : (struct timeval start_time),                    *
 *           (struct timeval end_time)                       *
 * Output  : long                                            *
 *************************************************************/

static long compute_time(struct timeval start_time, struct timeval end_time)
{
  long sec, msec, time;

  /* time interval in sec */
  sec = (long)(end_time.tv_sec - start_time.tv_sec);

  /* time interval in msec */
  msec = (long)(end_time.tv_usec - start_time.tv_usec);

  /* compute time difference in msec */
  time = sec*1000000+msec;

  return time;

}/* end compute_time */

/*************************************************************
 * Name    : clusterResult                                   *
 * Function: output cluster result.                          *
 * Input   : void                                            *
 * Output  : int                                             *
 *************************************************************/
static int clusterResult(){
```

```
  int i, j, count;

  /* open hyperGraph file to write output */

  if ((out_fp = fopen("clusters.dat", "w")) == NULL )
    {
     printf("cannot open output file correctly! \n");
     return -1;
    }

  count = 0;
  for(i=0; i<groupIndex; i++){
    if(groupLength[i]>0){
      /* printf("c%d =[\n", count);*/
      fprintf(out_fp, "[ Klaster numer %d ]\n", i);
      //fprintf(out_fp, "c%d =[", count);
      for(j=0; j<groupLength[i]; j++){
           /* printf("%f %f \n", points[groups[i][j]].x, points[groups[i][j]].y);*/
           fprintf(out_fp, "%f %f \n", points[groups[i][j]].x, points[groups[i][j]].y);
      }/* end for */
      /* printf("];\n\n");*/
      //fprintf(out_fp, "];\n");
      count++;
    }/* end if */
  }/* end for */

  if(count != stopClusterNO){
    printf("existing cluster number is not equal to required cluster numer! \n");
    return -1;
  }/* end if */

  return 1;
}/* end clusterResult */

/*************************************************************
 * Name    : computeAIC_AC                                   *
 * Function: compute absolute inter-connectivity and absolute   *
 *          closeness of two nodes.                          *
 * Input   : struct node * node0 -- pointer to node0.          *
 *          struct node * node1 -- pointer to node1.          *
 *          float * aic -- pointer to the return value for    *
 *                      absolute inter-connectivity.        *
 *          float * ac  -- pointer to the return value for    *
 *                      absolute closeness.                 *
 * Output  :  float                                          *
 *************************************************************/
static int computeAIC_AC(struct node * node0, struct node * node1, float * aic, float * ac){
  int count, i, j;

  count = 0;
  (* aic) = 0.0;
  (* ac) = 0.0;
```

```
  /* count the edges from node0 */
  for(i=0; i<node0->numberPoints; i++){
    for(j=0; j<points[node0->points[i]].length; j++){
      if(belongs(node1, points[node0->points[i]].edges[j].pointNO)>0){
          (* aic) = (* aic) + points[node0->points[i]].edges[j].similarity;
          count++;
      }/* end if */
    }/* end for j */
  }/* end for i */
  /*  printf("!!! aic is %f,", *aic);*/
  /* count the edges from node1 */
  for(i=0; i<node1->numberPoints; i++){
    for(j=0; j<points[node1->points[i]].length; j++){
      if(belongs(node0, points[node1->points[i]].edges[j].pointNO)>0){
          (* aic) = (* aic) + points[node1->points[i]].edges[j].similarity;
          count++;
      }/* end if */
    }/* end for j */
  }/* end for i */
  /* printf(" aic is %f !!!", *aic);*/

  /*  printf("count is %d, ", count);*/

  if(count>0)
    {
     (* ac) = (* aic)/((float)count);
    }
  else
    {
     (*ac) = 0;
     (* aic) = 0;
    }

  if(*ac > 10000 || *aic > 10000){
    printf("ac is %e, aic is %e, count is %d \n", (*ac), (*aic), count);
  }
  return 1;
}/* end computeAIC_AC */
/*************************************************************
 * Name    : computeGoodness                                 *
 * Function: compute goodness for merging two clusters.      *
 * Input   : int group0 -- group0's index.                   *
 *           int group1 -- group1's index.                   *
 * Output  :  float                                          *
 *************************************************************/
static float computeGoodness(int group0, int group1){
  float goodness;
  float ri, rc;
  int ubFactor;

  /* form two nodes according to two groups */
  struct node * node0;
```

```
struct node * node1;
/* left and right is for cutting the node0 and node 1 to
 * compute internal connectivity and internal closeness.
 */
struct node * left0;
struct node * right0;
struct node * left1;
struct node * right1;

int i;
/* absolute inter-connectivity and absolute closeness */
float aic, ac, aic0, ac0, aic1, ac1;

/*!!!!!!!*/
ubFactor = 6;

/* First create node0 and node1 according to group0 and group1 */
node0 = (struct node *)calloc(1, sizeof(struct node));
node1 = (struct node *)calloc(1, sizeof(struct node));
if(node0==NULL || node1==NULL){
  printf("cannot allocate memory for node0 and node1. \n");
  return -1;
}/* end if */

node0->left=NULL;
node0->right=NULL;
node0->numberPoints = groupLength[group0];
node0->points = (int *)calloc(node0->numberPoints, sizeof(int));
if(node0->points == NULL){
  printf("cannot allocate memory for node0->points. \n");
  return -1;
}
for(i=0; i<groupLength[group0]; i++){
  node0->points[i]=groups[group0][i];
}/* end for */


node1->left=NULL;
node1->right=NULL;
node1->numberPoints = groupLength[group1];
node1->points = (int *)calloc(node1->numberPoints, sizeof(int));
if(node1->points == NULL){
  printf("cannot allocate memory for node0->points. \n");
  return -1;
}
for(i=0; i<groupLength[group1]; i++){
  node1->points[i]=groups[group1][i];
}/* end for */



/* Now compute absolute inter-connectivity and
```

```
 * absolute closeness.
 */
computeAIC_AC(node0, node1, &aic, &ac);


if(ac > 0.0 && aic > 0.0){
  /* now need to compute the internal connectivity of
   * node0 and node1.
   */
  left0 = (struct node *)calloc(1, sizeof(struct node));
  right0 = (struct node *)calloc(1, sizeof(struct node));
  if(left0==NULL || right0==NULL){
    printf("cannot allocate memory for node0 and node1. \n");
    return -1;
  }/* end if */

  if(cutNode(node0, left0, right0, ubFactor)<0)
    {
        printf("cut node0 error! \n");
        return -1;
    }

