Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Jan Navrátil

Hledání shluků v grafech

Katedra Aplikované Matematiky

Vedoucí bakalářské práce: RNDr. Bernard Lidický, PhD
Studijní program: Informatika, Programování

Praha, 2011

Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS

Jan Navrátil

Cluster Analysis on Graphs

Department of Applied Mathematics

Supervisor: RNDr. Bernard Lidický, PhD
Study Program: Computer Science, Programming

Praha, 2011

I would like to thank my supervisor, RNDr. Bernard Lidický, PhD, for his assistance and guiding me on this thesis and all who advised me during the process.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

Prague, August 5, 2011 Jan Navrátil

Název práce: Hledání shluků v grafech
Autor: Jan Navrátil
Katedra: Katedra Aplikované Matematiky
Vedoucí bakalářské práce: RNDr. Bernard Lidický, PhD
E-mail vedoucího: bernard@kam.mff.cuni.cz

Abstrakt: Cílem práce je vytvořit aplikaci, která dokáže hledat shluky v grafech. Aplikace obsahuje na výběr několik modifikovaných algoritmů klasické shlukové analýzy a grafových algoritmů využívaných pro detekci komunit v komplexních sítích. Účelem práce není optimalizace implementace na rychlost ale možnost vyzkoušení a porovnání výsledků jednotlivých algoritmů a ověření, jestli jsou specializované grafové algoritmy skutečně vhodnější. Jádrem úkolu bylo tedy modifikovat algoritmy shlukové analýzy (byly vybrány čtyři, s řadou kombinovatelných nastavení) pro práci nad grafy a společně s algoritmy hledajícími komunity v sítích (dva, oba ve více variantách) je implementovat a prezentovat v rámci jedné aplikace.

Klíčová slova: shluková analýza, grafy, algoritmy, komunity v sítích

Title: Cluster Analysis on Graphs
Author: Jan Navrátil
Department: Department of Applied Mathermatics
Supervisor: RNDr. Bernard Lidický, PhD
Supervisor's e-mail address: bernard@kam.mff.cuni.cz

The goal of this thesis is to create an application which will be able to identify clusters in graphs. The application contains modified algorithms from cluster analysis and graph algorithms used for identifying communities in complex networks. The purpose of this work is not speed optimalisation of implementation but the opportunity to try and compare results of each algorithm and verify whether special graph algorithms are truly better. The core of this task was to modify algorithms from classical cluster analysis (four were chosen, with a set of settings each) to work with graphs and implement and present them within one application along with community detection algorithms (two, in more versions each).
Keywords: cluster analysis graph algorithms

Contents

Introduction

Recent advances have brought out the importance of identifying clusters or communities in networks in various branches of science such as sociology (groups of friends in social or colleagues in collaborative networks), medicine (patients with similar disease in diagnostic systems) or biology (metabolic networks, gene maps). This topic is very popular in research but can also bring economical benefits to technological domains (traffic in the Internet, power grids) and business by the opportunity of improving customer targeting strategies or minimizing distribution expenses.

The demand for improvements motivate applied mathematicians to develop better algorithms for community detection. There are different approaches known to the problem [4][5][6]. This problem in a generalised form is the topic of cluster analysis [1][3].

The goal of this thesis is to explore some algorithms from both domains. Also create an application which will implement those algorithms and which will let us to experiment with them.

The first chapter contains definitions, the second contains used measure functions and metrics. The compared algorithms are described in the third chapter. The implementation aspects of this work are presented in the fourth chapter. The next chapter is a user's manual for the Clusterer application. The algorithms are compared in the seventh chapter. The last chapter is the conclusion and suggestions for further work.

Chapter 1

Preliminaries

In this chapter we define necessary conceptions such as *graph, community, cluster arrangement* and *cluster analysis.*

1.1 Graph

"A *graph* is an abstract representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by mathematical abstractions called *vertices*, and the links that connect some pairs of vertices are called *edges*."[18]

Let $G = (V, E)$ be a graph, where the set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and the set of edges $E \subseteq \{\{a, b\} | a, b \in V, a \neq b\}$. By $dist(u, v)$ we denote the length of the shortest path containing vertices u and v. We define $dist(u, v)$ to be infinity if no such path exists. G is associated with its *distance matrix* $D : D_{ij} := dist(i, j)$ and its *adjacency matrix* $A : A_{ij} = 1 \iff D_{ij} = 1$,

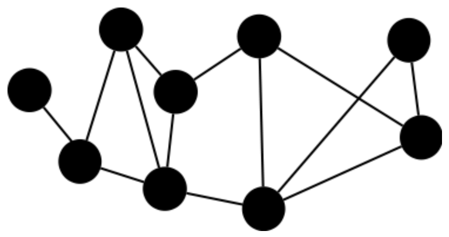


Figure 1.1: an example of a graph

otherwise $A_{ij} = 0$. We denote the class of all graphs by $\mathcal{G}$. Figure 1.1 shows an example of a graph.

1.2 Community

A *Cluster*, usually denoted by C, is a subset of vertices. We denote the class of all clusters of a graph by $\mathcal{C}$. The *size* of a cluster C is denoted by $|C|$. A *Community* is a cluster satisfying the condition of community:

"...the network divides naturally into groups of nodes with dense connections internally and sparser connections between groups."[17]

Note that there is no single precise definition of community. More precise definitions are in section 3.5.

An example of a graph with community structure is shown in 1.2.

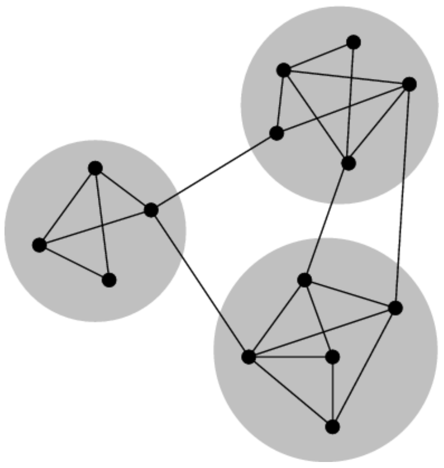


Figure 1.2: An example of a graph with a community structure [17]

1.3 Cluster Arrangement

A *cluster arrangement* K is a partition of the vertices of a graph into clusters $C_1, C_2, \dots C_k$:

$$K = \{C_1, C_2, \dots, C_k\}$$

where k is the size of the cluster arrangement. We denote the class of all cluster arrangements of a graph by $\mathcal{K}$.

1.4 Cluster Analysis

Classical cluster analysis is a tool for disjunctive division of a set of vectors from metric space into classes (clusters). For the purposes of this thesis, the domain had to be changed to the space of graphs and the range to the space of cluster arrangements of graphs.

Chapter 2

Metrics and Energy Functions

To evaluate the quality of a cluster arrangement, two energy functions and one metric are used. Recall that distances between two vertices u and v are available by D_{uv}, giving the length of the edge $e = \{u, v\}$.

2.1 Cluster Energy Function

Energy function, denoted by E, is a function $\mathcal{C} \to \mathbb{R}^+$. These functions are used for expressing the dispersion (deviation, distance from each other) of vertices in the cluster. Some of the methods used for calculating cluster energy of C are:

- **sum of all distances** between vertices of C.

$$E(C) = \sum_{u,v \in C} D_{uv}$$

- **sum of all distances squared**

$$E(C) = \sum_{u,v \in C} D_{uv}^2$$

- **average distance** = sum of all distances over size of the cluster

$$E(C) = \frac{1}{|C|} \sum_{u,v \in C} D_{uv}$$

- **average distance squared** = sum of all distances squared over size of the cluster
$$E(C) = \frac{1}{|C|} \sum_{u,v \in C} D_{uv}^2$$

All these energy functions give us the information how dispersed the vertices are. For a given number of vertices, members of a lower energy cluster are better concentrated and such cluster is usually a better result of the cluster analysis.

