ClusterVisualizationKit Manual

University of Leipzig

Contributors:
R. Speck
A.-C. Ngonga Ngomo

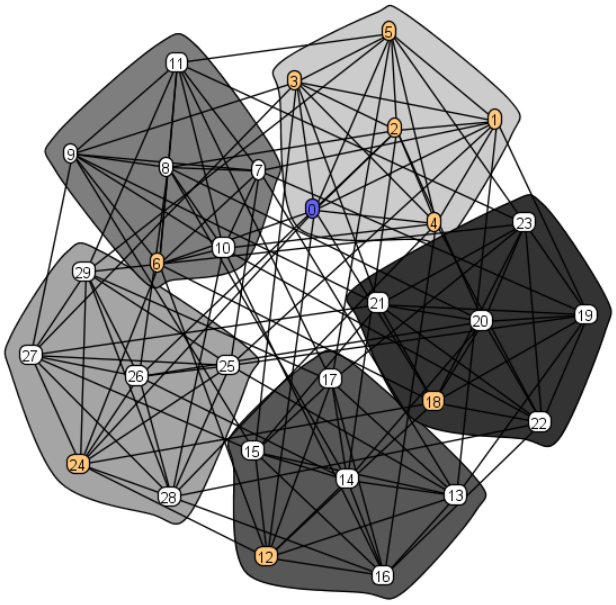

Figure 1: A clustered 6-5-partite clique.

September 13, 2010

Contents

1 Introduction

This manual is intended as a guide for using the Cluster Visualization Kit (CVK), a tool and framework for graph clustering. It presents the functionality of CVK in its current version. In addition, it provides insight in the architecture of the underlying framework, so as to enable any user to extend it as required for his purposes.

1.1 Purpose

Our main aim while developing the CVK was to provide a lightweight framework that allows rapid integration and testing of novel graph clustering algorithms. Roughly speaking, the CVK in its current version allows to (1) visualize input graphs stored in several formats, (2) cluster the input graph using any of the algorithms integrated, (3) integrate novel algorithms very efficiently as required (see Section 7.1.1), (4) explore the resulting clustering and (5) store the resulting clustering for further processing.

The toolkit is intended for both beginners and experts: the CVK provides an easy-to-use graphical interface for users that not yet familiar with graph clustering. In addition, CVK's modular architecture allows the rapid integration of new clustering algorithms. Ergo, experts are provided with means to (1) efficiently develop, (2) visualize the results and (3) evaluate the quality of (new) clustering algorithms for graphs. The user interface was designed to be as intuitive as possible, so as to allow the easy manipulation of nodes and clusters during the exploration of the input graph and resulting clustering. In addition, the tool implements several metrics that provide a numeric impression of the quality of the clustering achieved by the algorithm applied.

The visualization component is based on the prefuse visualization toolkit[1], the interesting part of graphs generated by the application (random graph models) based on the jung framework[2] and the Lucene search engine[3] supports a fast search over nodes. The CVK implements the clustering algorithms BorderFlow, k-Nearest Neighbours and MCL in its current version. Further algorithms will be integrated soon. CVK is open-source and available under the BSD License.

1.2 Requirements

The only requirement to run the application is an installed JRE[4] version 1.6 upwards.

[1] `http://prefuse.org`
[2] `http://jung.sourceforge.net`
[3] See `http://lucene.apache.org/java/1_4_3` for all options.
[4] `http://www.java.com/de/download/manual.jsp`

2 Using the CVK

2.1 Launching

To start the application, double-click the ClusterVisualizationKit.jar file or type the following in your terminal: *java -jar ClusterVisualizationKit.jar*. Should you get *Out of Memory errors*, increase the maximum heap size for your java virtual machine (JVM) via the *-Xmx* option[5].

After you have started the application, you should see the main panel as shown in figure below.

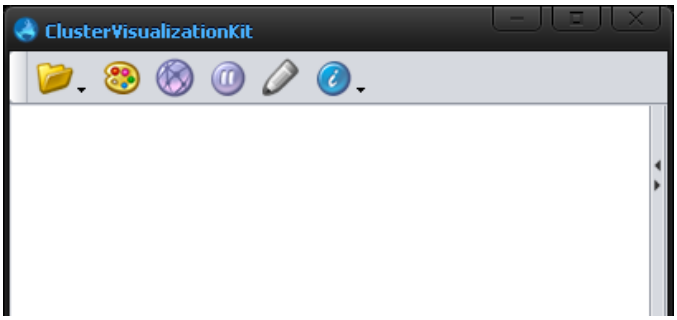

Figure 2: Welcome screen

2.2 Loading data

The first step in using the application is to read data. Two types of data can be loaded into the CVK: user-generated data (i.e., a file) or automatically generated data for tests purposes.