  /* Now compute internal inter-connectivity and
   * absolute closeness of node0.
   */
  computeAIC_AC(left0, right0, &aic0, &ac0);

  left1 = (struct node *)calloc(1, sizeof(struct node));
  right1 = (struct node *)calloc(1, sizeof(struct node));
  if(left1==NULL || right1==NULL){
    printf("cannot allocate memory for node0 and node1. \n");
    return -1;
  }/* end if */

  if(cutNode(node1, left1, right1, ubFactor)<0)
    {
        printf("cut node0 error! \n");
        return -1;
    }

  /* Now compute internal inter-connectivity and
   * absolute closeness of node0.
   */
  computeAIC_AC(left1, right1, &aic1, &ac1);


  /* Now compute Relative Inter-Connectivity */
  ri = (aic*2)/(aic0+aic1);

  /* Now compute Relative Closeness */
  rc = ac*(node0->numberPoints+node1->numberPoints)/((node0->numberPoints)*ac0 +
                                                     (node1->numberPoints)*ac1);
```

```
    if(ri>10000 || rc>10000){
      printf("ri or ic is wrong! ri= %f, rc= %f \n", ri, rc);
      return -1;
    }
    /* The final goodness according to p11 of the CHAMELEON paper */
    /* printf("ri is %f, rc is %f. \n", ri, rc);*/
    goodness = ri*pow((double)rc, ALPHA);
    free(node0->points);
    free(node1->points);
    free(left0->points);
    free(right0->points);
    free(left1->points);
    free(right1->points);
    free(left0);
    free(right0);
    free(left1);
    free(right1);
    free(node0);
    free(node1);
  }
  else{
    ri=0.0;
    rc=0.0;
    /* printf("ri is %f, rc is %f. \n", ri, rc);*/
    goodness = 0.0;

    free(node0->points);
    free(node1->points);
    free(node0);
    free(node1);
  }/* end if...else...*/

  return goodness;
}/* end computeGoodness */

/*****************************************************************
 * Name    : merge                                       *
 * Function: merge 2 clusters.                             *
 * Input   : int group0 -- group0's index.                    *
 *           int group1 -- group1's index.                    *
 * Output  :  int                                        *
 *****************************************************************/
static int merge(int group0, int group1){
  int i, j;

  groups[group0] = (int *)realloc(groups[group0],
                                  (groupLength[group0]+groupLength[group1])*sizeof(int)) ;
  if(groups[group0]==NULL){
    printf("cannot enlarge groups[group0]; \n");
    return -1;
  }
```

```
  /* move elements from group1 to group0 */
  for(i=groupLength[group0], j=0; i<(groupLength[group0]+groupLength[group1]); i++,
j++){
    groups[group0][i] = groups[group1][j];
  }/* end for */

  free(groups[group1]);
  groups[group1]=NULL;

  groupLength[group0] = groupLength[group0]+groupLength[group1];
  groupLength[group1] = 0;

  return 1;
}/* end merge */

/*****************************************************************
 * Name    : phase2                                      *
 * Function: phase 2, merging clusters.                        *
 * Input   : void                                      *
 * Output  :  int                                      *
 *****************************************************************/
static int phase2(){
  int clusterNO;
  float goodness;
  int group0, group1;
  int i, j;

  /*
   * groupIndex at the end of phase 1 gives the
   * number of initial clusters after phase 1.
   */
  clusterNO = groupIndex;

#ifdef Debug_phase2
  printf("At the beginning of phase 2, clusterNO is %d, stopClusterNO is %d \n",
          clusterNO, stopClusterNO);
#endif

  if(clusterNO<stopClusterNO){
    printf("at the beginning of phase 2, clusterNO is already smaller than stopClusterNO!
error\n");
    return -1;
  }/* end if */

  bestGoodness = 0.0;
  while(clusterNO>stopClusterNO){
    /* the for loop decide which two groups need to be merged */
    for(i=0; i<(groupIndex-1); i++){
      if(groupLength[i]!=0){
          group0 = i;
          for(j=(group0+1); j<groupIndex; j++){
            if(groupLength[j]!=0){
```

```
            group1 = j;

            /* Now compute the goodness for merging the two clusters */
            goodness = computeGoodness(group0, group1);
            if(goodness > 10000 || goodness < 0){
              printf("group0 is %d, group1 is %d,!!!\n",
                      group0, group1);
              printf("goodness is %f, some thing wrong!\n", goodness);

              return -1;
            }
            /* printf("group0 is %d, group1 is %d, goodness is %f !!! \n",
               group0, group1, goodness);
            */
            if(goodness > bestGoodness)
              {
                   bestGoodness = goodness;
                   mergeGroups[0] = group0;
                   mergeGroups[1] = group1;
              }
          }/* end if */
        }/* end for j*/
    }/* end if */
  }/* end for i*
  /* now merge the two selected groups */
  if(mergeGroups[0]==mergeGroups[1]){
    printf("mergeGroups[0]==mergeGroups[1]==%d, something wrong! \n",
             mergeGroups[0]);
    return -1;
  }

  if(bestGoodness == 0.0){
    printf("bestGoodness reached 0 \n");
    return 1;
  }

  printf("best goodness is %f \n", bestGoodness);

  if(merge(mergeGroups[0], mergeGroups[1])<0)
    {
       printf("merge error! \n");
       return -1;
    }

  clusterNO--;

#ifdef Debug_phase2
   printf("In the while loop,  clusterNO is %d, stopClusterNO is %d, mergeGroups[0] is %d,
mergeGroups[1] is %d. \n",
           clusterNO, stopClusterNO, mergeGroups[0], mergeGroups[1]);
#endif
```

```
   bestGoodness = 0.0;
   mergeGroups[0]=0;
   mergeGroups[1]=0;

  }/* end while */

  return 1;

}/* end phase2 */


/*************************************************************
 * Name    : belongs                                         *
 * Function: decide whether a point belongs to a node.       *
 * Input   : struct node * sourceNode --  the node           *
 *           int point -- the point                          *
 * Output  :  int                                            *
 *************************************************************/
static int belongs(struct node * sourceNode, int point){
  int i, tmp;

  tmp = 0;

  for(i=0; i<sourceNode->numberPoints; i++){
    if(sourceNode->points[i] == point)
      {
          tmp = 1;
          break;
      }/* end if */
  }

  return tmp;