2.2 Cluster-Cluster Distance Metric

Cluster-Cluster distance metric (C-C distance), denoted by M, is a function $\mathcal{C} \times \mathcal{C} \to \mathbb{R}^+$. It defines the distance between two clusters. To be considered a metric, the distance function has to meet the metric axioms:

- **axiom of non-negativity:** $M(C_1, C_2) >= 0$
- **axiom of identity:** $M(C_1, C_2) = 0 \iff C_1 = C_2$
- **axiom of symmetry:** $M(C_1, C_2) = M(C_2, C_1)$
- **triangle inequality:** $M(C_1, C_3) <= M(C_1, C_2) + M(C_2, C_3)$

Here are some commonly used C-C distances:

- **Single linkage**[11], also known as the nearest neighbour method. Two clusters are as far away as the nearest pair of vertices, each being from a different cluster.
$$M(C_1, C_2) = \min_{i \in C_1, j \in C_2} \{D_{ij}\}$$
- **Complete linkage**[12], also known as the furthest neighbour method. Two clusters are as far away as is the furthest pair of vertices, each being from a different cluster.
$$M(C_1, C_2) = \max_{i \in C_1, j \in C_2} \{D_{ij}\}$$

- **Average linkage**[1]. The distance of two clusters is the average distance of a pair of vertices, each from the different cluster. (The used average can be either arithmetic or geometric.)

$$M(C_1, C_2) = \frac{\sum_{i \in C_1, j \in C_2} D_{ij}}{|C_1||C_2|}$$

- **Centroid (Representative, Median) linkage**. Given a representative object for a cluster (centre of gravity, median, other), the distance of two clusters can be expressed as the distance of their representatives, denoted by *repr*.

$$M(C_1, C_2) = D_{repr(C_1), repr(C_2)}$$

  Originally[9], this method used gravity centres. However, without an embedding in a space to calculate them, the representative has to be chosen alternatively. In this application, the vertex with the lowest average distance to all other vertices in the cluster is selected, considered the median [10].

- **Average connection** The distance between two clusters it the fraction of the number of edges that connect them and the maximal possible number of interconnecting edges (each pair of vertices from distinct clusters)

$$M(C_1, C_2) = \frac{1}{|C_1||C_2|} \sum_{i \in C_1, j \in C_2} A_{ij}$$

  This metric uses the adjacency matrix instead of the distance matrix.

- **Energy addition** The distance between two clusters is the difference between the energy of the union $C_3 = C_1 \cup C_2$ and the sum of energies of the compared clusters.

$$M(C_1, C_2) = E(C_3) - E(C_1) - E(C_2)$$

  This metric uses a selected energy function.

There might be an unweighted graph on input. The axioms are the reason why it may be necessary to correct the graph to get sensual results. The correcting process sets the length of each edge to the length of the shortest path between the end vertices

2.3 Cluster Arrangement Ranking

Cluster arrangement ranking functions, denoted by $rank(K)$, may depend on the cluster energy function and the distance metric chosen. Here are some commonly used ranking functions:

- **Average energy times average distance**

$$rank(K) = \frac{1}{|K|^3} \sum_{C \in K} E(C) \sum_{C_1, C_2 \in K} M(C_1, C_2)$$

- **Average energy squared times average distance**

$$rank(K) = \frac{1}{|K|^3} (\sum_{C \in K} E(C))^2 \sum_{C_1, C_2 \in K} M(C_1, C_2)$$

- **Sum of energies over sum of distances**

$$rank(K) = \frac{\sum_{C \in K} E(C)}{\sum_{C_1, C_2 \in K} M(C_1, C_2)}$$

- **Sum of energies over number of clusters**

$$rank(K) = \frac{\sum_{C \in K} E(C)}{|K|}$$

- **Modularity**[7] is the difference between the number of edges within clusters and the expected number of such edges.

$$rank(K) = \frac{1}{2m} \sum_{C \in K} \sum_{i,j \in C} \left(A_{ij} - \frac{deg(i)deg(j)}{2m} \right)$$

  where $m = |E|$ is the number of edges and $deg(i)$ is the degree of a vertex i. Unlike other rank methods, it uses the *adjacency matrix* A.

Chapter 3

Algorithms

In this chapter, we review algorithms implemented in our application. Algorithms 3.1, 3.2, 3.3 and 3.4 are from classical cluster analysis. Algorithms 3.5 and 3.6 are specifically created for the problem of identifying communities in networks.

Algorithms from classical cluster analysis had to be modified to work with graphs. In description of each algorithm, the specific modifications are described. However, they still use *distance matrix* which contains the distances between each pair of vertices. An unweighted graph with only *adjacency matrix* can be completed and distances between its vertices can be corrected to the length of the shortest path between certain pair of vertices.

3.1 AGLO - Agglomerative Algorithm

This algorithm was proposed in [15]. It creates a hierarchic cluster arrangement system based on proximity of clusters. In each step, the algorithm merges the two nearest clusters, C_1, C_2 into one, $C_3 = C_1 \cup C_1$, making the new cluster arrangement smaller by one cluster. Ends with $k =$ desired number of clusters.

$$K_{k+1} \to K_k = (K_i \setminus \{C_1, C_2\}) \cup C_3$$

Pseudocode:

```
$K$ := set of clusters, containing a single vertex each
while $|K| > k$ do
    $c_1, c_2 \leftarrow$ the closest clusters
    Merge the $c_1, c_2$ into $c_3$.
    delete $c_1, c_2$ from $K$
    add $c_3$ to $K$
  end while
  return K
```

⊕ Straight idea, simple implementation.

⊕ Stable, resistant to extreme instances.

This algorithm does not need to be significantly modified for working with graphs. It does not even require the energy function because it only merges the nearest clusters regardless of their size (unless the implementation of C-C distance needs it).

3.2 DIVI - Divisive Algorithm, MacNaughton-Smith

This hierarchic divisive clustering algorithm was proposed in [8]. Like the agglomerative algorithm, it creates a hierarchic system of cluster arrangements, but in the reversed order. In each iteration, it tears the cluster with the highest energy (but with more than one vertex) into two. Ends when the required number of clusters is met (or all clusters have only one vertex). Pseudocode:

```
$K$ := set with one cluster containing all $n$ vertices
while $|K| < k$ do
  $U \leftarrow$ the biggest cluster
  if $|U| > 1$ then
    $W \leftarrow$ the furthest vertex
    select $u \in U$ with the highest $q$=(distance from the rest of $U$) - (distance
    from $W$)
    while $q[u] > 0$ do
      move $u$ from $U$ to $W$
      $u \leftarrow$ vertex from $U$ with highest $q$
    end while
  end if
end while
return $K$
```

This algorithm starts with one cluster containing all n vertices and ends with $k \leq n$ clusters. In each step, one cluster is divided into two, therefore there are $n - k$ iterations.

$\oplus$ Stable.

$\ominus$ Sensitive to distant elements which affect the initial phase of tearing.

This algorithm uses both the energy metric (for selection of the most convenient cluster to divide) and the C-C distance (for decision whether to keep moving vertices to the new cluster or stop).

3.3 K-means Algorithm - Forgy (Jancey)

This non-hierarchic optimising algorithm, proposed in [13], does not build up or divide down through a giant structure. It uses a given invariable number of clusters to converge to a solid state. To operate, clusters need to be given *representatives*. If elements were presented as vectors, centroids could be calculated as a gravity centres. Since there is no metric space to work in, a typical element for a representative is selected. It shall be the vertex with the lowest average distance to all other vertices in its cluster, the median. The algorithm pseudocode for k = a given number of clusters:

```
$K$ := set of $k$ clusters.
for $i := 1 \to k$ do
  add a random (but distinct from previous) vertex to the $i$-th cluster.
  set the median of the cluster to that vertex.
end for
repeat
  for $i$ :=1 $\to n$ do
    $C_1 \leftarrow$ the cluster containing $i$.
    $C_2 \leftarrow$ the cluster whose median is the closest to $i$.
    if $C_1 \neq C_2$ then
      move $i$ from $C_1$ to $C_2$.
    end if
  end for
  for $C = 1 \to k$ do
    recalculate the median of $C$-th cluster.
  end for
until no vertex is moved in the loop
```

```
return  $K$
```

- ⊕ Simple idea and easy implementation.
- ⊕ Vertices may move between clusters.
- ⊕ Converges typically in just few steps.
- ⊖ Dependent on mean representation.
- ⊖ May not give the global extreme, only local.
- ⊖ Strongly dependent on random initialisation.

Note on the Jancey method[14]:
Jancey method[1, p.141] is similar to the Forgy, the only difference is the correction of the centroid. Where Forgy always uses the current gravity centre for a representative, Jancey calculates it as the point reflection of the former representative by the current gravity centre. These operations with vectors are not available in the graph version. Therefore, both methods meld during graph conversion.