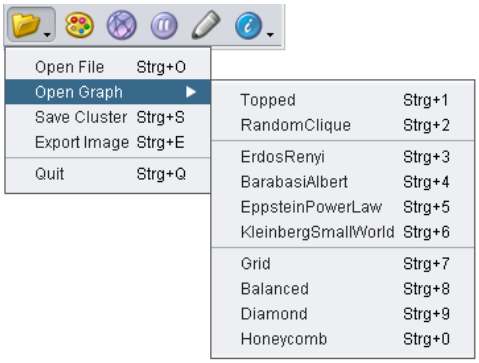

Figure 3: Toolbar

[5] `http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html`

2.2.1 Loading User-Generated Data

Currently, the CVK supports delimiter separated text files and the XML-format GraphML as input format for user-generated data. To import such data, use the toolbar or the corresponding keyboard shortcut as displayed in figure 3. After you have selected "Open File" from toolbar menu, you will see the dialog shown in 4. By using this dialog, you can choose the type of files to import, i.e., whether a delimiter separated or a GraphML file should be imported. The default "Open File" dialog is set to import delimiter separated data files as they are more commonly used than GraphML. The file extension is used to determine the file delimiter (3.2).

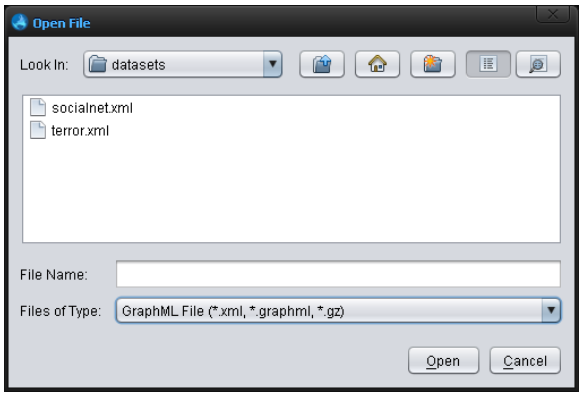

Figure 4: Open File Dialog.

2.2.2 Loading automatically generated data

This menu item allows you to load automatically generated data instead of user-generated data. The basic intention behind this feature was to allow users to view the results of algorithms or small graph without having to encode a graph manually.
After selecting "Open Graph", you will see the submenu as shown in figure 3. This submenu allows you to choose from various types of graphs to load, including the a topped tetrahedron,random n-k partite cliques (i.e., graphs that contains k cliques of n nodes, with $3 \leq n \leq 15$, $2 \leq k \leq 14$ and $k \leq n$), graphs generated by using the binomial model of Erdős and Rényi (n=100, p = 0.02) and some others that comes from the prefuse library.

2.3 Configuring

After you have loaded your data into CVK, you may want to configure the force layout to display your graph in a fashion suitable to your taste. To achieve this, use the sliders (Figure 5 - left) in the three boxes NBodyForce, DragForce and SpringForce. You can also change the Connectivity Filter,

this entry controls the maximal distance between nodes that are displayed and the node that is currently selected. Obviously, pushing this value to more than 3 when a large graph is loaded might lead to longer processing time before a stable graph is displayed. To stop the animation, please use the "Play/Pause" button on the menu bar.

One of the drawback of displaying nodes based on the surroundings of the node that is currently selected is that nodes that are at an infinite distance (i.e., not connected) of the currently selected node never get displayed. To guarantee that the user retains an overview of the graph, the interface provide an overview window below the sliders. To ensure that all nodes are displayed, we added the grid button that allows to display all nodes are once. Another way to select a node and thus to display the surroundings of the node is to use the clickable node table (figure 5 - right).

