Department of Computer Science
Algorithmic Bioinformatics

Universitätsstr. 1 D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Developing and Evaluating a Cytoscape App for Graph-Based Clustering

Philipp Spohr

Entwicklung und Evaluation einer Cytoscape-App für graphbasiertes Clustering

Submission:	22.11.2017
Supervisor:	Prof. Dr. Gunnar Klau
Second Assessor:	Prof. Dr. Stefan Conrad
Advisors:	Martin Engler
	Sven Schrinner

Declaration

I hereby confirm that this thesis is my own work. I have documented all sources and tools used. Any direct or indirect quote has been marked as such clearly with specification of the source.

Düsseldorf, November 19, 2017

Philipp Spohr

Abstract

Cluster analysis is the task of grouping objects so that objects in a group are similar to each other and dissimilar to objects of other groups. This technique is commonly used in many areas as a preprocessing step to analyze data. The Yoshiko algorithm is an algorithm for cluster analysis that utilizes a graph-based approach. It groups objects into clusters by transforming their relationships into a graph, reducing it using six reduction rules and then solving Weighted Cluster Editing (WCE).

WCE is the problem of choosing a subset of edges in an undirected complete weighted graph so that the edge-induced subgraph consists of disjointed cliques and the sum of edge weights is maximized. This is achieved by either applying a heuristic approach or solving an integer linear program (ILP) to calculate an exact solution. In this thesis we describe an implementation of the Yoshiko algorithm as an application for the network visualization software Cytoscape. Thereby, we provide an intuitive tool for cluster analysis. It allows the user to set edge-weights directly or via mapping and allows edges to be explicitly included in or excluded from the solution.

We then evaluate the algorithm in terms of running time and quality and compare it to TransClust, another clustering algorithm based on WCE. In addition, we analyze the effect of the reduction rules on the running time of the algorithm. For this analysis we use the clustering evaluation framework ClustEval with 16 real world and synthetic data sets.

We identified two rules suitable for reducing real-valued instances and found that – on the data tested – reduction is inefficient when using the heuristic mode as it increases running time. While the running time of Yoshiko in heuristic mode and TransClust was found to be similar, the running time of the ILP mode was highly dependent on the reduction, with good reductions ($> 95\%$ of input size) reducing the running time by several orders of magnitude.

Overall, we found that both modes of the Yoshiko algorithm generate solutions of similar or identical quality compared to TransClust. From this we draw the conclusion that, at least for the instances tested, exact algorithms for WCE do not result in a solution quality justifying the increase in running time.

Additionally, we observed that both graph-based algorithms performed weak on path-like structures (where groups are not highly connected).

Contents

1 Introduction

Today, we are generating and processing more data than ever. Social networks collect a large amount of information about users and their behavior. In biology, databases containing information about the microscopic components of life are growing: Every day researchers make new observations about DNA sequences, protein interactions or molecule structures. There are countless other examples from all disciplines as information technology has become widespread and indispensable in nearly every field. In order to analyze this data and gain new insights, one of the firsts steps is usually the grouping of similar objects in order to recognize underlying structures. We want objects that share a group to be similar while being dissimilar to objects of other groups. The process of creating such groups is usually referred to as cluster analysis. There are various algorithms for data clustering, each with their advantages and disadvantages, and even more implementations of these algorithms for different systems and programming languages. As cluster analysis becomes more common and frequently used by people with different backgrounds, intuitive graphical interfaces that are easy to use are gaining relevance.

1.1 Cytoscape

Cytoscape [17] is an open-source network visualization and analysis software. It allows modeling of various network types and provides many built-in tools for layout and analysis of the data. One of its strengths is the support of community-provided applications (more commonly referred to as *apps*) that enhance it and offer additional functionality. Many of the popular (most downloaded) apps offer clustering algorithms [7]. Cytoscape provides an open API as well as tutorials for developers to enable the creation of apps. An advantage of software such as Cytoscape is that it allows users to perform their data analysis in a more intuitive and accessible way compared to command-line tool pipelines.

1.2 Yoshiko

Yoshiko [12] is a command-line tool for cluster analysis written in C++ that uses a graph-based approach (see Section 2.2). We will refer throughout this thesis to both, the command-line tool as well as the underlying algorithm (see Section 2.3), as **Yoshiko**.

1.3 TransClust

TransClust [21] is another graph-based clustering tool based on the same graph problem as Yoshiko, making it a natural choice for a comparison. It was found to be among the top performing algorithms when applied to biomedical datasets [24].

1.4 Objective

For this thesis we modified Yoshiko to be used as a library for other software. Next, we developed a clustering application based on Yoshiko as an app for Cytoscape. We then analyzed the different configurations for Yoshiko in terms of running time and solution quality on test data sets. Additionally we compared those results to TransClust. Finally we will discuss the limitations of our methods as well as potential future work and research.

1.5 Methods and Resources

The software created for this thesis is based on the C++ command-line tool [12] of the Yoshiko algorithm. It makes use of LEMON [5], a graph library, and IBM ILOG CPLEX Optimization Studio [11], a solver for linear programs. The Cytoscape app is implemented in Java using the Cytoscape Swing App API [6]. For the creation of the evaluation software, which was written in Java, the ClustEval library [22] was used. The evaluation experiments were performed on the HHU (University of Düsseldorf, Germany) HPC-System using 16 Intel(R) Xeon(R) E5-2667 v4 cores (3.20 GHz) as well as 512 GB RAM.

2 Algorithmic Background

2.1 Notation and Basic Definitions

For this thesis we make the following definitions to provide a shared terminology with our readers.

Definition 1 (Undirected Graph)
An undirected graph *is a tuple* (V, E) *where* V *is the set of vertices and* E *the set of edges with* $E \subseteq \binom{V}{2}$.

Definition 2 (Undirected Complete Graph)
An undirected complete graph *is a graph* $G = (V, E)$ *where* $E = \binom{V}{2}$.

Definition 3 (Undirected Complete Weighted Graph)
An undirected complete weighted graph *is a tuple* (G, c) *where* G *is a complete graph* $G = (V, E)$ *and* c *a cost function* $c : E \to \mathbb{R} \cup \{-\infty, \infty\}$.

Throughout this thesis we will refer to $c(e)$ as the *weight* of e.

Definition 4 (Vertex-Induced Subgraph)
A vertex-induced subgraph G_S *with* $G = (V, E)$ *and* $S \subseteq V$ *is defined as a graph with* S *as the set of vertices where the set of edges contains all edges of* G *that have both endpoints in* S.

Definition 5 (Edge-Induced Subgraph)
An edge-induced subgraph G_S *with* $G = (V, E)$ *and* $S \subseteq E$ *is defined as a graph where the set of edges is* S *and where the set of vertices contains all vertices of* G *that are incident to an edge in* S.

Definition 6 (Clique)
A clique *of a complete graph* $G = (V, E)$ *is a subset* $V' \subseteq V$ *so that the vertex-induced subgraph* $G_{V'}$ *is a complete graph.*

Definition 7 (Cluster Graph)
A cluster graph *is a graph that consists of disjointed cliques.*

Definition 8 (Conflict Triple)
A conflict triple *in an undirected graph* $G = (V, E)$ *is a triple* $\{u, v, w\} \in \binom{V}{3}$ *so that* $\{u, v\} \in E \wedge \{v, w\} \in E \wedge \{u, w\} \notin E$ *(see Figure 1).*

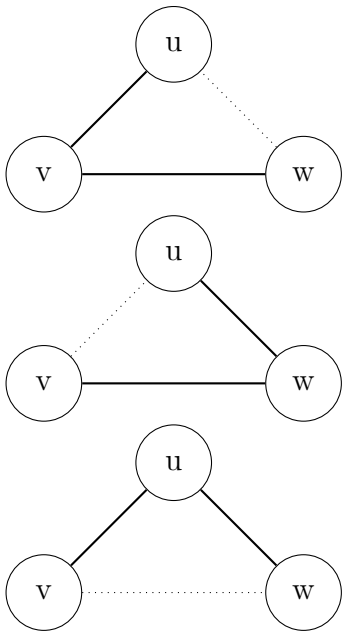


Figure 1: The three possible conflict triples

2.2 Weighted Cluster Editing

We have now covered the prerequisites necessary to move to our main focus.