3.4 Wishart Algorithm (MacQueen, McRae)

A non-hierarchic clustering algorithm based on K-Means, proposed in [16]. The main difference from K-Means is that the modified clusters recalculate their representatives in each step of assigning vertices to clusters, so there is a strong result dependence on the order of vertices inside clusters (or the structure iterated through).

Pseudocode:

```
$K$ := set of $k$ clusters.
for $i := 1 \to k$ do
    add a random (but distinct from previous) vertex to the $i$-th cluster.
    set the median of the cluster to that vertex.
end for
repeat
    for $i := 0 \to n$ do
        $C_1 \leftarrow$ the cluster containing $i$.
        $C_2 \leftarrow$ the cluster whose median is the closest to $i$.
        **if** $C_1 \neq C_2$ **then**
            move $i$ from $C_1$ to $C_2$.
            recalculate the median of $C_2$.
        **end if**
    **end for**
**until** no vertex is moved in the loop
**return** $K$
```

$\oplus$ Converges in few steps.

$\ominus$ Strongly dependent on initialisation and order in vertex container.

3.5 Girvan-Newman Betweenness Algorithm

This algorithm, proposed in [4], identifies communities in complex networks using the *edge betweenness* and a definition of community. Betweenness of an edge e, denoted by $btwns(e)$, is the number of shortest paths between all pairs of vertices that include e.

$$btwns(e) = |\{s_{ij} : e \in s_{ij}, i, j \in V\}|$$

where $s_{ij} \subseteq E$ is the shortest path between vertices i, j.

Split degree: Let C be a cluster and let $v \in C$. The degree $deg(v) = \sum_{u \in V} A_{uv}$ can be split into two contributions: $deg(v) = deg^{IN}(v) + deg^{OUT}(v)$, where $deg^{IN}(v)$ is the number of adjacent vertices in C, $deg^{OUT}(v)$ is the number of connections to the rest of the graph.

Community can be used with two definitions:

Strong Sense Cluster C is a *strong* community if

$$deg^{IN}(v) > deg^{OUT}(v), \forall v \in C.$$

In a strong community, each vertex has more connections within the community than with the rest of the graph.

Weak Sense Cluster C is a *weak* community if

$$\sum_{\forall v \in C} deg^{IN}(v) > \sum_{\forall v \in C} deg^{OUT}(v).$$

In a weak community, the sum of degrees within the community is larger than the sum of degrees with the rest of the graph.

The algorithm removes the edge with the highest betweenness from the graph in each step. When a component breaks down into two, the partition is then analyzed and the valid communities in the selected sense are noted. The algorithm ends when a required number of communities is reached or no edges remain.

Unlike other algorithms described in this thesis, this algorithm does not divide vertices into communities. It finds communities and separates single vertices that are not in a communtity. Results contain clusters that show community structure and clusters with single vertices. This fact complicates cluster arrangement ranking with otherwise useful functions.

Pseudocode:

```
while $|E| > 0 \land k >$ #of communities in $G$ do
  calculate betweenness for each edge
  remove edge with the highest betweenness from $E$
  if the number of components increased then
    $K \leftarrow$ component arrangement of $G$
    check if the components are communities
  end if
end while
return $K$
```

3.6 Pons-Latapy Algorithm Using Random Walks

The progress of this algorithm, proposed in [6], is similar to AGLO from 3.1, but before agglomerating, it computes the *transition matrix* and during its work the algorithm uses its own instance of cluster-cluster distance metric considering the probability of being visited by a *random walk.*

For this algorithm, the degree $d(i) = \sum_j A_{ij}$ (using a different notation for on purpose) of the vertex i is the number of its neighbors $\underline{\text{(including itself)}}$. Let us consider a discrete random walk process (or diffusion process) on the graph G. At each time step, a walker is on a vertex and moves to a vertex chosen randomly and uniformly among its neighbors (including itself, thus staying). The sequence of visited vertices is a Markov chain, the states of which are the vertices of the graph. At each step, the transition probability from vertex i to vertex j is $P_{ij} = \frac{A_{ij}}{d(i)}$. This defines the transition matrix P of the random walk.

The process is driven by the powers of P: the probability of going from i to j through a random walk of length t is $(P^t)_{ij}$. In the following, this probability will be denoted by P^t_{ij} [6]

The cluster-cluster distance metric is formulated as follows:

$$M(C_1, C_2) = \sqrt{\sum_{u \in V} \frac{\frac{1}{|C_1|}\sum_{i \in C_1} P^t_{iu} - \frac{1}{|C_2|}\sum_{i \in C_2} P^t_{iu}}{d(u)}}$$

Pseudocode:

```
$P$ := transition matrix for $G$, powered to $t$.
$K$ := set of $n$ clusters, containing a single vertex each
while $|K| > k$ do
   $C_1, C_2 \leftarrow$ the closest clusters chosen by the Random Walk cluster-cluster distance
   $C_3 \leftarrow C_1 \cup C_2$
   delete $C_1, C_2$ from $K$
   add $C_3$ to $K$
end while
return $K$
```

Authors of [6] suggest choosing t with $\mathcal{O}(\log(n))$. Too high values of t tend to only refer to $d(j)$ for P^t_{ij}, too small values of t do not carry enough information about the network topology.

Chapter 4

Implementation

The application is programmed in Java 6 [20], Eclipse IDE [19] was used during development. The application uses Swing GUI libraries. Java was chosen for its compatibility with multiple platforms.

4.1 Architecture

The application runs the main class, `Clusterer`, whose method `main(String[])` creates the GUI and allows the user to move through the work flow 5.2. The system of classes is shown in scheme 4.1. For easier access, class `Clusterer` is never instantiated and has static variables for maintaining data (graph and cluster arrangement), implementations of analysis methods (algorithms, functions and metrics) and all present types of output (graphics, additional graphics, log and alternative results).

4.2 Data Implementation

Our basic data structure is a graph. Graphs are implemented as instances of `Graph` containing a data structure, which represents **BOTH** the distance matrix D and the adjacency matrix A. This data structure is stored in an instance of `DistanceMatrix` which actually contains only the lower triangle half of the matrix because the entire square matrix would contain duplicate information due to symmetry of distance relation. Data in `DistanceMatrix` are stored in triangle array `float[][]`. Values A_{ij} and D_{ij} are accessed by functions `boolean isEdgeAdjacent(int i,int j)` and `float getDist(int i,int j)` in `Graph`,

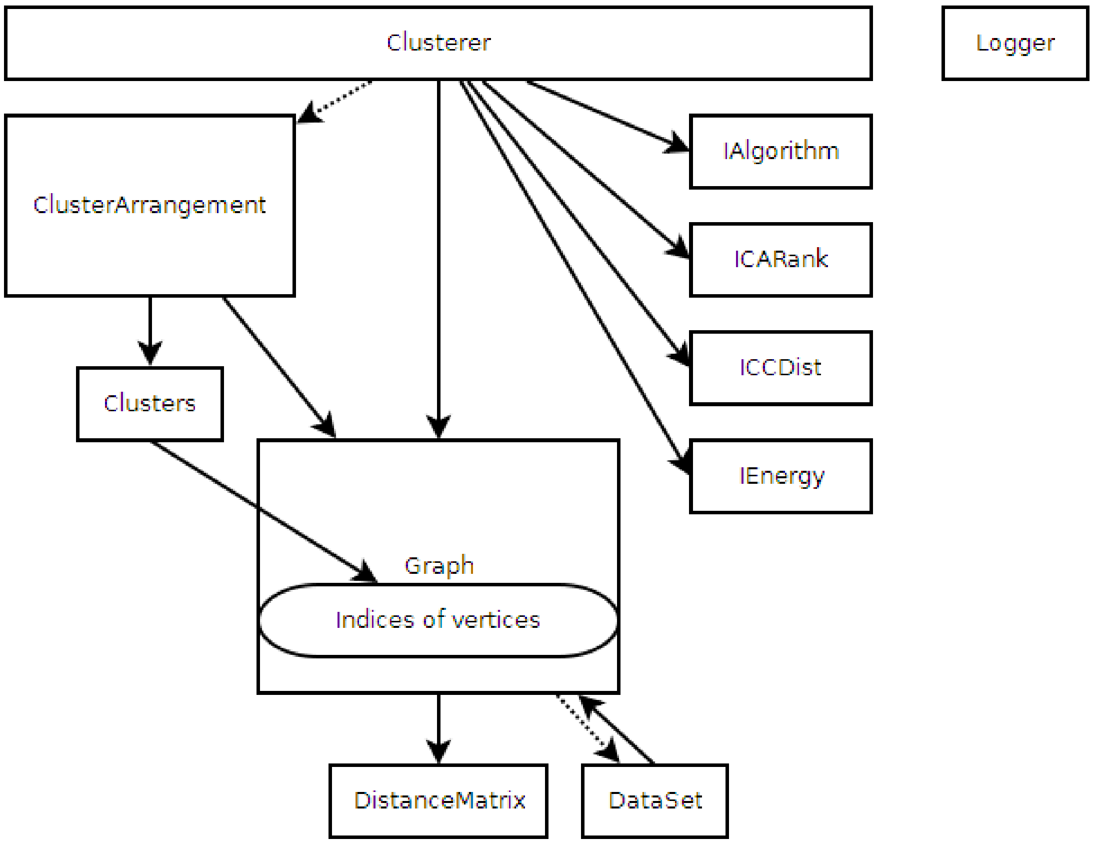


Figure 4.1: The scheme of classes

where i, j are indices of vertices.