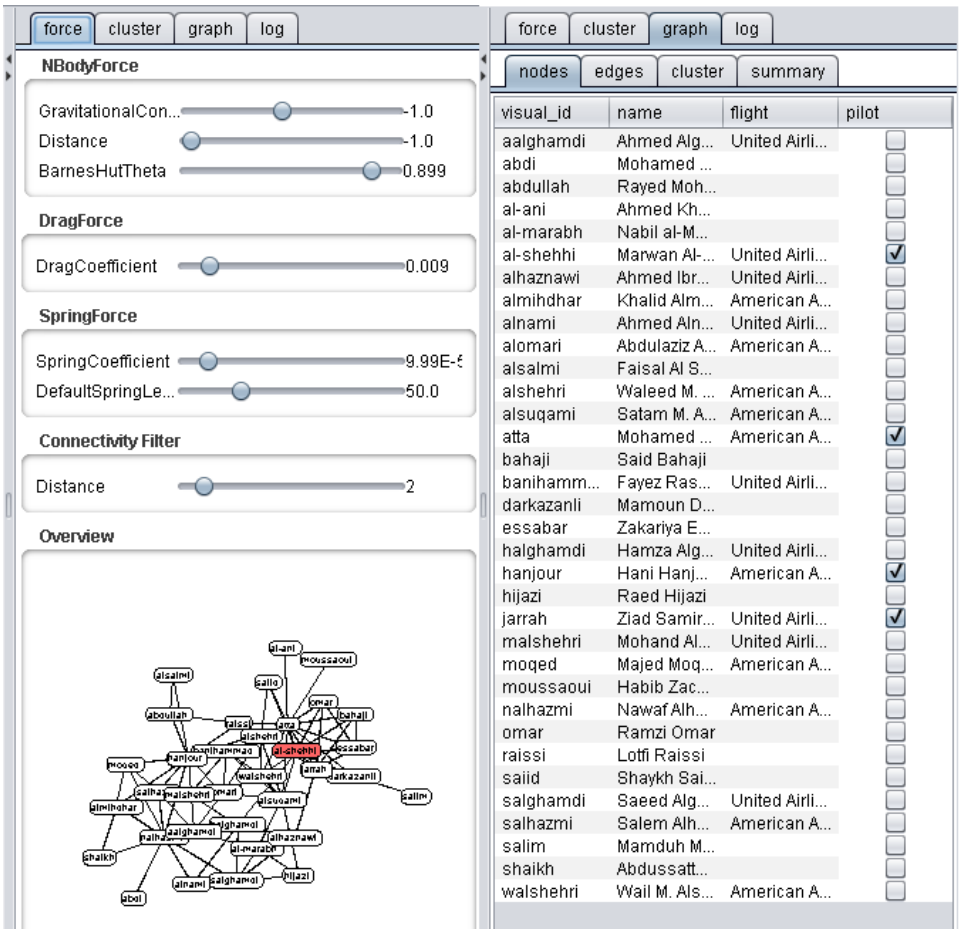

Figure 5: Force configurations (left) and clickable node table (right).

2.4 Clustering

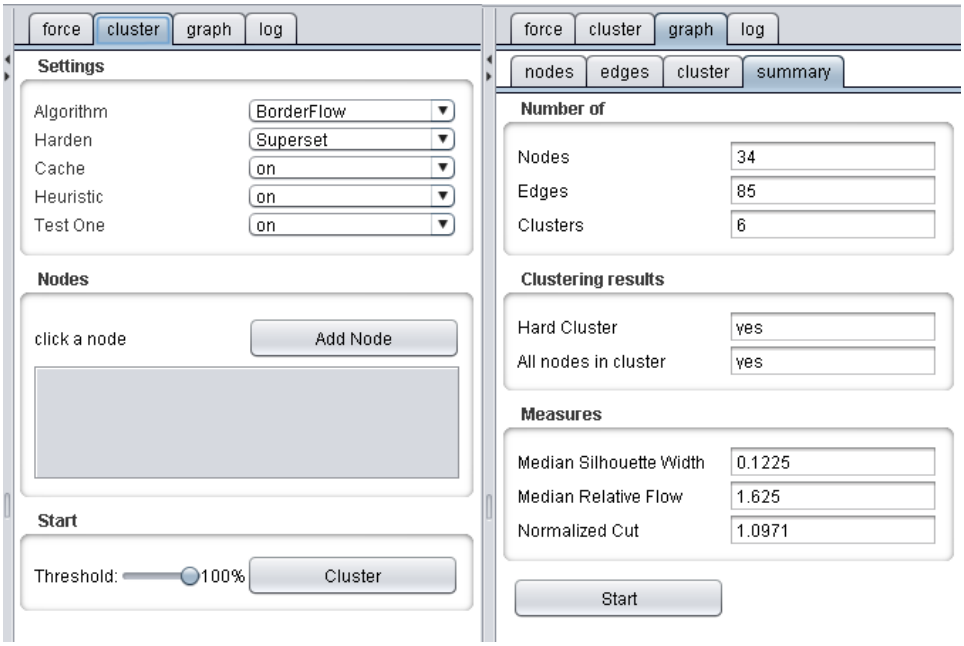

Figure 6: Clustering configurations (left) and graph statistic (right).

The main aim of the CVK is to enable users to test clustering approaches. To choose a clustering algorithm to process the data you loaded, go to the cluster tab (figure 6 - left), pick one of the algorithms[6] from the "Settings" box and select/set the parameters you need for your clustering process. The CVK was designed in such a way that each algorithm has its own clustering configuration. For example, the BorderFlow and k-Nearest Neighbours algorithms allow to select the nodes that are used as seed for clustering; they also provide a threshold slider, which sets the percentage of the maximal connectivity that a node can maximally have to be used as seed (obviously, moving the slider to 0 causes all nodes to be used as seeds). After having configured your clustering algorithm, press the "Cluster" button to start the clustering process. Depending on the size of the input data and the algorithm, the clustering might take a while. Please be patient.

After the completion of the clustering, the CVK allows you to navigate through the clusters by using mouse actions on the main panel. By using the color button, you can turn the display of the clusters to greyscale or color. Finally, the "Save clusters" menu item under the folder icon allows to save the results of the clustering for further processing or evaluation.

[6]See description in section 4

3 Supported Data Formats

The CVK support various file formats, which are explained below. Furthermore, it is possible to open such graphs directly from the menu bar in the application as described in section 2.2.2. The use of edge weights is possible in all file types. If no weights are set, the weight value is set to 1. Because most clustering algorithms are defined in such a way that they do not support edges that begin and end at the same node, edges with the same node as source and target are ignored when reading the input data. In addition, nodes without edges are ignored.

3.1 GraphML/XML

GraphML is a XML format supporting graph structure and typed data schemas for both nodes and edges. For more information about this format, see the GraphML home page[7]. The only restriction of the CVK when reading GraphML is that it does not support the mix of weighted and unweighted edges within one graph.

3.2 CSV/SSV/Tab/Txt

The CVK supports an edge list file format in which each line is an edge. In a line the first term is a source node, second term is a target node and the last term is a weight. We use delimiter separated values, each term in a line has to be separated dependents on the used file format extension as shown in the following table.