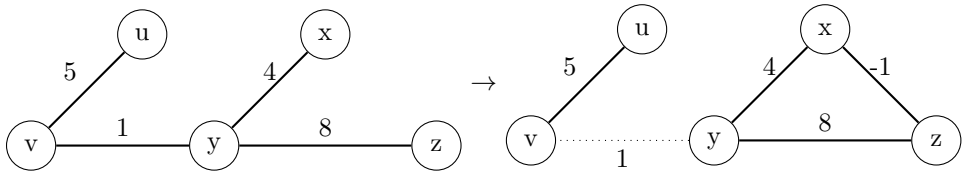
Definition 9 (Weighted Cluster Editing)
Given an undirected complete weighted graph: Choose a subset $E' \subseteq E$ so that the edge-induced subgraph of E' is a cluster graph and the sum of edge costs $\sum_{e \in E'} c(e)$ maximized.

Definition 10 (Editing Costs)

For a solution S with subset E' we call $EC(S) = \sum_{e \in E} \begin{cases} c(e) & \text{if } e > 0 \wedge e \notin E' \\ |c(e)| & \text{if } e < 0 \wedge e \in E' \\ 0 & \text{else} \end{cases}$

the editing costs *of S.*

This can be intuitively understood as the absolute values of negative edges, which are part of the solution, as well as the values of positive edges, which are not part of the solution, summed up.



For all missing edges $c(e) = -1$

Figure 2: An example of WCE and an optimal solution S with $EC(S) = 0$

As a practical measure we will not take edges with costs in $\{-\infty, \infty\}$ into account when calculating the sum of edge costs. Instead we will call an edge e with $c(e) = \infty$ permanent and always include it in the solution. An edge e with $c(e) = -\infty$ is called forbidden and will never be included in the solution. We note that the edge-induced subgraph is a cluster graph if and only if it contains no conflict triples [3]. The problem WCE was found to be NP-complete [16]. However, many instances can still be solved in a reasonable running time [2].

2.3 Yoshiko Algorithm

We can now describe how the Yoshiko algorithm operates (see Figure 3). The basic idea is to transform the input data into a WCE instance. We then solve WCE and interpret the cliques in the cluster graph as the desired clusters.

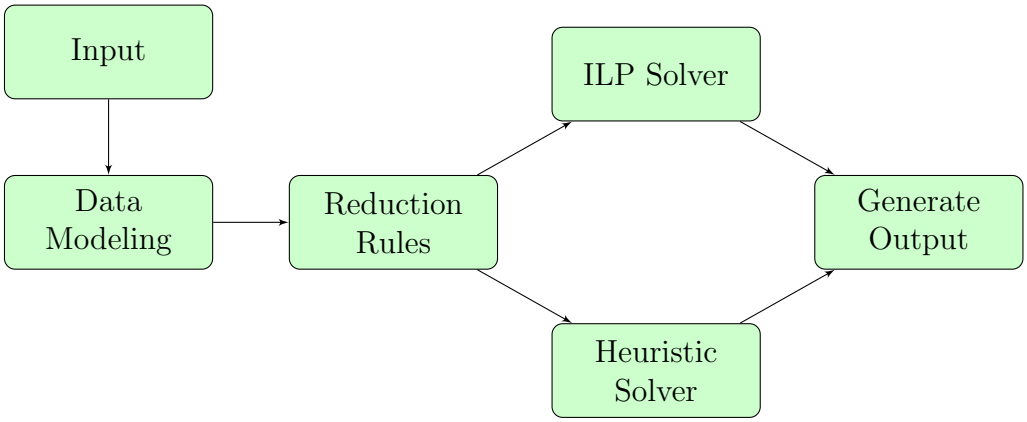


Figure 3: Overview of the Yoshiko Algorithm

2.3.1 Data Modeling

As a first step, we transform our raw input data into an undirected complete weighted graph. Depending on the format of the input data, we take different approaches. We will present and discuss two of them in this thesis as we employ one for the Cytoscape app implementation (see Section 3.2) and another for the evaluation (see Section 4.1.1). The Yoshiko command line application provides support for parsing various additional input formats.

2.3.2 Reduction Rules

The algorithm then reduces the instance by applying six reduction rules exhaustively. We will not cover those rules in this thesis but rather just acknowledge some of their properties and later analyze their performance. For a detailed description of the reduction rules refer to Böcker, Briesemeister, and Klau [2] as well as Böcker et al. [1]. The reduction rules identify edges that can not be part of an optimal solution for WCE or edges that are required to be part of an optimal solution. Vertices that are known to be connected can then be merged (while retaining the information about the original nodes and edges). Therefore, we note that the reduction rules do not affect the solution but may reduce the runtime. Previous analysis has shown those rules to reduce many unweighted instances of cluster editing (meaning the cost function is specified as $c : V \to \{-1, 1\}$) to trivial or at the very least small instances while performing less efficient on weighted instances [2]. The reduction process can result in disjointed graph components which then can be treated as separate instances.

2.3.3 Solving

The remaining reduced instances can be solved in two ways. We can either apply a heuristic that generates a cluster graph but does not necessarily maximize edge costs or calculate an exact and therefore optimal solution using an ILP.

ILP An exact solution for WCE can be gained by modeling and solving the following *integer linear program* (ILP) [10]:
Let (G, c) be an undirected complete weighted graph with vertices V and edges E.

Let $X : E \to \{0, 1\} = \begin{cases} 1, \text{ if the edge is part of the solution} \\ 0 \text{ else} \end{cases}$

We now maximize

$$\sum_{e \in E} X(e)c(e)$$

subject to

$$\begin{aligned} X(\{u, v\}) + X(\{u, w\}) - X(\{v, w\}) &\leq 1 \\ X(\{u, v\}) - X(\{u, w\}) + X(\{v, w\}) &\leq 1 \\ -X(\{u, v\}) + X(\{u, w\}) + X(\{v, w\}) &\leq 1 \end{aligned}$$

for each $\{u, v, w\} \in \binom{V}{3}$.
Using the inequalities as constraints, we guarantee that the solution contains no conflict triples as each inequality represents one possible conflict triple (see Figure 1). Therefore, our definition of WCE lends itself to formulation as an ILP naturally. A solution S for this ILP is an optimal solution for the WCE problem [10]. Constructing a trivial example where all edges carry a weight of zero, we realize that multiple optimal solutions are possible.
As we are restricted in X to 0 and 1 we can specify this program as a binary linear program.

Heuristic Algorithm Yoshiko uses a modified version of the kernel algorithm proposed by Böcker et al. [1] as a heuristic. For this approach we modify our graph model slightly: We treat edges e with $c(e) \leq 0$ as non-existing edges and call edges e with $c(e) = 0$ zero edges. Let (G, c) be an undirected complete weighted graph with vertices V and edges E. We call $E^+ = \{e : e \in E, c(e) > 0\}$ the set of existing edges.