`Graph`s may also contain a set of points in space, represented by an instance of `DataSet`. The data structure of this class is `Vector<Point>`, where `Point` is the representation of a point in space. The `DataSet` can not be processed on its own, therefore bringing it into the application will create a `Graph`, which is later processed.

4.3 Interfaces

Four interfaces are used in this application: `IAlgorithm` for implementation of algorithms, `ICARank` for implementations of cluster arrangement ranking functions, `ICCDist` for cluster-cluster distance metrics and `IEnergy` for energy functions.

These interfaces allow the application to work independently on the selected implementations.

Classes that implement each interface are:

- `IAlgorithm` Classes that implement this interface are implementations of clustering or community detection algorithms. This interface defines two methods: `ClusterArrangement getResult(Graph)`, which is meant to run the entire algorithm and return the result with the highest rank; and `ClusterArrangement getResult(Graph, int)`, which returns a result with the specified number of clusters.

  Class `GenericAlgorithm` is actually the only implementor of `IAlgorithm`. It was created to reduce duplicate code but with further development. This class is kept in the project for the case of necessary code insertions. All algorithms now extend this class and override absctract methods `ClusterArrangement calculateResult(Graph)` and `ClusterArrangement calculateResult(Graph, int)`, which actually perform the algorithm.

  Names of all classes extending `GenericAlgorithm` and therefore implementing `IAlgorithm` begin with `"Alg_*"`.

- `ICARank` This interface is implemented by cluster arrangement ranking functions. It defines only one method, `float rank(ClusterArrangement)`, which returns the rank of the given `ClustererArrangement`. This method is meant and allowed to statically access `Clusterer.ene` and `Clusterer.ccd`, which are variables for selected energy function and cluster-cluster distance.

  Names of all classes implementing this interface begin with `"CARank_*"`.

- `ICCDist` This interface is implemented by cluster-cluster distance metrics. It defines only one method, `float dist(Cluster, Cluster)`, which returns the cluster-cluster distance of entered clusters. This method may statically access `Clusterer.ene`.

  Names of all classes implementing this interface begin with `"CCDist_*"`.

- `IEnergy` This interface is implemented by cluster energy functions. It defines only one method, `float calc(Cluster)`, which returns the energy of the entered cluster.

  Names of all classes implementing this interface begin with `"Energy_*"`.

Chapter 5

User's manual

5.1 System requirements and installation

System requirements:

- OS with GUI
- Java 1.6

Installation and Running:

- Using Subversion + Ant
  - checkout with command
    `svn co https://clusterer.svn.sourceforge.net/svnroot/clusterer clusterer`
  - build and run with Ant by command
    `ant -f "clusterer/build.xml" Clusterer`
- Using a JAR archive
  - download the JAR from
    `http://sourceforge.net/projects/clusterer/files/`
  - Either run it as an application
  - Or Java it with command
    `java -cp "Clusterer.jar" com.sf.clusterer.Clusterer`

User can download the testing data

- as an archive from sourceforge
  `http://sourceforge.net/projects/clusterer/files/`
- individually from reference [21]
  `http://www.cc.gatech.edu/dimacs10/archive/clustering.shtml`

5.2 Work Flow

This is the idea of how work with Clusterer should look like:

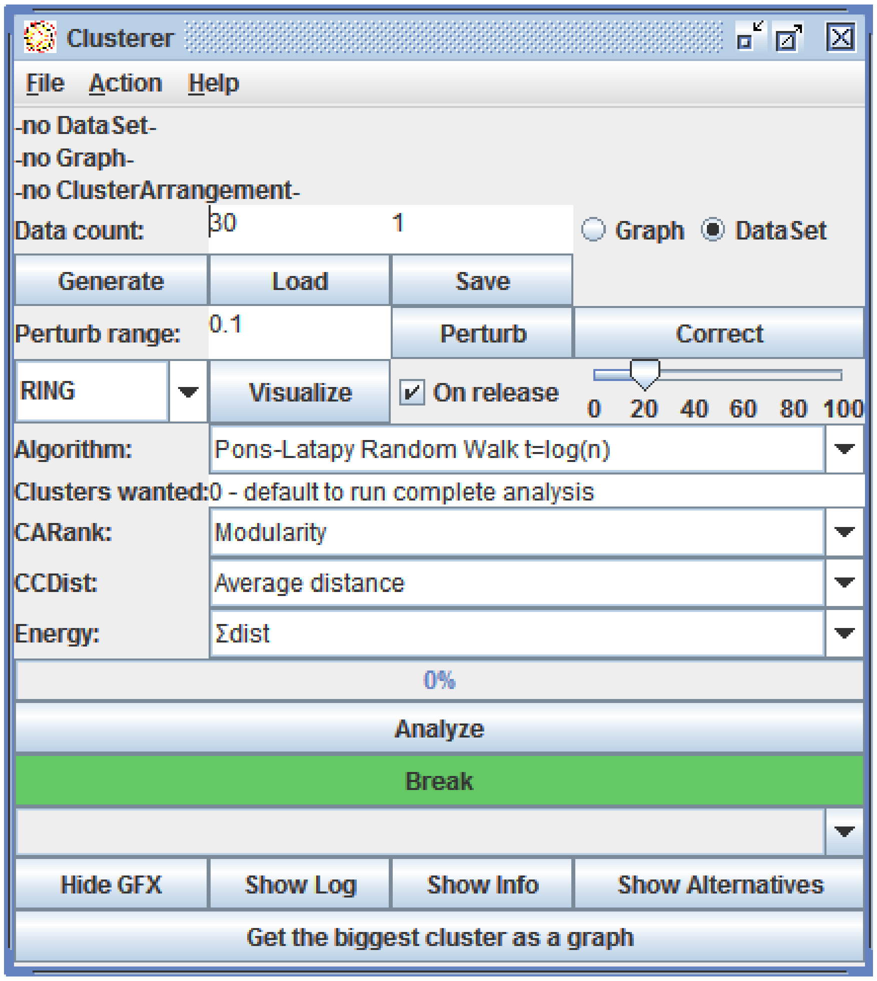
1. Obtain Data
   - Generate 5.4
   - Load from a file 5.5
2. Perturb data? 5.6.1
3. Correct data 5.6.2
4. Select an Algorithm, Cluster Arrangement ranking function, Cluster-Clsuter distance metric and an Energy function.
5. Run an analysis 5.7
6. Browse results 5.8.4

5.3 GUI

The application has a simple and user-friendly GUI. All tasks can be performed by clearly named buttons, all selections are made in combo boxes and the main actions are also available from the menu. All items in the window have a tooltip comment. See a screensot in figure 5.1.

5.4 Data Generating

It is possible to generate input data with some random properties. However, these random data are not suitable for community detection algorithms because the randomness is applied to edge distances, not edge existence. However, these data



-no DataSet-
-no Graph-
-no ClusterArrangement-

Data count: 30 1 Graph [✓ DataSet]

Perturb range: 0.1

RING ✓ On release

Algorithm: Pons-Latapy Random Walk t=log(n)

Clusters wanted: 0 - default to run complete analysis

CARank: Modularity

CCDist: Average distance

Energy: Σdist

0%

Figure 5.1: Overview of the GUI main window

Using two text fields, you can select the number of nodes and clusters. All the clusters will be the same size (total amount of data is moduled by the number of clusters).