}/* end belongs */


/*************************************************************
 * Name    : cutNode                                         *
 * Function: cut a node of the tree into two parts, and thus *
 *           return the left pointer and the right pointer.  *
 * Input   : struct node * source --  the source node to be cut.*
 *           struct node * left -- the left resulting node.  *
 *           struct node * right -- the right resulting node.*
 * Output  :  int                                            *
 *************************************************************/
/* May 30th, Weinan: one problem I have just found is that I need to
 * establish a kind of checking table to connect my global vertex
 * number with local vertex number to be used in the HMETIS_PartRecursive
 * function calling.
 */
static int cutNode(struct node * source, struct node * left, struct node * right, int ub){
  int nvtxs, nhedges;
```

```
int * vwgts = NULL;
int * eptr = NULL;
int * eind = NULL;
int * hewgts = NULL;
int nparts;
int ubfactor;
int * options = NULL;
int * part = NULL;
int * edgecut = NULL;

int tmp, i, j, k, l, tmpNO, flag, found, group0, group1, index0, index1;
/* this is the table used to correspond global vertice number with
 * local vertice number.
 */
int * checkTable;

/* number of vertices */
nvtxs = source->numberPoints;
checkTable = (int *)calloc(nvtxs, sizeof(int));
if(checkTable == NULL)
  {
    printf("cannot allocate memory for checkTable! \n");
    return -1;
  }
/* load the vertice's number into the checkTable */
for(i=0; i<nvtxs; i++){
  checkTable[i] = source->points[i];
}/* end for */

/* number of edges */
tmp=0;
for(i=0; i<nvtxs; i++){
  for(j=0; j<points[source->points[i]].length; j++){
    /* decide a point whether belongs to a node */
    if(belongs(source, points[source->points[i]].edges[j].pointNO)==1) /* !!!!!! */
        tmp++;
  }/* end for j*/
}/* end for i*/
nhedges = tmp;

/* weight of the vertices, because
 * the vertices are not weighted, so
 * according to p13 of the hMETIS manual,
 * vwgts can be NULL.
 */
vwgts = NULL;

/* eptr and eind */
/* because all my edges are of
 * size 2, so it makes things
 * easier.
 */
```

```
tmp = nhedges+1;
eptr = (int *)calloc(tmp, sizeof(int));
if(eptr == NULL)
  {
   printf("cannot allocate memory for eptr! \n");
   return -1;
  }
for(i=0; i<tmp; i++)
  eptr[i]=i*2;

/* when loading eind, need to check the checkTable */
eind = (int *)calloc(2*tmp, sizeof(int));
if(eind == 0)
  {
   printf("cannot allocate memory for eind! \n");
   return -1;
  }
k = 0;
for(i=0; i<nvtxs; i++){
  for(j=0; j<points[source->points[i]].length; j++){
    /* decide a point whether belongs to a node */
    if(belongs(source, points[source->points[i]].edges[j].pointNO)==1){
         /* eind[k] = source->points[i];*/
         eind[k] = i;
         k++;
         /* eind[k] = points[source->points[i]].edges[j].pointNO; */
         tmpNO =  points[source->points[i]].edges[j].pointNO;
         flag = 0;
         for(l=0; l<nvtxs; l++){
           if (tmpNO == checkTable[l]){
             flag = 1;
             found = l;
             break;
           }
         }
         if(flag == 1)
           eind[k] = found;
         else
           {
             printf("some thing wrong with checkTable! \n");
             return -1;
           }/* end if...else... */
         k++;
    }/* end if */
  }/* end for j*/
}/* end for i*/
if(k != 2*nhedges){
  printf("some thing wrong with eind! \n");
  return -1;
}/* end if */
```

```
/* hewgts: an array of size nhedges that stores the weight
 * of the hyperedges.
 */
hewgts = (int *)calloc(nhedges, sizeof(int));
if(hewgts == 0)
  {
   printf("cannot allocate memory for hewgts! \n");
   return -1;
  }
k = 0;
for(i=0; i<nvtxs; i++){
  for(j=0; j<points[source->points[i]].length; j++){
    /* decide a point whether belongs to a node */
    if(belongs(source, points[source->points[i]].edges[j].pointNO)==1){
        /*!!!!!! here I have to do a cast becasue now similarity is a double */
        hewgts[k] = (int)(points[source->points[i]].edges[j].similarity*DIAG);
        k++;
    }/* end if */
  }/* end for j*/
}/* end for i*/
if(k != nhedges){
  printf("some thing wrong with hewgts! \n");
  return -1;
}/* end if */

/* nparts: number of desired partitions */
nparts = 2;

/* ubfactor: relative imbalance factor */
ubfactor = ub;

/* options */
/* note that is options[0]=0,
 * then default values are used.
 */
options =  (int *)calloc(9, sizeof(int));
if(options == 0)
  {
   printf("cannot allocate memory for options! \n");
   return -1;
  }
options[0] = 0;

/* part */
part =  (int *)calloc(nvtxs, sizeof(int));
if(part == 0)
  {
   printf("cannot allocate memory for part! \n");
   return -1;
  }

/* edgecut */
```

```
edgecut = (int *)calloc(1, sizeof(int));
if(edgecut == 0)
  {
   printf("cannot allocate memory for edgecut! \n");
   return -1;
  }

HMETIS_PartRecursive(nvtxs, nhedges, vwgts, eptr, eind, hewgts,
                     nparts, ubfactor, options, part, edgecut);

group0 = 0;
group1 = 0;
for(i=0; i<nvtxs; i++){
  if(part[i] == 0)
    group0++;
  else if(part[i] == 1)
    group1++;
  else
    {
        printf("something wrong with part. \n");
        return -1;
    }/* end if..else...*/
}/* end for */

left->numberPoints = group0;
right->numberPoints = group1;
left->points = (int *)calloc(group0, sizeof(int));
if(left->points == NULL)
  {
   printf("cannot allocate memory for left->points \n");
   return -1;
  }
right->points = (int *)calloc(group1, sizeof(int));
if(right->points == NULL)
  {
   printf("cannot allocate memory for right->points \n");
   return -1;
  }
left->left = NULL;
left->right = NULL;
right->left = NULL;
right->right = NULL;

index0 = 0;
index1 = 0;
for(i=0; i<nvtxs; i++){
  if(part[i] == 0)
    {
        left->points[index0] = checkTable[i];
        index0++;
    }
  else if(part[i] == 1)
```