Definition 11 (Shared Neighborhood)
The shared neighborhood *of vertices u and v is*
$SN(u, v) = \{x \in V \setminus \{u, v\} : \{u, x\} \in E^+ \wedge \{v, x\} \in E^+\}$

Definition 12 (Exclusive Neighborhood)
The exclusive neighborhood *of vertices u and v is*
$EN(u, v) = \{x \in V \setminus \{u, v\} : \{u, x\} \in E^+ \veebar \{v, x\} \in E^+\}$

This leads us to the following two definitions, which are essential to our heuristic:

Definition 13 (Induced Costs for Setting an Edge to Forbidden)
$icf(\{u, v\}) = \max\{0, c(\{u, v\})\} + \sum_{w \in SN(u,v)} \min\{c(\{u, w\}), c(\{v, w\})\}$

Definition 14 (Induced Costs for Setting an Edge to Permanent)
$icp(\{u, v\}) = \max\{0, -c(\{u, v\})\} + \sum_{w \in EN(u,v)} \min\{|c(\{u, w\})|, |c(\{v, w\})|\}$

We call $icf(e)$ the induced costs of excluding edge e from the solution (marking it as *forbidden*) and $icp(e)$ the induced costs of including edge e in the solution (marking it as *permanent*). If we set an edge $\{u, v\}$ to forbidden, we know that there can be no shared neighbors of u and v in the solution as that would result in a conflict triple. By assuming that, for each shared neighbor, the edge carrying the lower weight would have to be removed from the solution as well, we will avoid this.

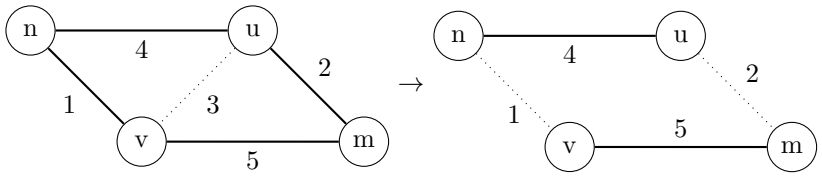


Figure 4: An example for $icf(\{u, v\}) = 6$

If we set an edge $\{u, v\}$ to permanent we can simply merge both vertices into a new vertex uv. It then needs to be decided whether we want to connect or disconnect the vertices adjacent to uv and we will simply make the cheapest choice in terms of editing cost.

Deciding to disconnect a vertex w means we will add a new edge

$$n = \{uv, w\} : c(n) = -(|c(\{u, w\})| + |c(\{v, w\})|)$$

whereas connecting it we would add

$$n = \{uv, w\} : c(n) = |c(\{u, w\})| + |c(\{v, w\})|.$$

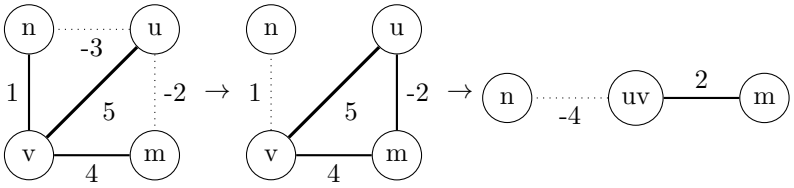


Figure 5: Setting $\{u, v\}$ to *permanent*, $icp(\{u, v\}) = 3$

The heuristic algorithm as described by Laude [13] works as following:
We calculate *icp* and *icf* for each edge. As a next step we choose the edge $\{u, v\}$ that yields the highest *icp* or *ilp* value. We then make a decision to either include or exclude it from the solution. If $icp(\{u, v\}) < icf(\{u, v\})$ we mark it as permanent (merging the incident nodes) else as forbidden (setting $c(\{u, v\}) = -\infty$). By directly merging nodes we guarantee that no conflict triples are created. Due to the fact that the induced costs may now have changed for all edges incident to u or v we will update *icp* and *icf* for all relevant edges. The process is then repeated with the next edge. We terminate the algorithm when all edges are set as either permanent or forbidden.

3 The Yoshiko-App for Cytoscape

3.1 Technical Details

This work builds on the C++ implementation of Yoshiko [12]. As the original implementation did not provide any interface for other applications to connect to, one of the first tasks was adjusting the software to support access from other applications. In order to achieve this, we restructured the program in a more modular fashion by strictly separating input-parsing, reduction-rule application, solving of the reduced instances and output generation.

As a next step, support for a virtual input format was added, meaning the generation of the graph instance could be controlled purely through function calls. A basic application programming interface (API) was then created to enable external software to use the tool as a library. Those functions were exposed via the Java Native Interface (JNI) convention to specifically allow access from Java. In order to improve the quality of incomplete processes in case of an early termination, we added a sorting step to the algorithm after the reduction phase in which we sort the instances by size in ascending order. We experienced that most runtime is spent on one difficult instance while other, smaller instances are solved fast. Therefore we decided to solve those first and thereby guarantee that a termination of the algorithm while solving the biggest instance still yields a result where all smaller instances are already finished. Finally a graphical interface that uses the Yoshiko library was created in Java as a Cytoscape app, which we will present in the following sections. In order to enable callbacks from the C++ part of the software back to Java we created abstract C++ classes and implemented them in Java. The sources for both, the library (the modified version of Yoshiko) as well as the Cytoscape app are available, refer to Appendix A.

Cytoscape framework In order to describe the functionality of the app we need to briefly cover some properties of Cytoscape. Graphs (referred to as networks in Cytoscape) are displayed visually in a user-defined style, meaning that the user can choose vertices (or nodes in Cytoscape) to be displayed for instance as labeled circles or squares. In addition, Cytoscape has two major panels that allow the user to interact with apps: One displaying interfaces and the other output. In both panels, the apps are represented as separate tabs.
For each network, tables containing additional information about the network can be defined. One table using the edges as index, the edge table, one using nodes as index, the node table, and one generic table holding information about graph properties always exist by default. The user has the option to create new tables or add columns to existing ones. One could for instance create a cost column in the edge table and insert values representing edge weights there. Each column contains one of five data types (String, Integer, Long Integer, Float or Boolean). Styles can be linked to tables, for instance a user can decide to scale node sizes according to a node table column containing numeric values.

3.2 Data Modeling

The Yoshiko algorithm models the data as an undirected complete weighted graph. As many input instances in Cytoscape do not describe such a graph missing edges and costs need to be modeled. This is achieved by using default values for insertion or deletion. We interpret missing edges as edges with a weight < 0. A default insertion cost $C_I \in [-\infty, 0]$ is used as $c(e)$ whenever the input instance does not contain an edge e whereas a default deletion cost $C_D \in [0, \infty]$ is used as $c(e)$ whenever the input instance does contain an edge e that has no cost associated.

Mapping Edge Costs The user has the possibility to use a numeric Cytoscape edge table column (which may contain integers or floating point precision numbers) as a source for the edge-cost function C. We would therefore set the cost function for an edge to a user-defined entry in this table column, indexed by the edge.

Insertion and Deletion Cost The default values C_I and C_D can be set by the user with the default values being $C_I = -1$ and $C_D = 1$. This corresponds to the unweighted cluster editing problem, assuming that no edge-cost function is mapped. It should be noted that the insertion cost value is not normalized by the software or in any way adjusted when a mapping is used. This means that the user needs to choose this value wisely to fit the data. As an example the user might have mapped the edge costs to a column containing values in the range of $10^6 - 10^7$. The default insertion cost of -1 is small in comparison and the algorithm will most likely insert all missing edges and generate one big cluster as a solution.
In general, the ratio $R = \frac{|C_I|}{C_D}$ can be interpreted as a density-value. $R > 1$ means, that the algorithm is more likely to delete edges in the process of generating cliques (resulting in smaller clusters), a value of $R < 1$ means insertions are more likely (resulting in bigger clusters).