It is possible to either generate a graph (see 5.4.1) or a DataSet (see 5.4.2).

5.4.1 Generating a Graph

Each cluster is generated as a separate graph and is connected to the result Graph by a set of Edges, again randomly generated, but with doubled maximal

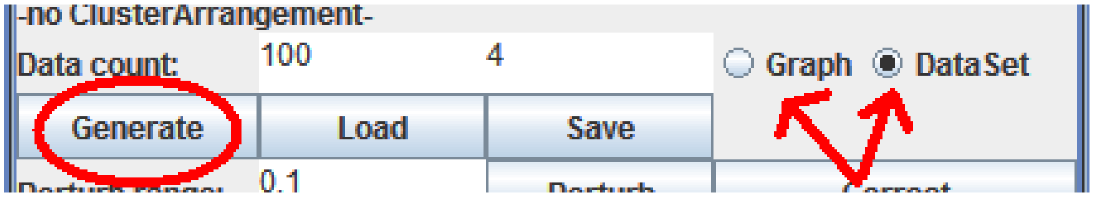


Figure 5.2: The data generating section of GUI

length.

Since this weighted graph is very probably not meeting the metric requirement, especially the triangle inequality, it has to be corrected before an analysis can be performed.

5.4.2 Generating a DataSet

The points are put into a plain (2D). This dimension was chosen for its ease of visualisation and imagination.

Points within a cluster randomly fill a square in plain. Each cluster has a randomly generated offset. Once the DataSet is generated, a graph is created, using all the points from DataSet as vertices and adding the complete set of edges, whose length is set to Euclid distance of the endpoints of each edge.

5.5 Data Loading

There are various ways to represent data, which is the reason for multiple supported file formats. The application is able to load a data file in these formats:

own DataSet format: *.ds
  first line: <dimension of data>
  remaining lines: {<data coordinates>*dimension}

PAJEK Graph format: *.net
  Vertices section: "*Vertices" <#vertices>
  vertex info: <index> <name> "ic" <fg color> "bc" <bg color>
  Edges section: "*Edges"
  edges: <index1> <index2> <weight> "w" <width> "l" <label>
  NOTE: gives weight instead of length. $length := e^{-weight}$ is applied here.

Incidence list format: *.graph
  first line: < #vertices> <#edges>
  on line $i + 1$, there are indices of vertices incident to i, separated by space.
  NOTE: may (and usually does) contain duplicate edges.

DIMACS Graph format: *.gr, *.col
  comments: "c" <comment until end of line>
  directory line: "p edge" < #vertices> < #edges>
  edge line: "e" <index1> <index2> [<length>]
  NOTE: default choice. In the opinion of author of this thesis, this format is the best.
  NOTE: if length is not present, assumed to be =1.

DIMACS modified format for Cluster Arrangement: *.ca
  contains also a graph in DIMACS format.
  directory line: "p ca" < #vertices> < #edges> < #clusters>
  edge lines
  cluster lines: "k" {indices of vertices belonging to k-th cluster}

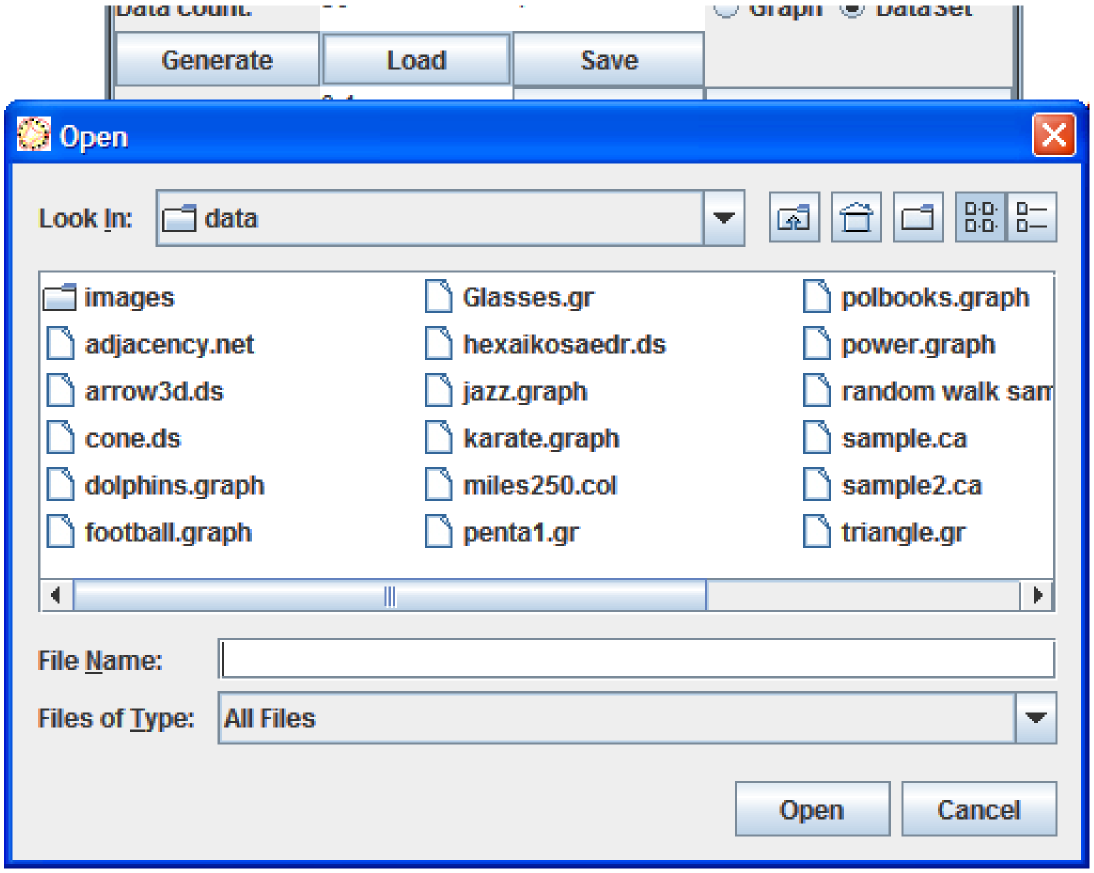
To load data from a file, press the Load button or select the Open option from the menu (Ctrl+O). See figure 5.3

5.6 Adjusting the Data

Before any real work begins, adjusting the data may be necessary; see figure 5.4.

5.6.1 Perturbation

If a Graph or a DataSet is acquired, it may come handy to be able to perturb the data (move Points/change the Edge length a bit). For that, there is the Perturb command button. With the perturb range, the maximal ratio the Edge length may be increased or decreased is controlled. If a DataSet is present, the coordinates of Points will be perturbed within the range value.



images
adjacency.net
arrow3d.ds
cone.ds
dolphins.graph
football.graph
Glasses.gr
hexaikosaedr.ds
jazz.graph
karate.graph
miles250.col
penta1.gr
polbooks.graph
power.graph
random walk sam
sample.ca
sample2.ca
triangle.gr

Figure 5.3: The loading file chooser triggered by clicking the Load button.

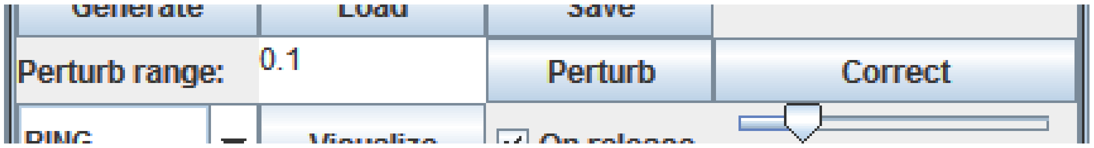


Figure 5.4: The data adjusting section of GUI.

5.6.2 Edge Length Correction

If a Graph is generated from a DataSet, its edge lengths are correct. Generated or Graph loaded from a file might need be *corrected*. This procedure adds an Edge, where it is missing and sets its distance to the length of the shortest path found in the Graph.

5.7 Analysis

Select a combination of algorithm, ranking, distance and energy from the corresponding combo boxes, put in the required number of clusters in the result and press the Analyze button.

To prematurely terminate a running analysis, push the Break button (red if any analysis is running).