```
    {
        right->points[index1] = checkTable[i];
        index1++;
    }
   else
    {
        printf("something wrong with part. \n");
        return -1;
    }/* end if..else...*/
 }/* end for */
 if(index0!=group0 || index1!=group1)
   {
    printf("some thing wrong with index0-1. \n");
    return -1;
   }

 free(vwgts);
 free(eptr);
 free(eind);
 free(hewgts);
 free(options);
 free(part);
 free(edgecut);

 free(checkTable);

 return 1;

}/* end cutNode */


/*****************************************************************
 * Name    : partition                                  *
 * Function: partition the graph recursively.                *
 * Input   : void                                     *
 * Output  :  int                                     *
 *****************************************************************/
static int partition(struct node * source, struct node * left, struct node * right){
 struct node * l_left;
 struct node * l_right;
 struct node * r_left;
 struct node * r_right;
 int i;

 /* cut the source node into left node and right node */
 if(cutNode(source, left, right, BC)<0){
   printf("cutNode error! \n");
   return -1;
 }
 /* link the two sub-trees to the root */
 source->left = left;
 source->right = right;
```

```
/* if left sub-tree is still bigger than threshold */
if(left->numberPoints > threshold) {
  l_left = (struct node *)calloc(1, sizeof(struct node));
  if(l_left == NULL)
    {
        printf("cannot allocate memory for left! \n");
        return -1;
    }
  l_right = (struct node *)calloc(1, sizeof(struct node));
  if(l_right == NULL)
    {
        printf("cannot allocate memory for right! \n");
        return -1;
    }

  if(partition(left, l_left, l_right)<0)
    return -1;
}else{
  groupLength[groupIndex] = left->numberPoints;
  groups[groupIndex] =  (int *)calloc(left->numberPoints, sizeof(int));
  if(groups[groupIndex] == NULL)
    {
        printf("cannot allocate memory for groups! \n");
        return -1;
    }
  for(i=0;  i<left->numberPoints; i++){
    groups[groupIndex][i] = left->points[i];
  }/* end for */
  groupIndex++;
  printf("groupIndex = %d \n", groupIndex);
  if(groupIndex >= MAXGROUPS)
    {
        printf("groupIndex is now bigger than MAXGROUPS \n");
        return -1;
    }

if(right->numberPoints > threshold){
  r_left = (struct node *)calloc(1, sizeof(struct node));
  if(r_left == NULL)
    {
        printf("cannot allocate memory for left! \n");
        return -1;
    }
  r_right = (struct node *)calloc(1, sizeof(struct node));
  if(r_right == NULL)
    {
        printf("cannot allocate memory for right! \n");
        return -1;
    }

  if(partition(right, r_left, r_right)<0)
```

```
    return -1;
  }else{
    groupLength[groupIndex] = right->numberPoints;
    groups[groupIndex] =  (int *)calloc(right->numberPoints, sizeof(int));
    if(groups[groupIndex] == NULL)
      {
          printf("cannot allocate memory for groups! \n");
          return -1;
      }
    for(i=0;  i<right->numberPoints; i++){
      groups[groupIndex][i] = right->points[i];
    }/* end for */
    groupIndex++;
    printf("groupIndex = %d \n", groupIndex);
    if(groupIndex >= MAXGROUPS)
      {
          printf("groupIndex is now bigger than MAXGROUPS \n");
          return -1;
      }

  return 1;
}/* end partition */

/*****************************************************************
 * Name    : computeSimilarity                                   *
 * Function: compute similarity between point i and point j      *
 * Input   :  int i -- index of point i                          *
 *          int j -- index of point j                            *
 * Output  :  double -- similarity in (DIAG-distance)/DIAG.      *
 *****************************************************************/
static double computeSimilarity(int i, int j) {
  float i_x, i_y, j_x, j_y;
  double simi;

  i_x = points[i].x;
  i_y = points[i].y;
  j_x = points[j].x;
  j_y = points[j].y;

  simi = (DIAG - sqrt((double)(i_x-j_x)*(i_x-j_x)+(double)(i_y-j_y)*(i_y-j_y)) )/DIAG;

  /*printf("simi is %f !!!", simi);*/

  return simi;
}/* end computeSimilarity */


/*****************************************************************
 * Name    : establish_hyperGraph                                *
 * Function: establish hyperGraph based on the points            *
 *          information.                                         *
 * Input   :  void                                               *
```

```
 * Output  : int                                          *
 ***********************************************************/
static int establish_hyperGraph() {
  int i, j, k, l;
  double *similarity = NULL;
  double bestSimi;
  int indexBestSimi;
  int flag = 0;

  similarity = (double *)calloc(sizeof(double), N);
  if(similarity == NULL)
    return -1;

  /* find the k_nearest points for each of the N point */
  for (i=0; i<N; i++){
    /* compute similarity of point i with each point j */
    for (j =0; j<N; j++){
      similarity[j] = computeSimilarity(i,j);
    }/* end for j */

    /* find the K_nearest points around point i */
    for (k=0; k<K_nearest; k++){

      bestSimi = 0.0;
      indexBestSimi = 0;

      for (j=0; j<N; j++){
          if(j != i){
            if (similarity[j]>bestSimi){
              indexBestSimi = j;
              bestSimi = similarity[j];
            }/* end if */
          }/* end if */
      }/* end for j */

      /* add a new edge to one of the two points */
      if(i<indexBestSimi){
          points[i].edges[points[i].length].pointNO = indexBestSimi;
          points[i].edges[points[i].length].similarity = bestSimi;
          points[i].length += 1;
          if(points[i].length > K5_nearest)
            {
              printf("length of the edges [%d] exceeds K5_nearest [%d]! bestSimi is %f \n",
                      points[i].length,K5_nearest,bestSimi);
              return -1;
            }
      }
      else {
          /* check whether this edge has already been added from the
           * other point.
           */
          flag = 0;
```

```
            for(l=0; l<points[indexBestSimi].length; l++){
              if(points[indexBestSimi].edges[l].pointNO == i){
                flag = 1;
                break;
              }/* end if */
            }/* end for l */

if(flag == 0){
points[indexBestSimi].edges[points[indexBestSimi].length].pointNO = i;
points[indexBestSimi].edges[points[indexBestSimi].length].similarity =
bestSimi;
              points[indexBestSimi].length += 1;
              if(points[indexBestSimi].length > K5_nearest)
              {
                printf("length of the edges exceeds K5_nearest!bestSimi is %f \n",
                        bestSimi);
                return -1;
              }
            }/* end if */
        }/* end else */

        similarity[indexBestSimi] = 0.0;