Mapping Permanent or Forbidden Edges The Yoshiko-App has additional convenience functions: The user can map edges to a boolean Cytoscape column to mark them as either *forbidden* (meaning that those edges will never be part of the solution) or *permanent* (meaning that those edges will always be part of the solution). Marking an edge e as forbidden is equivalent to $c(e) = -\infty$, marking an edge e as permanent is equivalent to $c(e) = \infty$. This way the user is able to apply expert knowledge in the process of modeling a suitable WCE instance in order to increase the quality of the solution.

Marking edges as forbidden or permanent takes precedence over mapping edge costs.

3.3 Reduction Rules

The application allows toggling each reduction rule (refer to Section 2.3.2) on or off. As a default option all rules are enabled, thus adopting the default setting for the Yoshiko command line software. In addition to choosing the rules, the user has the option of specifying a factor for the *Similar Neighborhood Rule*. This option may improve the efficiency of this rule when using real valued edge weights, as the rule itself rounds all numbers and uses integers in order to execute a dynamic program. Therefore multiplying all edge values with a factor > 1 results in a higher resolution (at the cost of a higher running time). For example the numbers 1.1 and 1.2 would be rounded to 1, losing information in the process whereas multiplying them by 10 would have them represented as 11 and 12 respectively. We will give some cautious advice for using those reduction rules later on (see Section 4.2.2). As we consider those options to be *advanced* they are by default hidden in the interface as to avoid confusion.

3.4 Status Bar

Due to the fact that the algorithm might take a substantial amount of time, the Yoshiko-App provides feedback about the process through a status bar. It informs the user about the currently applied reduction rule and – if the algorithm is in the ILP solving step – outputs the current gap (meaning $1 - \frac{l_b}{u_b}$, where l_b is the lower bound for the editing costs of the solution and u_b the upper bound). The user also has the option to cancel the process via a button included in the status bar which will result in the attempt to fetch a (potentially suboptimal) solution.

3.5 Presenting Solutions

The application gives the user a simple and clean interface for exploring and interacting with the results. At the top of the result panel that opens when the algorithm is finished or has terminated basic information is displayed for instance how many clusters were found, the gap between lower and upper bound in ILP mode or how high the editing costs for generating the solution were.

As multiple optimal solutions are possible the user can switch between them via tabs. The clusters are visually displayed in a list using the currently selected style and layout method to render them. Upon selection, the nodes in a cluster are highlighted in the original graph. The user can also choose to export clusters as separate graphs (networks).
We also added a meta-graph feature (see Figure 6). This will generate one subgraph for each cluster and then one graph where each node represents one cluster. Nodes in this meta-graph are scaled to represent cluster size and the edge size between clusters reflects how many edges exist between nodes belonging to the respective clusters. Given a sufficient zoom-level the user can also see the subgraphs embedded in the meta-graph. Through the context menu, accessible by right-clicking a node, the user can move into the subgraph of a given cluster node. Modifications to the original graph are registered and a warning is issued to the user that the solution might no longer be accurate. When a meta-graph is active, selecting a cluster in the solution tab results in the corresponding cluster node being highlighted. All subgraphs and meta-graphs can be disposed of when the result is discarded, cleaning the workspace.

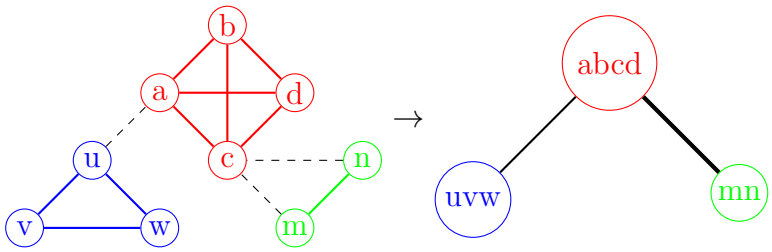


Figure 6: Model of the meta-graph feature

3.6 Additional Options

Library Selection The library section allows selection of the Yoshiko library. The location is saved on exit so it usually needs to be set only once. It also displays basic information about the version and contains a link to the repository where it can be downloaded.

Number of Solutions As the algorithm can generate multiple optimal solutions the user has the option to limit the number of solutions that are to be calculated (in order to save running time).

ILP Time Limit Due to the fact that the ILP part of the software can be very expensive in terms of running time a time limit can be set. After expiring the user is informed and can either retrieve a (possibly) suboptimal solution or continue the algorithm.

Multithreading By default CPLEX uses as many cores as possible. This might lead to the system becoming very unresponsive. The user has the option to limit CPLEX to a single thread, which might be useful for experimenting on low-end to mid-range systems in order to maintain a responsive operating system.

Experimental Callbacks Two experimental callbacks for CPLEX, namely *Triangle Cuts* and *Partition Cuts*, can be toggled on or off. They modify the way CPLEX operates in solving the ILP. We will not cover those in detail but rather give some basic observations about their performance later on. For more details refer to Böcker, Briesemeister, and Klau [2].

Help System We linked some panels like the data modeling panel to the relevant chapters of this thesis, hosted in the Yoshiko-App repository, enabling advanced users to refer to it as an in-depth manual.

4 Evaluation of the Yoshiko Algorithm

Evaluating a clustering algorithm is a difficult task. What a valid cluster is highly depends on the application and while a certain model might be well suited for a given task, it could easily fail on other tasks. There is also the technical difficulty of having to parse data sets that might appear in any representation into a format suitable for a given algorithm. In addition, many clustering algorithms work with one or more parameters that need to be set as they determine the quality of the solution. In order to attempt an evaluation, we use the ClustEval [23] framework, a software kit that converts data sets into various formats, runs specified cluster algorithms on them and then evaluates the found clusters with various quality measurements. It also allows automatized tuning of parameters to determine an optimal configuration for a certain data set.

4.1 Methodology

4.1.1 Input Processing

In order to apply our clustering method to the data sets, which are encoded in various formats, we use ClustEval to convert each data set into a similarity matrix.

Definition 15 (Similarity Matrix)
A similarity matrix *M for a data set with n entries $X_1, X_2, \ldots, X_n$ is an upper triangular matrix of size n with entries in $[0, \infty] \cup \{-\infty\}$, where each entry $A_{i,j}$ corresponds to the symmetric similarity between two data points X_i and X_j.*

To transform this matrix into an undirected complete weighted graph we use the same method that is utilized by TransClust. We use the following algorithm: For each data point X_i (or row A_i in the matrix) we create a vertex V_i. We then choose a threshold parameter $T \in [0, \infty)$. For each pair of nodes $\{V_a, V_b\} : a \neq b$ we insert an edge with weight $A_{a,b} - T$.
Note that this transformation preserves forbidden and permanent edges (with weights $\in \{-\infty, \infty\}$).
The threshold parameter can be interpreted as a density parameter.
If we choose

$$T = \min\{A_{i,j} : 1 \leq i < j \leq n\} + \varepsilon$$

with a small $\varepsilon > 0$ we generate relatively large clusters while choosing

$$T = \max\{A_{i,j} : 1 \leq i < j \leq n\} - \varepsilon$$

with a small $\varepsilon > 0$ results in relatively small clusters. It is worth noting that $T \leq \min(\{A_{i,j} : 1 \leq i < j \leq n\})$ simply generates one big cluster whereas $T \geq \max(\{A_{i,j} : 1 \leq i < j \leq n\})$ will result in one cluster per vertex.