File Extension	Separator
csv	comma
ssv	space
tab/txt	tabulator

[7] `http://graphml.graphdrawing.org/`

4 Supported algorithms

The CVK is intended to enable beginners to utilize exisiting algorithms and developers to test their new algorithms. The current version of the CVK implements the clustering algorithms described in the following.

4.1 BorderFlow

BorderFlow is a general-purpose graph clustering algorithm [2]. It uses solely local information for clustering and achieves a soft clustering of the input graph. The standard definition of clusters is that they have a maximal intra-cluster density and inter-cluster sparseness. When considering a graph as the description of a flow system, this definition of a cluster implies that a cluster X can be understood as a set of nodes such that the flow within X is maximal while the flow from X to the outside is minimal. The idea behind BorderFlow is to maximize the flow from the border of each cluster to its inner nodes (i.e., the nodes within the cluster) while minimizing the flow from the cluster to the nodes outside of the cluster.

4.2 k-Nearest Neighbours

In pattern recognition, the k-nearest neighbours algorithm (k-NN) was orginally a method for classifying objects based on closest training examples in the feature space. k-NN is a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of its nearest neighbor. k-NN can be used for seed-based clustering simply by stating that each of the $k-1$ nearest neighbors of an input seed are the elements of a cluster of size k (seed + neighbors). Clusters that happen to containe the same elements are merged to one cluster.

4.3 MCL

The basic idea underlying the MCL algorithm [5] is that dense regions (i.e., clusters) in sparse graphs correspond with regions in which the number of k-length paths is relatively large. Thus, random walks of length k have a higher probability to begin and end in the same dense region than for other paths. The algorithm starts uses a stochastic matrix M to represent the input graph. Then, it alternates two operations (expansion and inflation) to compute the set of transition probabilities until M does not change substantially. The result is a complete and does not contain overlapping clusters.

5 Measurements

Several means for assessing the quality of a clustering have been defined in the past. In this section, we describe those measurements that are computed after the completion of each clustering (Figure 6-right). These metrics can be easily extended as required by the user. In the following, we will assume that $G = (V, E, \omega)$ is a weighted directed graph with a set of vertices V, a set of edges E and a weighing function ω, which assigns a positive weight to each edge $e \in E$. V can be partitioned into k subsets $X_1, ..., X_k$. Furthermore, we define $\Omega(X_1, X_2) = \sum_{x_1 \in X_1, x_2 \in X_2} \omega(x_1 x_2)$ as the function that assigns the total weight of the edges from a subset $X_1 \subset V$ to a subset $X_2 \subset V$. The CVK toolkit implements the median of average silhouette width, the median of the relative flow and the normalized cut measures.

5.1 Median of average silhouette width

The silhouette measures of a set is given by the following equation[3]

$$s(x) = \frac{b(x) - a(x)}{max\{a(x), b(x)\}}$$

where $a(x)$ is the average similarity between vertex $x \in X$ and all other vertices in subset X, and $b(x)$ is the average similarity between vertex $x \in X$ and all other vertices in the neighbour subsets to X.
The average of $s(x)$ for all vertices x in a subset X, is called the average silhouette width of X. The average $\overline{s}$ of $s(x)$ for all vertices $x \in V$, is called the average silhouette width for the entire data set.
"The Silhouette Coefficient(SC) is a measure where the maximum is taken over all k for which the silhouettes can be constructed[...]"[1].

$$SC = max_k \overline{s}(k)$$

Values between 0.7 and 1.0 indicate clustering results with excellent separation between clusters, for the range from 0.5 to 0.7 one finds that vertices are clearly assigned to cluster centers. Values from 0.25 to 0.5 indicate that cluster centers can be found, though there is considerable noise, below a value of 0.25 it becomes practically impossible to find significant cluster centers and to definitely assign the majority of data points.

We implemented the median of the average silhouette width s_{med} to provide users of the CVK with a quick evaluation of their clustering with respect to the the partitioning of the input data:

$$s_{med} = \begin{cases} s_{\frac{n+1}{2}} & \text{if n is odd} \\ \frac{1}{2}(s_{\frac{n}{2}} + s_{\frac{n}{2}+1}) & \text{if n is even} \end{cases}$$

with $s_j \in \{s_1, ...s_n\}$ a sorted list of the average silhouette width of all X_i, $i \in \{1, ..., k\}$.

5.2 Average relative flow

As another mean to asses the quality of a clustering, we implemented the median of the relative flow[2], which is defined as:

$$F(X) = \frac{\Omega(b(X), X)}{\Omega(b(X), n(X))}$$

where $b(X)$ is the set of border vertex of X and $n(X)$ is the set of direct neighbors of X.

5.3 Normalized Cut

Any subset of vertices $X \subset V$ creates a cut, which is a partition of V into two disjoint subsets X and $V \setminus X$. The size of a cut X of graph G is defined as $cut(X) = \omega(X, V \setminus X)$ and measures the weight of edges that have to be eliminated in order to obtain the two components X and $V \setminus X$. The normalized cut[4] normalizes the cut measurement with the total sum of all vertex degrees over all vertices in a subset X and is defined as

$$ncut(X_1, ..., X_k) = \frac{1}{2} \sum_{i=1}^{k} \Omega(X_i, V \setminus X_i)/vol(X_i)$$

where $vol(X) = \sum_{x \in X} d_x$ is the volume of subset X and $d_x = \sum_y \omega(xy)$ is the degree of vertex $x \in X$.