      }/* end for k */

    }/* end for i */


#ifdef Debug_establish_hyperGraph
    j =0;
    printf("hyper edge for point %d is: \n", j);
    for(i=0; i<points[j].length; i++){
      printf("%d  %f \n", points[j].edges[i].pointNO,  points[j].edges[i].similarity);
    }/* end for */
    j = 36;
    printf("hyper edge for point %d is: \n", j);
    for(i=0; i<points[j].length; i++){
      printf("%d  %f \n", points[j].edges[i].pointNO,  points[j].edges[i].similarity);
    }/* end for */
    j = 49;
    printf("hyper edge for point %d is: \n", j);
    for(i=0; i<points[j].length; i++){
      printf("%d  %f \n", points[j].edges[i].pointNO,  points[j].edges[i].similarity);
    }/* end for */
#endif

    free(similarity);
    similarity = NULL;
    return 1;

}/* end establish_hyperGraph */
```

```
/*****************************************************************
 * Name    : initialize                                          *
 * Function: initialize data                                        *
 * Input   :  void                                              *
 * Output  : int                                                *
 *****************************************************************/
static int initialize() {
  int i;

  bestGoodness = 0.0;

  groupIndex = 0;

  index_matlab = 0;

  for(i=0; i<MAXGROUPS; i++){
    groups[i]=NULL;
    groupLength[i]=0;
  }

  /* this is the threshold of size of node to stop partition */
  threshold = (int)(N*MINSIZE);
  if(threshold < 1)
    {
      printf("threshold less than 1, error! \n");
      return -1;
    }

  root = (struct node *)calloc(1, sizeof(struct node));
  if(root == NULL)
    {
      printf("cannot allocate memory for root! \n");
      return -1;
    }



  root->numberPoints = N;
  root->points = (int *)calloc(N, sizeof(int));
  if(root->points == NULL)
    {
      printf("cannot allocate memory for root->points! \n");
      return -1;
    }
  /* for the root node, all the points belong to it */
  for(i=0; i<N; i++)
    {
      (root->points)[i]=i;
    }

#ifdef Debug_initialize
```

```
  for(i=0; i<N; i++)
    {
     printf("(root->points)[%d]=%d \n", i, (root->points)[i]);
    }
#endif

  /* initialize points */
  for(i=0; i<N; i++)
    {
     points[i].length = 0;
    }/* end for i */

  return 1;
}/* end initialize */

/*****************************************************************
 * Name    : readData                                            *
 * Function: read in Data from the data file                     *
 * Input   :  void                                               *
 * Output  : void                                                *
 *****************************************************************/
/*
 * May 23th, 2001. Weinan: readData() has been tested
 * being right.
 */
static int readData() {

  int i;

 /* open the data file */
  if ((fp = fopen(fileName, "r")) == NULL )
    {
     printf("cannot open input file correctly! \n");
     return -1;
    }

  /* Now start reading data */
  for (i=0; i<N; i++)
    {
     fscanf(fp, "%f%f", &points[i].x, &points[i].y);
    }/* end for i */

  return 1;

}/*end readData */

/*****************************************************************
 * Name    : parsingInput                                        *
 * Function: parsing input command line                          *
 * Input   : argc, argv                                          *
 * Output  : void                                                *
 *****************************************************************/
```

```
/*
 * May 23th, 2001. Weinan: parsingInput() has been tested
 * being right.
 */
static int parsingInput(int argc, char *argv[]) {

  if(argc == 4)
    {
     sscanf(argv[1], "%d", &N);
     if(N>MAXSIZE)
          {
           printf("The size of the data set is bigger than MAXSIZE! \n");
           return -1;
          }

     if ((int)strlen(argv[2]) >= MAX_fileName)
          {
           printf("the name of the file is too long. "
                  "It should be within %d characters \n",
                  MAX_fileName);
           return -1;
          }
     else
          {
           sscanf(argv[2], "%s", fileName);
          }

     sscanf(argv[3], "%d", &stopClusterNO);
     if(stopClusterNO>N)
          {
           printf("stopClusterNO is now bigger than N, impossible! \n");
           return -1;
          }
    } /* end if */
  else
    {
     printf("argument error \n");
     printf(" Usage of the program should be of the form: \n");
     printf(" chameleon N fileName stopClusterNO \n");
     return -1;
    }/* end else */

#ifdef Debug_parsingInput
  printf("Input data is: \n ");
  printf("N = %d \n", N);
  printf("fileName = %s \n", fileName);
  printf("stopClusterNO = %d \n", stopClusterNO);
#endif

  return 1;