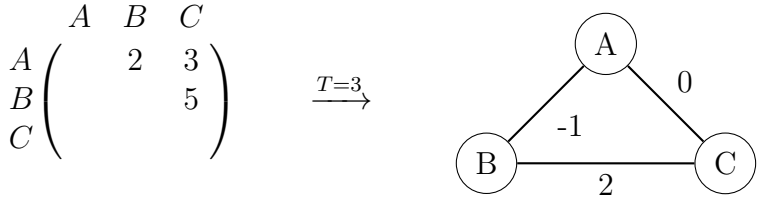


Figure 7: Transforming a similarity matrix into a WCE instance

4.1.2 Quality Measurements

In order to judge the quality of the clusters we apply two measurements.

Silhouette Value The silhouette value [15] is an intrinsic value, meaning that it can be calculated without knowledge of an optimal solution (gold standard). It is based on the concept that a good clustering has homogeneity between elements inside the same cluster and separation between elements that are in different clusters. In order to define it we need to cover some mandatory definitions.

Definition 16 (Average Dissimilarity To Own Cluster)
The average dissimilarity *of an object i to* its cluster *is* $a(i) = \frac{1}{|c_i|} \sum_{j \in c_i} d(i,j)$, *where d is a distance measure appropriate*[1] *for the data set and c_i the cluster i is in.*

Definition 17 (Average Dissimilarity To Other Clusters)
The average dissimilarity *of i to* all other clusters *is* $b(i) = \frac{1}{n-|c_i|} \sum_{j \in C \setminus c_i} d(i,j)$, *where n is the number of all objects and C the set of all objects.*

n and C correspond to $|V|$ and V in our graph model.

Definition 18 (Silhouette Value)
The silhouette value *is defined as* $S = \frac{1}{n} \sum_i s_i$, *where* $s_i = \frac{1}{n} \frac{b(i)-a(i)}{\max\{a(i),b(i)\}}$

F_1-Score The F_1-Score [14] is an extrinsic value, meaning that it can only be calculated with knowledge of an optimal solution. It originates from binary classification where either 0 or 1 is predicted and then compared to the actual occurrence.
If the hypothesis is 0 and the result 1 we call it a *False Negative* (FN).
If the hypothesis is 1 and the result 1 we call it a *True Positive* (TP).
If the hypothesis is 0 and the result 0 we call it a *True Negative* (TN).
If the hypothesis is 1 and the result 0 we call it a *False Positive* (FP).
In order to define the F_1-Score we need to define two expressions first:

Definition 19 (Precision)
The precision *of a prediction is defined as* $\frac{TP}{TP+FP}$.

Definition 20 (Recall)
The recall *of a prediction is defined as* $\frac{TP}{TP+FN}$.

[1] We are using similarity values as defined earlier. ClustEval generates these values by applying a suitable distance measure to the input file. As an example, data that represents points in the $\mathbb{R}^2$ linear vector space could be processed using the euclidean distance. For a detailed description of the conversions that are actually used, refer to Wiwie [23].

We can now define:

Definition 21 (F_1-Score)
The F_1-Score *of a prediction is defined as* $2 \cdot \frac{precision \cdot recall}{precision + recall}$

It has been found that the Silhouette value correlates strongly with the F_1-Score. It is therefore a good quality measure to optimize parameters for when no gold standard is known [24].

4.1.3 Divisive Parameter Optimization

To optimize our threshold parameter during the input processing step, we apply divisive parameter optimization using ClustEval. This method simply takes an integer parameter x and divides the range of similarity values into x uniform steps. We then apply our clustering algorithm and assess the results with a quality measurement of choice to determine the best parameter.

4.1.4 Data Sets

For this evaluation we choose 16 of the data sets provided by ClustEval. The data sets include synthetic data sets as well as real world examples. A gold standard is given for each data set describing the 'correct' clustering. Visualizations (if helpful) as well as sources are included in Appendix B. We excluded some data sets shipped with ClustEval. This was either due to the running time exceeding the scope of this project or due to errors, which we couldn't resolve, setting up the experiment.

4.2 Results

4.2.1 Parameter Optimization

As the ILP mode of the Yoshiko algorithm has a long running time, we optimized the threshold parameter using the heuristic mode under the assumption that both modes share a common optimal threshold. We used divisive parameter optimization with 100 subdivisions on all 16 data sets, choosing the silhouette value as the relevant criterion. We included the full data (see Appendix D).
As evident from the data it is difficult to give a general recommendation for the threshold. This is because the threshold/silhouette functions do not follow a simple pattern. More specific, they are not monotonous and have, in some instances, multiple maxima. Attempting to cluster using the full range of possible parameters can therefore not be avoided.

Another observation was that some instances had discrete steps, meaning that a broad range of parameters resulted in the same solution. Comparing the optimal thresholds to those found by TransClust (see Appendix C) we noted that we achieved the same threshold yielding the same silhouette value on many data sets while performing marginally worse on others. This led to the hypothesis that we might reach the exact same silhouette values using more subdivision steps. We then reran the experiment on the sets that showed a difference, using 1000 subdivisions and in addition clustered using the optimal threshold as determined by ClustEval. Unfortunately, our hypothesis was shown to be wrong as we achieved the same result (or even worse silhouette values when using the TransClust configuration).
We also attempted parameter optimization, using ILP mode on few instances with a reduced number of subdivisions and observed the exact same curves the heuristic showed though we would refrain from drawing any conclusions for now. In addition we looked at the curves for TransClust on some instances and noted that TransClust seems to be more consistent (see Appendix D:*ChangSpiral*), meaning that the average silhouette value across all parameters was higher.

4.2.2 Running Time Analysis

In order to determine the optimal configuration for Yoshiko we ran different configurations of the software, using the previously optimized threshold parameters.

Reduction Rules As a first step, we analyzed the efficiency of the reduction rules. Those rules do not affect the solution [2] so the only relevant measurement is running time. We made runs on the data sets with each of the six reduction rules toggled on alone, using the heuristic mode for the solving step. It should be noted, that all the data sets contained real numbers whereas some reduction rules are specifically engineered towards integers.
We found that *Clique Rule*, *Almost Clique Rule*, *Critical Clique Rule* and *Heavy Edge Rule* performed very poorly. On most data sets no reduction was achieved and on the remaining few we received reductions $< 2\%$.
In contrast, the *Parameter Dependent Rule* achieved reductions of 99% on some instances and the *Similar Neighborhood Rule* achieved reductions ranging from 0.003% to $\sim 55\%$. Both came with an increase in running time, the *Parameter Dependent Rule* elongated the process by a factor < 2 while *Similar Neighborhood Rule* increased it up to a factor of 1000. Using heuristic mode, no reduction resulted in an overall decreased running time.

We would therefore cautiously conclude that the first four rules are not suitable for real-valued instances. Furthermore we suggest using no reduction rules at all when running in heuristic mode and *Parameter Dependent Rule* as well as *Similar Neighborhood Rule* in ILP mode. For the full data refer to Appendix E.1

Comparison between ILP, Heuristic and TransClust We also compared both modes of Yoshiko to TransClust in terms of running time (see Figure 8). All reduction rules were turned on for the ILP mode – assuming that their increase in running time would be negligible compared to the ILP solving step – while we used none in heuristic mode.
Unsurprisingly, the ILP mode was very costly in terms of running time, exceeding TransClust and the heuristic mode on some instances by a factor of up to 10^4 or larger. TransClust and the heuristic mode of Yoshiko performed comparably. We note that the ILP mode achieved running times in the same magnitude on data sets where a large amount of reduction was achieved as is apparent on the instances *ColiFind*, *ColiNeed*, *ColiState* and *ColiTime*. The full data is included, refer to Appendix E.3.