6 Hardening

Many clustering algorithms return overlapping (i.e., BorderFlow) clusters or clusterings that do not cover all input data (i.e., DBScan, RNSC). The CVK provides means to ensure that each clustering do not contain overlapping clusters and are complete, both characteristics being desirable for several types of applications. Currently, the CVK implements two different hardening strategies: MaxQuality and SuperSet. Both require the following input data:

1. a graph $G = (V, E, \omega)$
2. a set $\Psi = \{X_1, ..., X_k\}$ of k subsets of V with
3. $\forall i \in \{1, ..., k\} X_i \neq \emptyset$.

A hardening process consists of transforming the set Ψ of k subsets to a set Ψ' of k' subsets such that $\forall i, j \in \{1, ..., k'\} X'_i \cap X'_j = \emptyset$.

6.1 Hardening with MaxQuality strategy

The MaxQuality strategy applies a function f, the quality function, that assigns each subset X a value $f(X) \geq 0$.

1. if $\exists i \in \{1, ..., k\} | X_i | = |V|$ then remove X_i from Ψ
2. calculate $f(X_i)$ for all $X_i \in \Psi$
3. sort all X_i by descending $f(X_i)$
4. put in a set S_{max} all X_i with maximal $f(X_i)$
5. if S_{max} contains h subsets $X^{'}$such that $\bigcap_{i \in \{1,...,h\}} X_i^{'} = \emptyset$ then put all these h subsets to $\Psi^{'}$, remove these h subsets from Ψ and go to step 7
6. else while $S_{max} \neq \emptyset$ do
   (a) find subsets in S_{max} they share any vertex, put these in $S_{max}^{'}$ and remove them from S_{max} and from Ψ
   (b) take all shared vertices v in $\bigcap_{X \in S_{max}^{'}} X = R$ and remove them from all subsets $X \in S_{max}^{'}$ such that $\bigcap_{X' \in S_{max}^{'}} X^{'} = \emptyset$
      i. if $|R| = |\bigcup_{X' \in S_{max}^{'}} X^{'}|$ then put R to $\Psi^{'}$
      ii. else assign each vertex $v \in R$ to a subset $X^{'} \in S_{max}^{'}$ such that $f(X')$ is maximal and put these $X^{'}$ to $\Psi^{'}$
   (c) set $S_{max}^{'} = \emptyset$
7. remove all vertices $v \in V^{'} = \bigcup_{X' \in \Psi'} X^{'}$ from subsets in Ψ
8. remove empty sets in Ψ
9. if $\Psi \neq \emptyset$ go to step 2
10. assign each vertex $v \in V \setminus V^{'}$ to a subset $X^{'} \in \Psi^{'}$ so that $f(X^{'})$ is maximal.

6.1.1 MaxQuality with relative flow function

The implementation uses relative flow for the quality function f as described in section 5 and considers already used vertices in $\Psi^{'}$, that means, in step 2 all $v \in \Psi^{'}$ were ignored.

6.1.2 MaxQuality with silhouette function

The implementation uses silhouette for the quality function f as described in section 5 but only in step 2. In step 6.b.ii and 10 relative flow is used also.

6.2 Hardening with SuperSet strategy

The SuperSet strategy works as follows:

1. Discard all subsets X_i such that $\exists X_j : X_j \subset X_i$.
2. Order all remaining X_j into a list $L = \{\lambda_1, ..., \lambda_m\}$ in descending order with respect to the number of seeds that led to their computation. Formally, let $\mu(\lambda_i) \subseteq V$ be the set of seeds of a subset λ_i. Then
$$i < j \Rightarrow |\mu(\lambda_i)| \geq |\mu(\lambda_j)|.$$
3. Let k be the smallest index such that the union of all λ_i with $i \leq k$ equals V:
$$\bigcup_{i=1}^{k} \lambda_i = V \wedge \forall j < k : \bigcup_{i=1}^{j} \lambda_i \subset V.$$
Discard all $\lambda_z \in L$ with $z > k$.
4. Re-assign each v' to the subset(s) X' such that
$$X' = \underset{X \in \{\lambda_1, ..., \lambda_k\}}{\arg\max} \frac{\sum_{v \in X} w(v', v)}{|X|}.$$
5. Return the new subsets.

7 Developers section

This section goes beyond presenting the features of the CVK and shows how to extend the framework. It is intended for advanced users that aim at integrating their own algorithm or measures in the suite. The developers section is divided into four subsections. The first given an introduction to the packages and describes how to extend the CVK by integrating own algorithms. Then the following sub sections lists the dependencies, bugs and change log.

7.1 Packages

The CVK contains two main packages. The *bf* package, which contains the implementations of the BorderFlow and the kNN algorithms and the *cvk* package, which contains the core of the cvk framework. All details of the interfaces, methods and classes implemented in the CVK can be found in the Javadoc of the project[8]. The best entry point to understand the way the CVK functions is the cvk.gui.Main class.