After an analysis finishes, the result and usually other cluster arrangements are available to display. The selection can be made by clicking on an item in Alternatives Frame refalternatives or in the Alternatives Combo Box, which can be also browsed by arrows.

5.8 Visualisation

5.8.1 Graphics

There are two ways to visualise the data. If a DataSet is present, it can be projected to each pair of axis within its dimensionality. Therefore, the result visualisation is a scatter plot matrix [2]. For this option, select the value $DATA$ in the visualisation combo box. See figure 5.5.

If the DataSet is not present, we can always apply the RING visualisation as shown in figure 5.6. The vertices are printed in a circle ("ring"; ellipse if the frame is not a square), the edges are painted between them with their darkness and width respective to their relative shortness. To hide some of the (long thus least important) edges, use the visibility slider.

5.8.2 Log

To view text output from the running process, toggle the Log. Each log entry contains a time stamp.

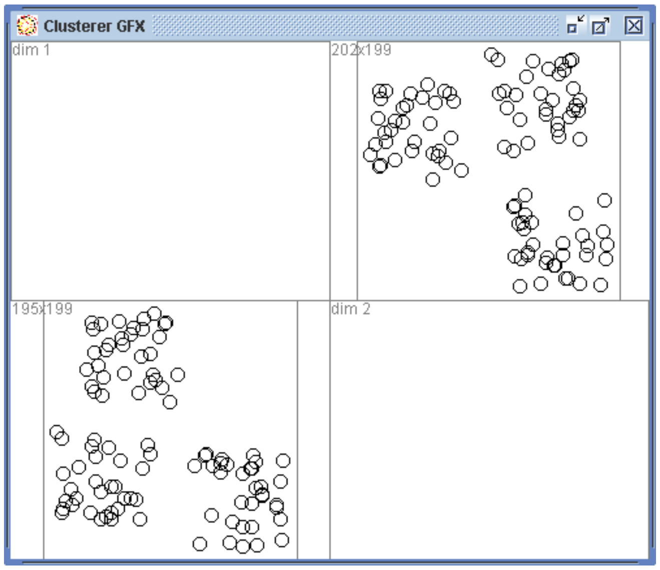


Figure 5.5: Sample DataSet painted in a data mode

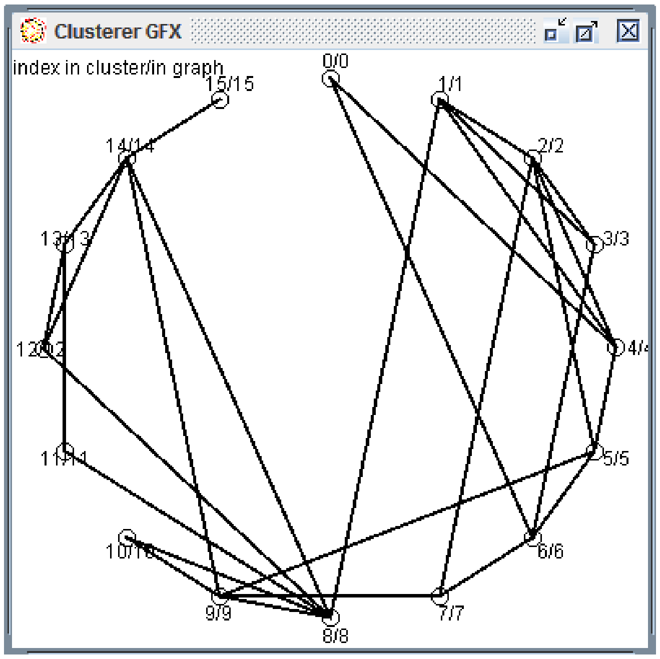


Figure 5.6: Sample graph painted in a ring

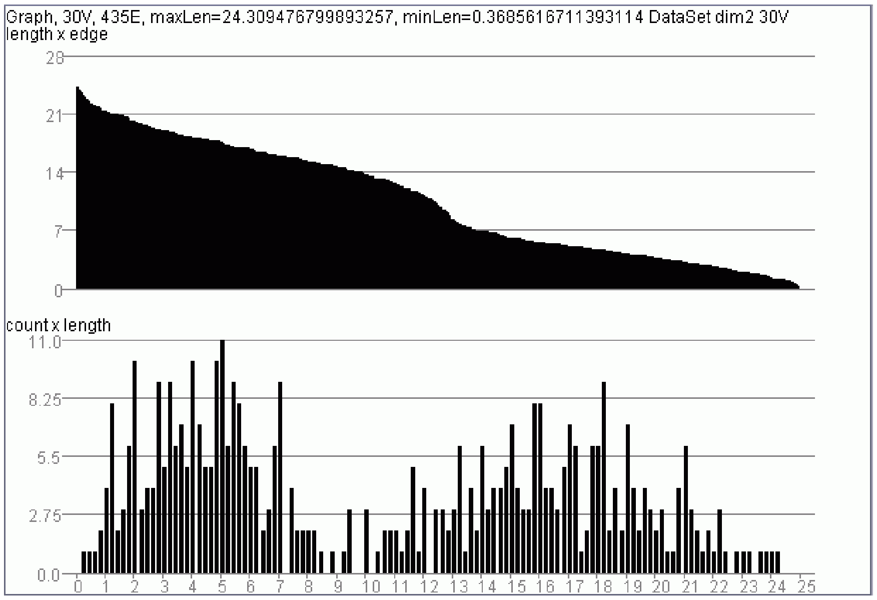


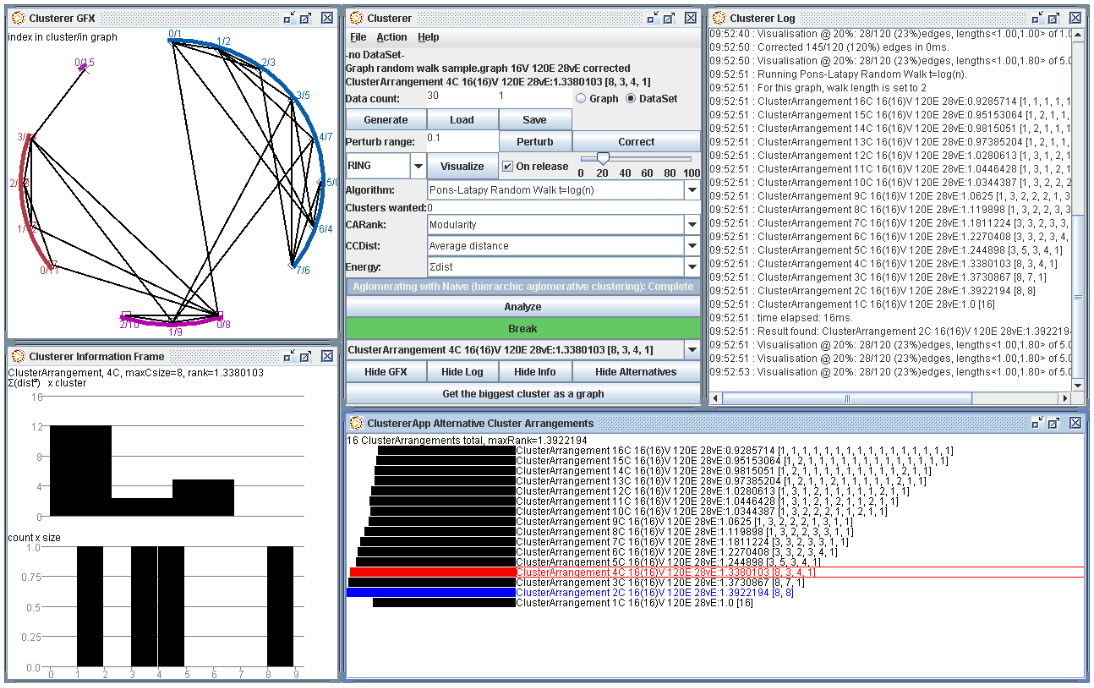
Figure 5.7: An example of Info Frame showing a generated graph with two distant clusters.