Experimental Callbacks Finally we tested the two experimental callbacks we just briefly touched upon in Section 3.6 and compared their influence on running time. The *Triangle Cuts* callback appeared especially slow on many instances due to a lack of parallelization (The callback itself can not be run on multiple cores). Yet, it yielded exceptionally good speedups on some instances, for example *ChangSpiral* or *FuFlame*. We would therefore refrain from making a conclusive recommendation about its usage for now and suggest leaving it turned off as to avoid critical instances. Employing the *Partition Cuts* callback we achieved similar results to the basic ILP mode with a tendency of a slight reduction in running time over all instances. As a result we would advise using this callback on real-valued instances. For the full data refer to Appendix E.2.

4.2.3 Quality Comparison

As a final step, we used our optimized parameters and compared Yoshiko in heuristic and ILP mode to TransClust using the F_1-Score (see Figure 9). Overall both modes performed similar, with both heuristic and ILP mode often generating the exact same solution. The same holds for the comparison to TransClust. We could also observe that differences in F_1-Score were usually reflected in a difference in silhouette value though we also saw exceptions to that rule (See *Synthetic Spirals*).

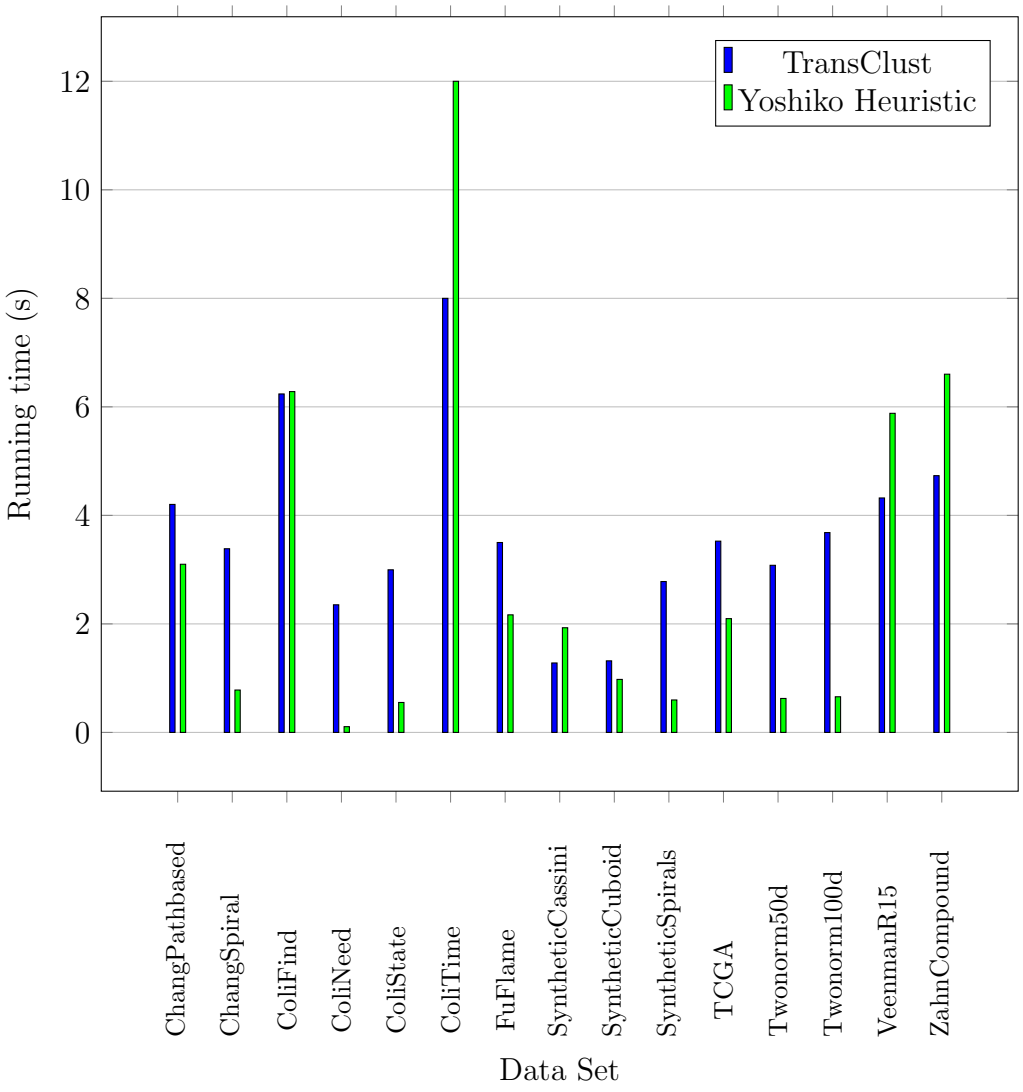
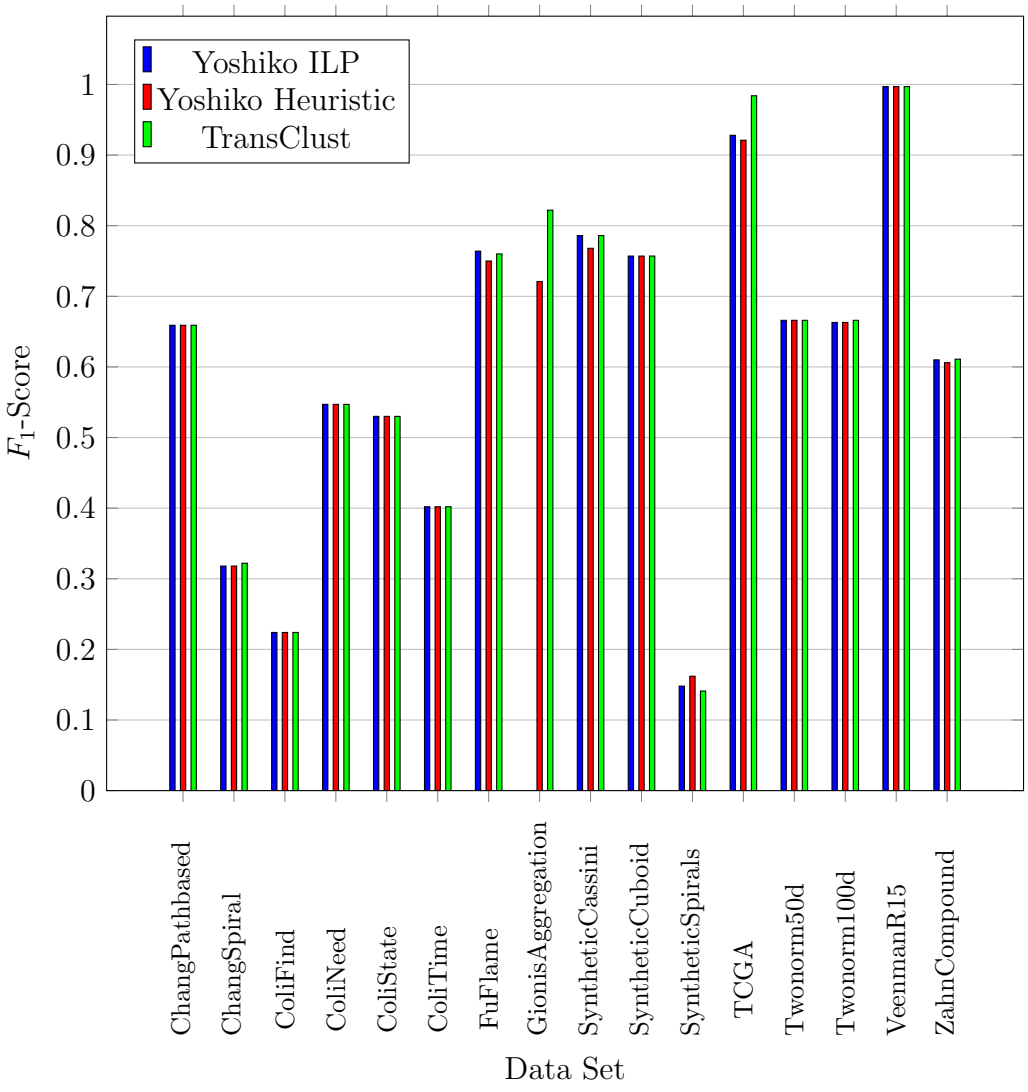


Figure 8: Running time comparison TransClust and Yoshiko Heuristic

Across all data sets the ILP mode did not always yield better results than the heuristic mode as can be seen looking at *Synthetic Spirals*. Summing this up we reach the conclusion that heuristic approaches to WCE yield good results. As the ILP mode calculates an exact solution to WCE we can conclude that the heuristic algorithms find the exact solution to WCE on many data sets as well. Equivalently, exact algorithms might not deliver results justifying the increase in runtime.