7.1.1 The cvk package

This package contains the CVK framework per se and is surely the more interesting package for developers. Figure 7 gives an overview of all sub-packages contained in the CVK package, including with a short description of their functionality.

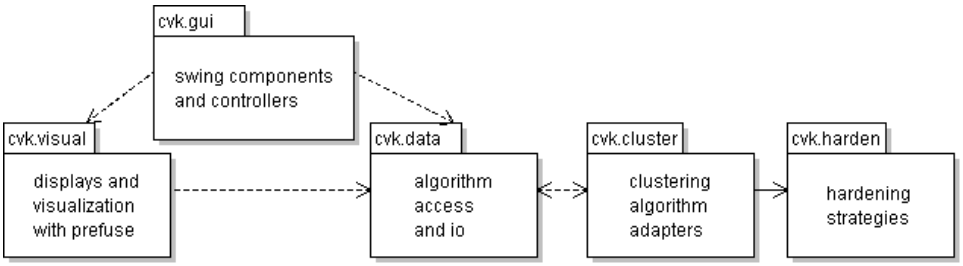

Figure 7: Package cvk.*.

cvk.data

This package provides classes to read or generate data. The resulting graph is stored into a prefuse.data.Graph instance in a Model instance which has a instance of cvk.cluster.ClusterContext to get access to a current used instance of cvk.cluster.ClusteringAlgorithm class (compare figure 8).

[8] `http://borderflow.sourceforge.net/doc/index.html`

The Model class is the only class that needs to be accessed by external classes to read input data by means of the integrated EdgeListGraphReader or GraphMLReaderMod, to generate graphs by using the GraphGenerator and to assign tasks to the currently chosen cvk.cluster.ClusteringAlgorithm instance.

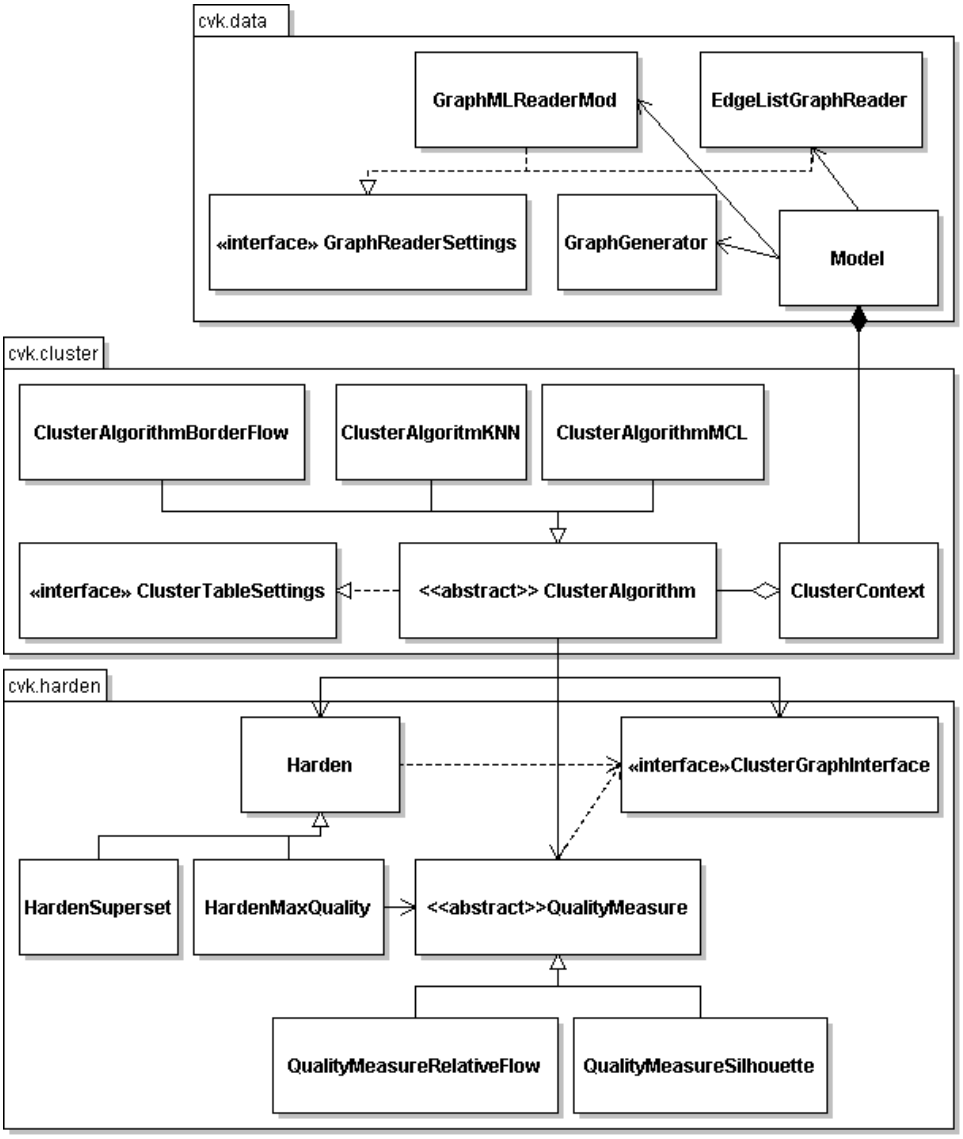

Figure 8: Package cvk.data, cvk.cluster and cvk.harden.