}/* end parsingInput */
```

5.2. Implementacja wizualizacji w języku c#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Text.RegularExpressions;
using System.Globalization;
using System.Threading;

namespace ChameleonFilter
{
   public partial class Form1 : Form
   {
      List<Pobj> dataset     = new List<Pobj>();    // list of virgin vectors
      List<Pobj> groups      = new List<Pobj>();    // list of grouped (clustered) vectors
      List<Cluster> clusters  = new List<Cluster>(); // list of clusters, each cluster got different
cluster ID
      List<Color> cols       = new List<Color>();   // list of generated colors
      List<Raster> rasters    = new List<Raster>();  // list of rasters eq list of clusters

      // forms
      DrawForm myForm        = new DrawForm();
      RemoveNoise cmyForm     = new RemoveNoise();

      int mySize             = 10;
      int draw_w             = 850;
      int draw_h             = 600;

      // images
      Image myPicture        = null;
      Image cmyPicture       = null;

      // noise param
      int population         = 0;
      int minX               = 0;
      int minY               = 0;
      int maxX               = 0;
      int maxY               = 0;
      int m_raster_size      = 1;

      // buttons
      int method             = 0;

      public Form1()
      {
```

```
    InitializeComponent();

    // draw form
    myForm.Width      = draw_w;
    myForm.Height     = draw_h;
    myForm.Paint     += new PaintEventHandler(paint2);
    myForm.MouseMove += new MouseEventHandler(wheel_MouseMove);
    myForm.Show();

    // form after removing noise
    cmyForm.Width    = draw_w;
    cmyForm.Height   = draw_h;
    cmyForm.Paint   += new PaintEventHandler(cpaint2);
    //cmyForm.MouseMove += new MouseEventHandler(wheel_MouseMove);

    //ini vars
    getcolors();

    // components


}

private void Form1_Paint(object sender, PaintEventArgs e)
{

  // e.Graphics.DrawLine(Pens.Black, 0, 0, 88, 88);
}

private void Form1_Load(object sender, EventArgs e)
{

}

public void getcolors() {
Color c;
int rstep = 25;
int gstep = 45;
int bstep = 55;

for (int r = 0; r < 255; r = r + rstep)
{
   for (int g = 0; g < 255; g = g + gstep)
   {
      for (int b = 0; b < 255; b = b + bstep)
      {
          c = Color.FromArgb(r,g,b);
          cols.Add(c);
      }
     }
   }
   /*
```

```
        for (int i = 0; i < cols.Count; i++)
        {
            Console.WriteLine("i :" + i.ToString() + " color: " + cols[i].ToString());
        }*/
    }

    public String readfile()
    {

        Stream myStream              = null;
        //int size                   = -1;
        string text                  = "";
        string file                  = "";
        OpenFileDialog openFileDialog    = new OpenFileDialog();

        openFileDialog.InitialDirectory  = @"E:\work\studia\V sem\Metody wspolczesnej
algorytmiki\apps\chameleon_parse\Debug";
        openFileDialog.Filter            = "dat files (*.dat)|*.dat";
        openFileDialog.FilterIndex       = 2 ;
        openFileDialog.RestoreDirectory  = true;

        if(openFileDialog.ShowDialog() == DialogResult.OK)
        {
            try
            {
                if ((myStream = openFileDialog.OpenFile()) != null)
                {
                    file             = openFileDialog.FileName;
                    if (file.EndsWith("t"))
                    {
                        using (myStream)
                        {
                            text = File.ReadAllText(file);
                        }
                    }
                    else
                    {
                        MessageBox.Show("Wybierz plik .dat!");
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: Could not read file from disk. Original error: " +
ex.Message);
            }
        }

        //Console.WriteLine(size); // <-- Shows file size in debugging mode.
        //Console.WriteLine(text); // <-- For debugging use only.
        return text;
    }
```

```
        public void parseDS(String fin) { // parse data set
            if (fin.Length > 0)
            {
            int i       = 0;
            // czytamy wektory 2 wymiarowe
            // odpowiednio na kazdym klastrze ustawiamy kolor obiektu oraz rodzica grupy

            string[] lines = Regex.Split(fin, "\n");
            string[] vecs;

                foreach (string line in lines)
                {
                    if(!String.IsNullOrEmpty(line)) {
                        try
                        {
                            i++;
                            listBoxDATASET.Items.Add(line);
                            vecs = line.Split(new Char[] { ' ' });
                            float x = float.Parse(vecs[0], CultureInfo.InvariantCulture.NumberFormat);
                            float y = float.Parse(vecs[1], CultureInfo.InvariantCulture.NumberFormat);
                            //tmp.setX(x);
                            //tmp.setY(y);
                            //Console.WriteLine("i: " + i.ToString() + " x: " + tmp.getX().ToString() + "
y: " + tmp.getY().ToString());
                            dataset.Add(new Pobj(x, y, 0));
                        }
                        catch (Exception e)
                        {
                            MessageBox.Show("Błąd pliku!");
                            break;
                        }
                    }

                }
                RozmiarTxtBox.Text = i.ToString();

            } else {
                MessageBox.Show("Pusty plik!");
            }
        }

        public void parseCL(String fin) // parse clusters
        {
            int cluster_no = -1;

            if (fin.Length > 0)
            {
                //String fout  = "";

                // czytamy klastry
                // odpowiednio na kazdym klastrze ustawiamy kolor obiektu oraz rodzica grupy
```

```
            string[] lines   = Regex.Split(fin, "\r\n");
            string[] vecs;

            /*
             *  Random generator = new Random(DateTime.Now.Millisecond);
                                int red = generator.Next(256);
                                int green = generator.Next(256);
                                int blue = generator.Next(256);
                                Color newCol = new Color.FromArgb(red, green, blue); */

            bool restart = true;
            int  ix,iy   = 0;

            foreach (string line in lines)
            {
               listBoxCLUSTERS.Items.Add(line);

               if (line.StartsWith("["))
               {
                  cluster_no++;

                  if (cluster_no > 0)
                  {
                     //    Cluster(int id, float minX, float minY, float maxX, float maxY)
                     clusters.Add(new Cluster(cluster_no-1, minX, minY, maxX, maxY));
                     rasters.Add(new Raster());
                     restart = true;
                  }
               } else {
                  if (!String.IsNullOrEmpty(line))
                  {
                     try
                     {
                        vecs    = line.Split(new Char[] { ' ' });
                        float x = float.Parse(vecs[0],
CultureInfo.InvariantCulture.NumberFormat);
                        float y = float.Parse(vecs[1],
CultureInfo.InvariantCulture.NumberFormat);
                        ix      = (int)x;