Missing F_1-Scores mean that the algorithm exceeded a runtime of six hours.

Figure 9: Clustering quality comparison

5 Outlook and Discussion

5.1 Multigraphs and Loops

As Cytoscape supports multigraphs and loops (meaning there can be more than one edge connecting the same pair of vertices as well as edges connecting a node to itself), we were faced with the situation that our model (WCE) was not sufficient for a generic Cytoscape clustering application. Before discussing how to modify the Yoshiko algorithm in order to cover multigraphs one needs to clarify how those structures are to be interpreted in terms of data clustering.
For example, given a graph representing interactions between objects, one could think of a multigraph where the edges contain two properties: The type of interaction and the strength of interaction. We could then reduce the graph instance under the assumption that the interaction type is chosen from a finite set by assigning an associated weight factor. Merging all edges connecting two vertices using the sum of the weighted strengths, we would get an undirected weighted graph. This method could be applied for a network of protein-protein interactions where multiple experiments are merged into one graph. In this case we would define a given experiment as an interaction type and assign a weight based on the quality of the experiment. There are other possible scenarios and it is therefore difficult to think of a generic approach to clustering such objects. (One could even discuss whether such a structure is in itself useful for clustering.)

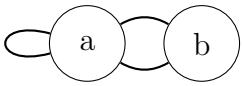


Figure 10: A small multigraph with a loop

There are also several interpretations of what a loop represents: In some cases it might represent feedback mechanisms and in others those loops might simply be artifacts from converting data. The basic model we use does not support loops at all but one could easily think of methods that take loops into account. One way would be a graph model where both, edges and vertices, contain weights and where the weight of a node is the weight of all incident edges summed up. In such a model loops would be taken into account.

For our implementation we decided to ignore both objects (and simply warn the user when his input data contains such an object). Future research could analyze whether the reduction rules are applicable to multigraphs and loops while retaining optimality and adjust WCE to support such structures.

5.2 Limitations of the Evaluation Method

While we achieved some valuable insights during the evaluation process there are still many open questions. This is in part due to the limited scope of the project. The overall assessment of runtime might be inaccurate due to various factors:
As we performed the evaluation on multiple cores using ClustEval we had no direct control over threading and could not guarantee a uniform processing of all configurations. We noted fluctuations running the same experiment multiple times although they remained $< 1s$.
For a comprehensive statement as to whether using the ILP is really useful one would need to perform a complete parameter optimization using the ILP or alternatively, find a proof for the assumption that an optimal heuristic threshold is always an optimal ILP threshold. We also think that the amount of data is unsatisfactory. In order to observe patterns and find optimal configurations depending on certain characteristics of the input instance we would have to analyze more data sets (for a start those that had an excessive runtime). In addition, we presume that a broader variety of input instances could be useful to gain additional insights. As an example, we had neither data sets containing *permanent* or *forbidden* edges (similarity values $\in \{-\infty, \infty\}$), nor data sets consisting of integer values.
Generally speaking, the process of transforming all input instances into similarity matrices might not be applicable for every real life application. One could think of a graph model, where missing edges have fixed costs independent of the similarity between their vertices. Forcing each input instance into a similarity matrix might then come with a loss of information.
Summing this up, we are looking forward to more future research on the aspects mentioned, hopefully resulting in new insights.

5.3 Limitations of WCE as a Clustering Model

WCE is an intuitive and successful model for the clustering of data but it can not be applied to all real-world problems. This is due to the fact that there is no fixed definition of what a 'valid' clustering is. While using WCE as a model assumes that the clusters are highly-connected groups of objects there might be cases that are not covered. We would like to elaborate on this using the data set *Synthetic Spirals* (see Appendix B) as an example. The data set consists of points in $\mathbb{R}^2$ describing two intertwined spirals.
When processing this object using the optimal threshold, Yoshiko transforms the similarity matrix into an undirected weighted complete graph. This graph consists of two connected components. We now note that those two connected components are already the clusters that one would intuitively group those points into by simply looking at the plot. Those components however, are not cliques. Therefore our algorithm will now split the spirals into little clusters (even triangles) as this is the most cost-efficient way of creating cliques. We are now left with a strange solution.
Generalizing this problem, one could define 'path-like' structures as such where objects are similar to at most two other objects of the same cluster while being potentially more dissimilar to other objects of their cluster than to objects of another cluster. Those structures are trivially not well-represented using WCE.
Further research could formalize such objects more precisely and find ways of identifying them. Our algorithm could then be adjusted to scan for those objects in a preprocessing step and change the approach accordingly.

5.4 Improving the Cytoscape-App

While the main functionality is implemented, the Yoshiko-App has many aspects that can be improved.
A natural step would be to publish the app or submit it to the Cytoscape app store. Right now this is impossible due to our software relying on CPLEX, which we can not distribute. We are therefore forced to require potential users having CPLEX installed in order to use the app. This could be avoided using a free alternative to solve the ILP. We could also imagine a basic version of the software employing the heuristic approach that would detect an existing CPLEX installation and then offer the ILP mode as an optional feature.
Similar to the method used during our evaluation process we could envision a threshold cutoff feature that would transform complete similarity graphs consisting exclusively of weighted positive edge values into a real valued instance.

As a user might be content with a solution that is not optimal in the sense of a WCE solution, a feature that would let the user specify a target gap and then cancel the ILP solving process as soon as this gap is reached, might be of benefit. Throughout the analysis we occasionally observed, that the solver spent most of its time closing gaps $< 1\%$. Such an option could therefore potentially reduce the runtime significantly.
Newer versions of Cytoscape focus on automation; We could therefore support callbacks to process multiple graphs using external scripts and generating the solutions as meta-graphs.
An important insight while working on this thesis was the importance of the graph modeling phase. In order to choose an efficient configuration for our algorithm as well as an appropriate method of transforming the input into a graph we need to have a clear understanding of our input data. While cluster analysis is usually thought of as the first step in processing data it might be useful to add more preprocessing steps. As of now the user is still responsible for representing her/his input data in a way that fits the WCE model. For instance, having a graph with exclusively negative edge weights requires transformations before we get a meaningful WCE problem as the algorithm would simply create one cluster per vertex otherwise. Similarly, as most Cytoscape networks are not complete graphs, choosing an appropriate insertion cost is vital to the quality of the resulting clustering. We could therefore implement different strategies for suggesting such a value (for instance a mean value of all edge weights). Throughout our evaluation process we also found that some instances can be processed in a short running time using the *Triangle Cuts* callback. It would be valuable to identify for which input instances this callback is helpful and for which it is not. Ideally a final version of Yoshiko could suggest a complete configuration based on the input instance to the user that would result in the lowest expected runtime and the highest expected silhouette value. Assuming that the instance is weighted with floating point precisions values, we would, for instance, turn off the first four reduction rules.
Another scenario that –as of now– is not supported is a modified version of WCE where the user specifies the number of clusters he wants to achieve. For this we would have to modify our model so that a fixed number of clusters is generated.
Most certainly, many other things can be added or improved. Having released the software as open-source software, we are looking forward to contributions, feedback or suggestions.