5.8.3 Info Frame

To show more information about the given Graph/ClusterArrangement, there is the Info Frame. For a Graph, it demonstrates the Edge length distribution. In the first diagram, there are (scaled) bars representing edge length sorted by their length (infinite edges are red and short). In the second diagram, there are amounts of edges within a length range. For a cluster arrangement, the first diagram shows the energies of clusters, the second shows the amounts of vertices with specified amounts of vertices. See an example in figure 5.7

5.8.4 Alternatives Frame

When an analysis finishes, there may be other interesting results to browse. To view them in the order of appearance, toggle the Alternatives Frame. The available Cluster Arrangements are presented in a text form, with a bar representing their relative ranking to the left. The result with optimal rank is painted in blue, the currently selected result is painted in red. The set of alternative results is cleared every time another analysis is run.



-no DataSet-
Graph random walk sample.graph 16V 120E 28vE corrected
ClusterArrangement 4C 16(16)V 120E 28vE:1.3380103 [8, 3, 4, 1]
Data count: 30 1 Graph [✓ DataSet]
Perturb range: 0.1
RING [✓ On release] 0 20 40 60 80 100
Algorithm: Pons-Latapy Random Walk t=log(n)
Clusters wanted: 0
CARank: Modularity
CCDist: Average distance
Energy: Σdist
Aglomerating with Naive (hierarchic aglomerative clustering): Complete
ClusterArrangement 4C 16(16)V 120E 28vE:1.3380103 [8, 3, 4, 1]

```
09:52:40 : Visualisation @ 20%: 28/120 (23%)edges, lengths<1.00,1.00> of 1.0
09:52:50 : Corrected 145/120 (120%) edges in 0ms.
09:52:50 : Visualisation @ 20%: 28/120 (23%)edges, lengths<1.00,1.80> of 5.0
09:52:51 : Running Pons-Latapy Random Walk t=log(n).
09:52:51 : For this graph, walk length is set to 2
09:52:51 : ClusterArrangement 16C 16(16)V 120E 28vE:0.9285714 [1, 1, 1, 1, 1
09:52:51 : ClusterArrangement 15C 16(16)V 120E 28vE:0.95153064 [1, 2, 1, 1,
09:52:51 : ClusterArrangement 14C 16(16)V 120E 28vE:0.9815051 [1, 2, 1, 1, 1
09:52:51 : ClusterArrangement 13C 16(16)V 120E 28vE:0.97385204 [1, 2, 1, 1,
09:52:51 : ClusterArrangement 12C 16(16)V 120E 28vE:1.0280613 [1, 3, 1, 2, 1
09:52:51 : ClusterArrangement 11C 16(16)V 120E 28vE:1.0446428 [1, 3, 1, 2, 1
09:52:51 : ClusterArrangement 10C 16(16)V 120E 28vE:1.0344387 [1, 3, 2, 2, 2
09:52:51 : ClusterArrangement 9C 16(16)V 120E 28vE:1.0625 [1, 3, 2, 2, 2, 1, 3
09:52:51 : ClusterArrangement 8C 16(16)V 120E 28vE:1.119898 [1, 3, 2, 2, 3, 3
09:52:51 : ClusterArrangement 7C 16(16)V 120E 28vE:1.1811224 [3, 3, 2, 3, 3,
09:52:51 : ClusterArrangement 6C 16(16)V 120E 28vE:1.2270408 [3, 3, 2, 3, 4,
09:52:51 : ClusterArrangement 5C 16(16)V 120E 28vE:1.244898 [3, 5, 3, 4, 1]
09:52:51 : ClusterArrangement 4C 16(16)V 120E 28vE:1.3380103 [8, 3, 4, 1]
09:52:51 : ClusterArrangement 3C 16(16)V 120E 28vE:1.3730867 [8, 7, 1]
09:52:51 : ClusterArrangement 2C 16(16)V 120E 28vE:1.3922194 [8, 8]
09:52:51 : ClusterArrangement 1C 16(16)V 120E 28vE:1.0 [16]
09:52:51 : time elapsed: 16ms.
09:52:51 : Result found: ClusterArrangement 2C 16(16)V 120E 28vE:1.392219
09:52:51 : Visualisation @ 20%: 28/120 (23%)edges, lengths<1.00,1.80> of 5.0
09:52:51 : Visualisation @ 20%: 28/120 (23%)edges, lengths<1.00,1.80> of 5.0
09:52:53 : Visualisation @ 20%: 28/120 (23%)edges, lengths<1.00,1.80> of 5.0
```

```
16 ClusterArrangements total, maxRank=1.3922194
ClusterArrangement 16C 16(16)V 120E 28vE:0.9285714 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
ClusterArrangement 15C 16(16)V 120E 28vE:0.95153064 [1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
ClusterArrangement 14C 16(16)V 120E 28vE:0.9815051 [1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1]
ClusterArrangement 13C 16(16)V 120E 28vE:0.97385204 [1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1]
ClusterArrangement 12C 16(16)V 120E 28vE:1.0280613 [1, 3, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1]
ClusterArrangement 11C 16(16)V 120E 28vE:1.0446428 [1, 3, 1, 2, 1, 2, 1, 1, 2, 1, 1]
ClusterArrangement 10C 16(16)V 120E 28vE:1.0344387 [1, 3, 2, 2, 2, 1, 1, 2, 1, 1]
ClusterArrangement 9C 16(16)V 120E 28vE:1.0625 [1, 3, 2, 2, 2, 1, 3, 1, 1]
ClusterArrangement 8C 16(16)V 120E 28vE:1.119898 [1, 3, 2, 2, 3, 3, 1, 1]
ClusterArrangement 7C 16(16)V 120E 28vE:1.1811224 [3, 3, 2, 3, 3, 1, 1]
ClusterArrangement 6C 16(16)V 120E 28vE:1.2270408 [3, 3, 2, 3, 4, 1]
ClusterArrangement 5C 16(16)V 120E 28vE:1.244898 [3, 5, 3, 4, 1]
ClusterArrangement 4C 16(16)V 120E 28vE:1.3380103 [8, 3, 4, 1]
ClusterArrangement 3C 16(16)V 120E 28vE:1.3730867 [8, 7, 1]
ClusterArrangement 2C 16(16)V 120E 28vE:1.3922194 [8, 8]
ClusterArrangement 1C 16(16)V 120E 28vE:1.0 [16]
```

Figure 5.8: Overview of fully shown GUI with a loaded graph and a complete analysis.

5.9 Data and Results Storage

Graphs and Cluster Arrangements can be saved into a file. Graphs are saved in DIMACS format with the `.gr` extension. ClusterArrangements can be saved using the extended DIMACS format into a `.ca` file. Note that a ClusterArrangement contains a Graph and so it does in the file.

To save the data, press the Save button or select the Save item from the menu (Ctrl+S). A saving dialog similar to loading (shown in figure 5.3) will appear.

Chapter 6

Algorithm Comparison

Algorithms were run on sample data, mostly downloaded from [21].

random walk sample - A sample graph used in [6]. 16 vertices, 28 edges.

karate - Zachary's Karate club. A network of friendships within 34 members of a sports team. 78 edges.

dolphins - Network of frequent associations between 62 dolphins from Doubtful Sound, New Zealand. 159 edges.

polbooks - A network of books about US politics, published around 2004 and sold by Amazon.com. Edges represent 441 frequent copurchasing of 104 books.

football - American College football: network of games between 115 Division IA colleges during regular season Fall 2000. 613 edges.

jazz - Jazz musicians network. List of 2742 edges of the network of 198 Jazz musicians.

power - Power grid. Network representing the topology of the Western States Power Grid of the US. 4941 vertices, 6594 edges.

Table 6.1: First three algorithms.