                        iy      = (int)y;

                        if (restart)
                        {

                           minX = ix;
                           minY = iy;
                           maxX = ix;
                           maxY = iy;

                           restart = false;
                        }
```

```
                    if (ix < minX) { minX = ix; }
                    if (iy < minY) { minY = iy; }
                    if (ix > maxX) { maxX = ix; }
                    if (iy > maxY) { maxY = iy; }

                    groups.Add(new Pobj(x, y, cluster_no));
                }
                catch (Exception e)
                {
                    MessageBox.Show("Błąd pliku! "+e.InnerException.ToString());
                    break;
                }
            }
        }

    }
}
else
{
    MessageBox.Show("Pusty plik!");
}

clusters.Add(new Cluster(cluster_no - 1, minX, minY, maxX, maxY));
rasters.Add(new Raster());

ClusterstextBox.Text = (cluster_no+1).ToString();

/*
String cstr = "";
for (int i = 0; i < clusters.Count(); i++)
{
    cstr  = i.ToString() + " = min [" + clusters[i].getXmin().ToString();
    cstr += " , " + clusters[i].getYmin().ToString() + "]";
    cstr += " ; max[" + clusters[i].getXmax().ToString();
    cstr += " , " + clusters[i].getYmax().ToString() + "]";
    cstr += "\n";
    MessageBox.Show(cstr);
}
*/

}

private void openBtnDATASET_Click(object sender, EventArgs e)
{
    dataset.Clear();
    listBoxDATASET.Items.Clear();
    parseDS(readfile());
}
```

```
private void button2_Click(object sender, EventArgs e)
{
   groups.Clear();
   clusters.Clear();
   rasters.Clear();
   listBoxCLUSTERS.Items.Clear();
   parseCL(readfile());
}

public void paint2(object sender, PaintEventArgs e)
{
      if (myPicture != null)
      {
         e.Graphics.DrawImage(myPicture, 0, 0, draw_w, draw_h); //draw the image

      }
}

public void cpaint2(object sender, PaintEventArgs e)
{
   if (cmyPicture != null)
   {
      e.Graphics.DrawImage(cmyPicture, 0, 0, draw_w, draw_h); //draw the image

   }
}

public void CreatePicture(Image cin,int data)
{
   Image canvas;

   if (cin != null)
   {
      canvas = cin;
   }
   else
   {
      canvas = new Bitmap(draw_w, draw_h);
   }

   // create an object that will do the drawing operations
   Graphics g           = Graphics.FromImage(canvas);
   g.SmoothingMode      = SmoothingMode.AntiAlias;
   Color    c           = Color.Aqua;
   Color   pc           = Color.Aqua;
   SolidBrush solidBrush = new SolidBrush(Color.Black);
   // wymiary
   float x              = 0;
   float y              = 0;
   float width          = (float)mySize;
   float height         = (float)mySize;
```

```
        //
        int j               = 0;
        int gr              = 0;
        int maxgr           = 50;
        int step            = 10;
        int legend_s        = 10;
        int legend_x        = draw_w - (legend_s * 3);

        // MessageBox.Show(dataset.Count.ToString());ni
        if (data == 0)
        {
           for (int i = 0; i < dataset.Count; i++)
           {
              x           = dataset[i].getX();
              y           = dataset[i].getY();
              //Console.WriteLine("i: "+i.ToString()+" x: " + x.ToString() + " y: " +
y.ToString());
              g.FillEllipse(solidBrush, x, y, width, height);
              myPicture    = canvas;
              //Thread.Sleep(20);
              myForm.Refresh();
              //myForm.Invalidate();
           }
        }

        if ( (data == 1) || (data == 4) )
        {
           for (int i = 0; i < groups.Count; i++)
           {

              //c           = groups[i].getC();
              c = cols[groups[i].getP()];
              solidBrush = new SolidBrush(c);

              if (!pc.Equals(c))
              {
                 gr++;
                 if (gr == maxgr) { legend_x = legend_x + legend_s; j = 0; }

                 //MessageBox.Show("pc : " + pc.ToString() + " ? c: " + c.ToString());
                 g.FillRectangle(solidBrush, legend_x, j, legend_s, legend_s);
                 //g.FillEllipse(solidBrush, 800, 0+j, width, height);
                 j = j + step;
              }

              pc = c;

              x = groups[i].getX();
              y = groups[i].getY();

              //Console.WriteLine("i: "+i.ToString()+" x: " + x.ToString() + " y: " +
y.ToString());
```

```
                g.FillEllipse(solidBrush, x, y, width, height);

                if (data == 1)
                {
                   myPicture = canvas;
                   myForm.Refresh();
                }
                else
                {
                   cmyPicture = canvas;
                   cmyForm.Refresh();
                }
                //myForm.Invalidate();


             }
          }
          else
          {
             for (int i = 0; i < clusters.Count(); i++)
             {
                g.DrawRectangle(Pens.Red, clusters[i].getXmin(), clusters[i].getYmin(),
clusters[i].getXmax() - clusters[i].getXmin(), clusters[i].getYmax() - clusters[i].getYmin());
                myPicture = canvas;
                myForm.Refresh();
             }
          }
          // now the drawing is done, we can discard the object

          g.Dispose();

          myForm.Invalidate();
          cmyForm.Invalidate();
          // return the picture
       }

       private void drawBtnDATASET_Click(object sender, EventArgs e)
       {
          CreatePicture(null,0); //create and store dataset
       }

       private void button1_Click(object sender, EventArgs e)
       {
          CreatePicture(myPicture,1); //create and store clusters
       }

       public void wheel_MouseMove(object sender, MouseEventArgs e) {
          StatusLab.Text = "vec[" + e.X.ToString() + "," + e.Y.ToString() + "]";

       }

       private void button4_Click(object sender, EventArgs e)
```

```
{
   myPicture.Dispose();
   myPicture = null;
   myForm.Refresh();
}

private void button5_Click(object sender, EventArgs e)
{
   myForm.Show();
}

private void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
{

}

private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
   //myPicture.Dispose();
}

private void SizeTextBox_TextChanged(object sender, EventArgs e)
{
   mySize = Int32.Parse(SizeTextBox.Text);
}

private void button6_Click(object sender, EventArgs e)
{
   CreatePicture(myPicture, 3); //create boxes for raster
}

private void button3_Click(object sender, EventArgs e)
{
   m_raster_size = Int32.Parse(GridSizeTextBox.Text);
   population    = Int32.Parse(textBox5.Text);
   cmyForm.Show();

   switch (method)
   {
      case 0:    //chosen = "raster";
                 run_raster();
                      break;
      case 1:    //chosen = "radius";
                // run_radius();
                      break;
      case 2:    //chosen = "eucl";
                // run_euclidean();
                      break;
      default:   //chosen = "def";
                      break;
   }
```

```
            CreatePicture(null, 4); //show clusters after removing noise

        }

        public void run_raster()
        {
            Color pc    = Color.AliceBlue;
            Color c     = Color.AliceBlue;
            int   gr    = -1;
            int grid_id = 0;
            float x     = 0;
            float y     = 0;
            int   minX  = 0;
            int   minY  = 0;