cvk.cluster

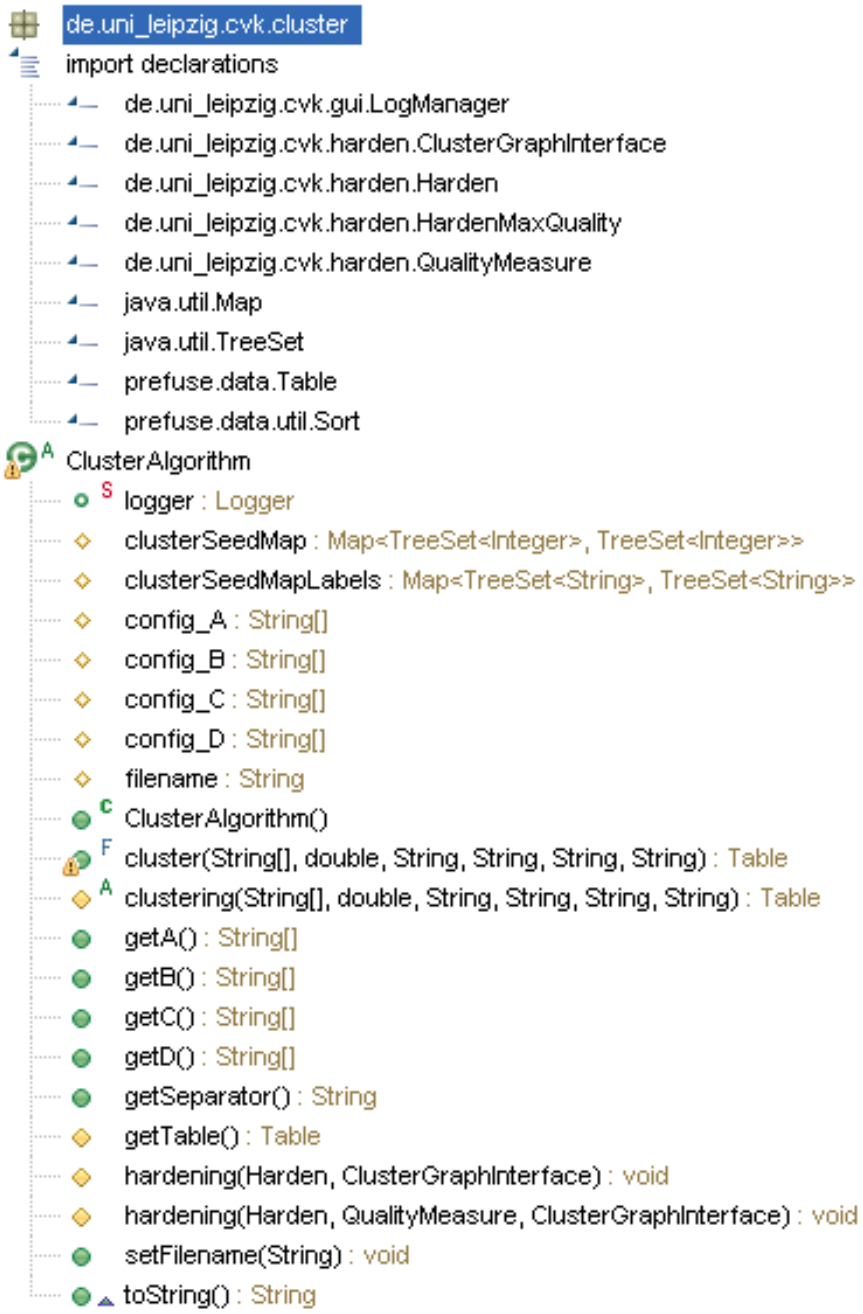

Figure 9: Abstract ClusterAlgorithm class.

This package contains the adapter classes (ClusterAlgorithmBorderFlow, ClusterAlgorithmKNN, ClusterAlgorithmMCL) for all supported clustering

algorithms as shown in figure 8. Each of these adapter classed is inherited from the abstract ClusterAlgorithm class and implements at least the abstract clustering method as shown in figure 9. These adapters are managed by the ClusterContext class.