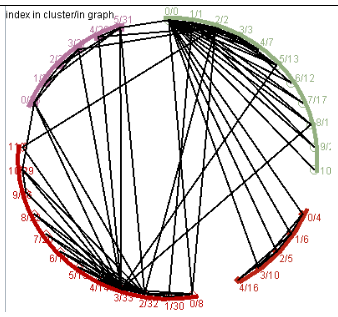
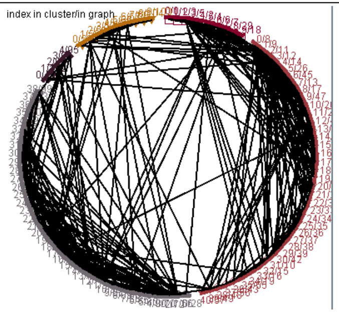
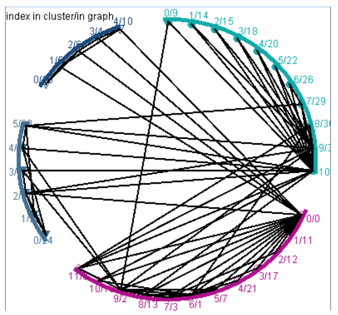
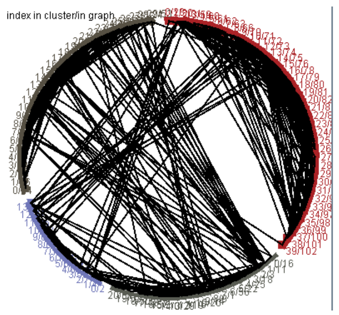
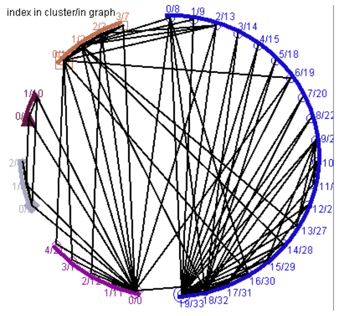
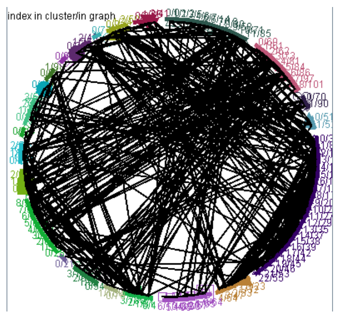
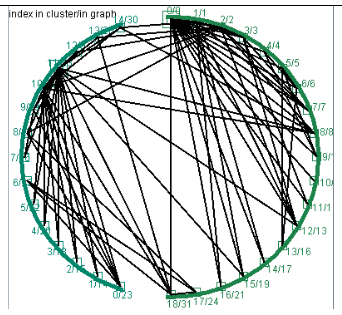
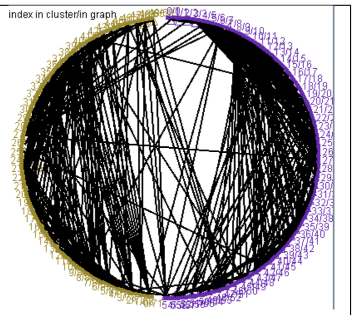
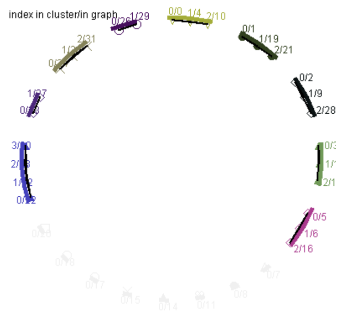
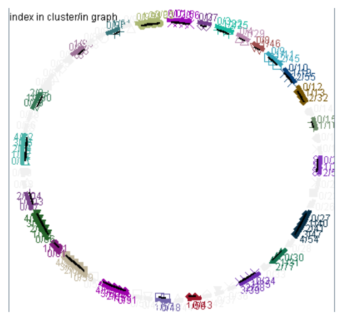
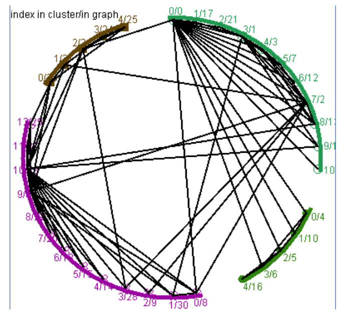
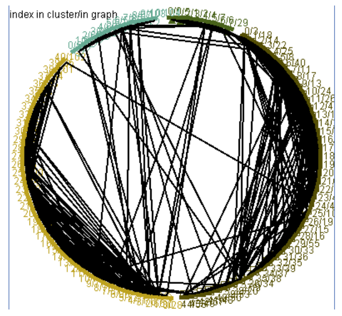
	karate	polbooks
AGLO		
DIVI		
K-Means		

Table 6.2: Second three algorithms.

	karate	polbooks
Wishart		
Girvan-Newman		
Pons-Latapy		

Tables 6.1 and 6.1 show results with highest modularity of algorithms run on graphs **karate** and **polbooks**. Note that it is hardly possible to objectively evaluate the results when there is no exact definition and no single correct result.

On data **karate** three algorithms give similar results with four clusters. These results seem natural and correct to the author of this thesis. Wishart algorithm returned only two clusters but they also appear acceptable. Other results, chosen by modulariy rank, do not seem to be useful.

On data **polbooks**, the results were more different. Results algorithms with previously good results seem to make sense in this case too. Since this graph is larger than **karate**, it is hard to imagine the true structure and find the best result.

Results of K-Means algorithm do not seem helpful and the algorithm seems not to be suitable for community detection. Girvan-newman algorithm turned out to be incompatible with modularity ranking function due to trash clusters. On the other hand, more satisfying results were found while browsing alternatives from this algorithm.

Results for other combinations of data, algorithms and other parameters are not included because they would be too many. Fortunately, reader is given the opportunity to try his own combination of settings.

Chapter 7

Conclusion and Further Work

In the Clusterer application, six different algorithms were implemented, all in several versions. A parser for four file formats of data, one for output. A mechanism for dual representation of graph, tools for generation, perturbation and correction of data, a system of results evaluation, visualisation and browsing.

Implemented algorithms were run on several data of various types from different sources.

Further work on this project may include implementation of more algorithms and ranking functions, a tool for comparison of two or more results with matching percentage, a tool for generating unweighted graphs, a force based visualisation and/or clustering algorithm, DataSet reconstruction for graphs to enable $DATA$ visualisation.

Bibliography

[1] Lukasová, A., Šarmanová, J.: *Metody shlukové analýzy.* SNTL, Praha, 1985.

[2] Hand, D.J., Mannila H., Smyth P.: *Principles of Data Mining.* MIT, 2001.

[3] Hebák P., Hustopecký J.: *Vícerozměrné statistické metody s aplikacemi.* SNTL/ALFA, Praha, 1987.

[4] Girvan, M., Newman, M.E.: *Community structure in social and biological networks.* PNAS, 2002.

[5] Radicchi F., Castellano C., Cecconi F., Loreto V., Parisi D.: *Defining and identifying communities in networks.* PNAS, 2002.

[6] Pons P., Latapy M.: *Computing Communities in Large Networks Using Random Walks.* LIAFA, 2004.

[7] Girvan, M., Newman, M.E. Phys. Rev. E 67, 026126 (2003).

[8] MacNaughton-Smith, P.: *Some Statistical and Other Numerical Techniques for Classifying Individuals.* London, Home Office Ressearch Research Unit Report, 1965.

[9] Sokal, R.R., Michener, C.D.: *A Statistical Method for Evaluating Systematic Relationships.* Univ. Kansas Sci. Bull., 1958, p. 1409-1438.

[10] Gower, J.C.: *A Comparison of some Methods of Cluster Analysis.* Biometrics, 1967, p.623-637.

[11] Sneath, P.H.A.: *Evaluation of Clustering Methods.* In: Numerical Taxonomy. Red. A.J.Cole London and New York, Academic Press, 1969

[12] Sokal, R.R.,Sneath, P.H.A.: *Principles of Numerical Taxonomy.* San Francisco and London,W.H.Freeman and Comp. 1963

[13] Forgy,E.W.: *Cluster Analysis of Multivariate Data: Efficiency Versus Interpretability of Classifications.* Biometric Soc. Meetings, Riverside, California (Abstract in Biometrics 21, No. 3, 768), 1965.

[14] Jancey, R.C.: *Multidimensional group analysis.* Australian J. Botany, 14:p. 127- 130, 1966.

[15] Ward, J.H.: *Hierarchical grouping to optimize an objective function.* Journal of the American Statistical Association **58** (1963) 236–244

[16] Wishart, D.: *Mode analysis.* In: Numerical Taxonomy (ed. A. J. Cole). Academic Press, London and New York. 1969. p. 282-308.

[17] `http://en.wikipedia.org/wiki/Community_structure`

[18] `http://en.wikipedia.org/wiki/Graph_(mathematics)`

[19] `http://www.eclipse.org/`

[20] `http://www.oracle.com/technetwork/java/index.html`

[21] `http://www.cc.gatech.edu/dimacs10/archive/clustering.shtml`

[22] `http://dimacs.rutgers.edu/`

Appendix A

CD Contents

- Electronic version of this text
- Eclipse Java project containing all the source codes and a build file
- Runnable JAR file with the application
- Testing data
- Generated Javadoc