            int   maxX  = 0;
            int   maxY  = 0;

            for (int i = 0; i < groups.Count; i++)
            {
                c = cols[groups[i].getP()];

                //if ( ( !pc.Equals(c) ) || (i == 0) )

                if ( (!pc.Equals(c)) )
                {
                    gr++;
                    minX = clusters[gr].getXmin();
                    minY = clusters[gr].getYmin();
                    maxX = clusters[gr].getXmax();
                    maxY = clusters[gr].getYmax();

                    //MessageBox.Show("cluster: " + gr.ToString() + " minX[" + minX.ToString() +
"] minY[" + minY.ToString() + "] maxX[" + maxX.ToString() + "] maxY[" +
maxY.ToString()+"]");
                    rasters[gr].size = m_raster_size;
                    rasters[gr].ini_grids(minX, minY, maxX, maxY);
                    //rasters[gr].print_grids();

                }

                pc      = c;
                x       = groups[i].getX();
                y       = groups[i].getY();

                grid_id = rasters[gr].return_gridID(minX, minY, x, y);
                groups[i].setGridID(grid_id);                 // set grid id
                rasters[gr].inc_grid_population(grid_id);  // increase population of each grid in
current raster for current cluster
            }

            // we have to check each grid in each raster for desired grids
            // if population in grid is less than we want we have to remove grid
```

```
            for (int i = 0; i < rasters.Count; i++)
            {
               //MessageBox.Show("raster & cluster = " + i.ToString());
               //rasters[i].print_grids();
               rasters[i].check_grids(population);
            }

            // now we have to check each object in each cluster
            // if the grid in each raster of each cluster was remove we will remove object in order
to remove noise
            String debug = "";

            for (int i = 0; i < groups.Count; i++)
            {
               gr      = groups[i].getP();
               grid_id = groups[i].getGridID();
               x       = groups[i].getX();
               y       = groups[i].getY();

               debug   = "raster & cluster = " + gr.ToString() + " vec[" + x.ToString() + "," +
y.ToString() + "]=" + grid_id.ToString();

               if ( !(rasters[gr].grid_exist(grid_id)) )
               {
                  groups.RemoveAt(i);
                  debug += "\nWektor znajduje sie w kratce o malej ilosci obiektow, usuwam!";
               }

               //MessageBox.Show(debug);
            }
         }

         private void MetodaComboBox_SelectedIndexChanged(object sender, EventArgs e)
         {
            method = MetodaComboBox.SelectedIndex;
         }

         private void GridSizeTextBox_TextChanged(object sender, EventArgs e)
         {

         }

         private void textBox5_TextChanged(object sender, EventArgs e)
         {

         }

         private void button7_Click(object sender, EventArgs e)
         {
            cmyForm.Show();
         }
```

```
    }
}

class Raster
{
    public int size    = 1;
    private int dx     = 0;
    private int dy     = 0;
    //private float px   = 0;
    //private float py   = 0;
    List<int> grid     = new List<int>(); // each grid got number of elements based on this fact we can remove noise

    public Raster()
    {

    }

    public void ini_grids(int minX, int minY, int maxX, int maxY)
    {
        //grid.Clear();
        dx = Math.Abs(maxX - minX) / size;
        dy = Math.Abs(maxY - minY) / size;
        //dodajemy o 1 do dx i dy ze wzgledu na wychodzenie poza zakres siatki rastera

        for (int i = 0; i < ((dx+1) * (dy+1)); i++)
        {
            grid.Add(0);
        }
    }

    public int return_gridID(int minX, int minY, float p_x, float p_y)
    {
        int result = 0;
       // px    = p_x;
       // py    = p_y;
        int x = ((int)p_x - minX) / size;
        int y = ((int)p_y - minY) / size;

        y = y * dx;
        result = x + y;
       // MessageBox.Show("dx :"+dx.ToString()+" dy :"+dy.ToString()+" p_x: " + p_x.ToString() + " p_y: " + p_y.ToString() + " minX: " + minX.ToString() + " minY: " + minY.ToString() + " = " + result.ToString());
        //MessageBox.Show("dx :" + dx.ToString() + " dy :" + dy.ToString() + " p_x: " + p_x.ToString() + " p_y: " + p_y.ToString() + " = " + result.ToString());
        return result;
    }

    public void inc_grid_population(int grid_no)
```

```
    {
        if (grid_no < 0)
        {
            MessageBox.Show("Grid ID below 0! " + grid_no.ToString());
            grid_no = 0;
        }

        if (grid_no > grid.Count()-1)
        {
            //MessageBox.Show("Grid ID out of index! " + grid_no.ToString()+ "dx :" +
dx.ToString() + " dy :" + dy.ToString() + " x: " + px.ToString() + " y: " + py.ToString() );
            grid_no = grid.Count()-1;
        }

            grid[grid_no] = grid[grid_no] + 1;
    }

    public void check_grids(int population)
    {
        for (int i = 0; i < grid.Count(); i++)
        {
            if (grid[i] < population)
            {
                //grid.RemoveAt(i);
                grid[i] = 0;
            }
        }
    }

    public bool grid_exist(int grid_no)
    {
        bool result = false;
        int  g_max  = grid.Count()-1;

        if (grid_no > g_max)
        {
            MessageBox.Show("Index "+grid_no.ToString()+" out of range!
["+g_max.ToString()+"]");
            grid_no = g_max;
        }

        if (grid[grid_no] != 0)
        {
            result = true;
        }

        return result;
    }

    public void print_grids()
    {
        String result = "";
```

```
        for (int i = 0; i < grid.Count(); i++)
        {
            result += "[" + i.ToString() + "] = " + grid[i] + "\t ";
        }

        MessageBox.Show(result + "\n dx = " + dx.ToString() + " \n dy = " + dy.ToString());

    }


}

class Cluster
{
    int m_minX, m_minY, m_maxX, m_maxY;
    int m_id;

    public Cluster(int id, int minX, int minY, int maxX, int maxY)
    {
        m_id   = id;
        m_minX = minX;
        m_minY = minY;
        m_maxX = maxX;
        m_maxY = maxY;
    }

    public int getXmin()
    {
        return m_minX; // vec[x]
    }

    public int getYmin()
    {
        return m_minY; // vec[y]
    }

    public int getXmax()
    {
        return m_maxX; // vec[x]
    }

    public int getYmax()
    {
        return m_maxY; // vec[y]
    }

    public int getID()
    {
        return m_id; // cluster id
    }
```

```
}

class Pobj
{
   float m_X, m_Y; // coordinates
   int   m_P;      // ID of parent cluster
   int   m_G;      // ID of grid in raster method

   public Pobj(float x, float y, int p)
   {
      m_X = x; // vec[x]
      m_Y = y; // vec[y]
      m_P = p; // cluster parent
      m_G = 0;
   }

   public void setX(float x)
   {
      m_X = x; // vec[x]
   }

   public void setY(float y)
   {
      m_Y = y; // vec[y]
   }

   public void setP(int p)
   {
      m_P = p; // cluster parent
   }

   public float getX()
   {
      return m_X; // vec[x]
   }

   public float getY()
   {
      return m_Y; // vec[y]
   }

   public void setGridID(int grid_no)
   {
      m_G = grid_no; // grid ID
   }

   public int getGridID()
   {
      return m_G; // grid id
   }

   public int getP()
```

```
    {
        return m_P; // cluster parent
    }
}
```