The following example explicates how to implement and integrate an adapter (ClusterAlgorithmFoobar) for self-developed algorithm (Foobar). The general is pretty simple, all you need to do is:

1. Extend ClusterAlgorithm class by overriding the member String arrays to set new configuration options and implement the abstract clustering method.

```
public class ClusterAlgorithmFoobar extends ClusterAlgorithm{

  public ClusterAlgorithmFoobar(){
    config_A = new String[]{"Label_A","value_1","value_2"};
  }
  @Override
  protected Table clustering(
            String[] seeds,  double threshold,
            String values_A, String values_B,
            String values_C, String values_D ){

    Foobar.loadFile(filename, getSeparator());
    List<TreeSet<String>> clusters = Foobar.clusters(values_A);

    Table table = getTable();
    for(TreeSet<String> cluster : clusters)
      table.set(table.addRow(), CLUSTER_COLUMN_NAME, cluster);

    return table;
  }
}
```

2. Integrate the adapter by modifying the cvk.gui.Main.clusterVisualizationKitDemo() method which starts CVK:

```
Model model = new Model();
ClusterContext cc = model.getClusterContext();
cc.addAlgorithm("Foobar", new ClusterAlgorithmFoobar());
JPanel content = new Application(model).getJContentPane();
```

In 1. the four String parameters of the clustering method, comes from the member String arrays config_A ,..., config_D and can be used to provide possible paramaterizations of the novel algorithm. For example, the implementation of kNN uses config_A = new String[]{"k","1",...,"100"} for the parameter k. It is important to notice that the first element of the array is the JLabel for the parameter. A user can choose the other elements of each array from a JComboBox in the application. These are the values that are sent back to the clustering method as parameters so that they can be used for configurations.

The String array parameter (seeds) gets all nodes that were added in Nodes table (figure 5-left) to use as seeds. If no seeds added to the table, the seeds parameter will be null.
The double parameter (threshold) gets a value between 0 and 1 selected by the user from the Threshold slider.
The return statement is a prefuse.data.Table with a specified column which holds the cluster data sets. The column name (CLUSTER_COLUMN_NAME) and data type (CLUSTER_COLUMN_NAME_TYPE) are defined in ClusterTableSettings interface and are set to "Cluster" and TreeSet of String. The getTable() method gets a valid table instance.
The String member variable filename contains the path to a delimiter separated text file, the used separator get back by getSeparator() method.
It is possible to add more informations to the returned prefuse.data.Table instance, which will be displayed in the cluster list under the graph tab.

In 2. the first parameter of addAlgorithm(...) method is the name of your algorithm shown in the application and has to be different from already existing names. The second parameter is your algorithm instance. By starting the application all algorithm adapters will be instantiated.

cvk.harden

Figure 10: Abstract Harden class.

This package contains the hardening strategies (described in section 6). The hardening techniques implemented in this package can be used by all other algorithms implemented within the CVK.

A detailed description for developers will come soon. Until then, we refer to the implementation of the BorderFlow adapter (ClusterAlgorithmBorderFlow).

7.1.2 The bf package

This package contains the BorderFlow and the k-Nearest Neighbors algorithms (described in section 4), which were available from the beginning of this project on. The BorderFlow algorithm was developed independently from the CVK and integrated in agreement with its main developer.

7.1.3 The mcl package

This package contains the implementation of the MCL algorithm adapter.

7.2 Dependencies

7.2.1 Prefuse

ClusterVisualizationKit uses the prefuse toolkit and therefore the prefuse library beta-20071021 has to be included in your lib folder.
`http://prefuse.org`

7.2.2 Lucene

The prefuse toolkit supports a fast search with the Lucene search engine and we use it certainly. The library of Lucene search engine version 1.4.3 has to be included in your lib folder also.
`http://lucene.apache.org/java/1_4_3`

7.2.3 Jung

The "Java Universal Network/Graph Framework" is used to generate random graphs.
`http://jung.sourceforge.net`

7.3 Bugs

The application contain no critical bugs (i.e., bug that lead to a system failure) as far as we are awate. Still some minor bugs are still included in the library we utilize as listed in the subsequent section. Please do not hesitate to contact us if you have any helpful tips or comments on how to avoid these bugs. Furthermore, any feedback on unknown bugs will be greatly appreciated.

java.lang.IllegalArgumentException

An IllegalArgumentException with an invalid row index is thrown after a clustering task sometimes.

7.4 Change Log

Version - beta 0.4 (9. August 2010)

Added	toolbar, java look and feel
Added	measurements
Added	mcl algorithm
Fixed	some small bugs

Version - beta 0.3 (7. July 2010)

Added	java doc to source
Added	graph generator for cliques and Erdos-Renyi
Added	harden for all algorithms
Added	split package cvk.data to package cvk.data.cluster
Fixed	covered draw of clusters
Fixed	ClusterAlgorithm class hierarchy
Fixed	some small bugs

Version - beta 0.2 (23. Jun 2010)

Added	image export
Added	graphml and delimiter file support
Added	load example graphs from menu
Added	log to file
Added	choice of a node from node list
Added	edge list in graph statistic
Added	knn algorithm
Fixed	ConcurrentModificationException
Fixed	some small bugs

Version - beta 0.1 (31. May 2010)

First release.

References

[1] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis (Wiley Series in Probability and Statistics)*. Wiley-Interscience, March 2005.

[2] Axel-Cyrille Ngonga Ngomo and Frank Schumacher. Border flow a local graph clustering algorithm for natural language processing. In *Proceedings of the 10th International Conference on Intelligent Text Processing and Computational Linguistics (CICLING 2009)*, pages 547–558, 2009. Best Presentation Award.

[3] Peter Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.*, 20(1):53–65, 1987.

[4] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 2000.

[5] S. van Dongen. *A Cluster algorithm for graphs*. PhD thesis, Centrum voor Wiskunde en Informatica, 2000.