5.5 Integration in other Frameworks

Given the fact that Yoshiko has been transformed from a C++ command line application to a library it is now possible to implement the algorithm for other frameworks beside Cytoscape with relative ease. While Cytoscape is especially engineered towards working with biomedical data one could think of implementations for frameworks that focus on social networks but also completely different applications such as embedding the algorithm in language analysis tools.

6 Acknowledgments

I would like to express my deep gratitude to the supervisor of this thesis Prof. Dr. Gunnar Klau for his support and constructive feedback throughout the process. In addition I want to thank Martin Engler and Sven Schrinner for their advice and valuable guidance. Information from Dr. Christian Wiwie regarding his ClustEval platform was indispensable and I would like to thank him for his patience with my many questions. Computational support and infrastructure for the software evaluation was provided by the "Centre for Information and Media Technology" (ZIM) at the University of Düsseldorf (Germany). Lastly, I would like to specifically thank Philipp Rehs for assisting me in working with the HPC and even setting up a R environment with the required packages for this project.

7 Bibliography

References

[1] Sebastian Böcker, Sebastian Briesemeister, Quang B.A. Bui, and Anke Truss. "Going weighted: Parameterized algorithms for cluster editing". In: *Theoretical Computer Science* 410.52 (2009). Combinatorial Optimization and Applications, pp. 5467–5480. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2009.05.006. URL: http://www.sciencedirect.com/science/article/pii/S0304397509003521.

[2] Sebastian Böcker, Sebastian Briesemeister, and Gunnar W. Klau. "Exact Algorithms for Cluster Editing: Evaluation and Experiments". In: *Algorithmica* 60.2 (June 2011), pp. 316–334. ISSN: 0178-4617. DOI: 10.1007/s00453-009-9339-7. URL: http://dx.doi.org/10.1007/s00453-009-9339-7.

[3] Sebastian Briesemeister. "Weighted cluster editing for clustering biological data". Diplomarbeit. Friedrich-Schiller Universität Jena, Dec. 2007.

[4] Hong Chang and Dit-Yan Yeung. "Robust Path-based Spectral Clustering". In: *Pattern Recogn.* 41.1 (Jan. 2008), pp. 191–203. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2007.04.010. URL: http://dx.doi.org/10.1016/j.patcog.2007.04.010.

[5] Egerváry Research Group on Combinatorial Optimization. *LEMON LEMON C++ Graph Library.* URL: http://lemon.cs.elte.hu/trac/lemon (visited on 11/03/2017).

[6] Cytoscape Consortium. *Cytoscape 3.5.1 Cytoscape Swing App API.* URL: http://chianti.ucsd.edu/cytoscape-3.5.1/API (visited on 11/03/2017).

[7] Cytoscape Consortium. *Cytoscape App Store.* URL: http://apps.cytoscape.org/apps (visited on 11/03/2017).

[8] Limin Fu and Enzo Medico. "FLAME, a novel fuzzy clustering method for the analysis of DNA microarray data". In: *BMC Bioinformatics* 8.1 (Jan. 2007), p. 3. ISSN: 1471-2105. DOI: 10.1186/1471-2105-8-3. URL: https://doi.org/10.1186/1471-2105-8-3.

[9] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. "Clustering Aggregation". In: *ACM Trans. Knowl. Discov. Data* 1.1 (Mar. 2007). ISSN: 1556-4681. DOI: 10.1145/1217299.1217303. URL: http://doi.acm.org/10.1145/1217299.1217303.

[10] Martin Grötschel and Yoshiko Wakabayashi. "A Cutting Plane Algorithm for a Clustering Problem". In: *Math. Program.* 45.1 (Aug. 1989), pp. 59–96. ISSN: 0025-5610. DOI: 10.1007/BF01589097. URL: https://doi.org/10.1007/BF01589097.

[11] IBM. *CPLEX IBM ILOG CPLEX Optimization Studio*. URL: https://www.ibm.com/us-en/marketplace/ibm-ilog-cplex (visited on 11/03/2017).

[12] Mohammed El-Kebir, Gunnar Klau, and Emanuel Laude. *Yoshiko GitHub Repository*. URL: https://github.com/ls-cwi/yoshiko (visited on 11/03/2017).

[13] Emanuel Laude. "Identifikation von Arten durch gewichtetes Cluster-Editing". Bachelor's thesis. Julius-Maximilians-Universität Würzburg, Feb. 2013.

[14] Cornelis Joost van Rijsbergen. *Information Retrieval*. URL: http://www.dcs.gla.ac.uk/Keith/Preface.html (visited on 11/03/2017).

[15] Peter J. Rousseeuw. "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis". In: *Journal of Computational and Applied Mathematics* 20.Supplement C (1987), pp. 53–65. ISSN: 0377-0427. DOI: https://doi.org/10.1016/0377-0427(87)90125-7. URL: http://www.sciencedirect.com/science/article/pii/0377042787901257.

[16] Ron Shamir, Roded Sharan, and Dekel Tsur. "Cluster Graph Modification Problems". In: *Discrete Appl. Math.* 144.1-2 (Nov. 2004), pp. 173–182. ISSN: 0166-218X. DOI: 10.1016/j.dam.2004.01.007. URL: http://dx.doi.org/10.1016/j.dam.2004.01.007.

[17] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S. Baliga, Jonathan T. Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. "Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks". In: *Genome Res* 13.11 (Nov. 2003). 14597658[pmid], pp. 2498–2504. ISSN: 1088-9051. DOI: 10.1101/gr.1239303. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC403769/.

[18] Nora Speicher. "Towards the identification of cancer subtypes by integrative clustering of molecular data". Master's thesis. Universität des Saarlandes, Dec. 2012.

[19] Princeton University. *"About WordNet." WordNet. Princeton University*. URL: http://wordnet.princeton.edu (visited on 11/03/2017).

[20] Cor Veenman, Marcel Reinders, and Eric Backer. “A maximum variance cluster algorithm”. In: 24 (Oct. 2002), pp. 1273–1280.

[21] Tobias Wittkop, Dorothea Emig, Sita Saunders, Sven Rahmann, Mario Albrecht, John Morris, Sebastian Böcker, Jens Stoye, and Jan Baumbach. “Partitioning biological data with transitivity clustering”. In: 7 (June 2010), pp. 419–20.

[22] Christian Wiwie. *clusteval-lib Clusteval Library*. URL: `https://gitlab.compbio.sdu.dk/clusteval/clusteval-lib` (visited on 11/03/2017).

[23] Christian Wiwie. “Development of an integrated clustering evaluation framework for cluster analysis”. Master’s Thesis. Saarland University Center for Bioinformatics, May 2013.

[24] Christian Wiwie, Jan Baumbach, and Richard Rottger. “Comparing the performance of biomedical clustering methods”. In: *Nat Meth* 12.11 (Nov. 2015). Analysis, pp. 1033–1038. ISSN: 1548-7091. URL: `http://dx.doi.org/10.1038/nmeth.3583`.

[25] Charles T. Zahn. “Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters”. In: *IEEE Trans. Comput.* 20.1 (Jan. 1971), pp. 68–86. ISSN: 0018-9340. DOI: `10.1109/T-C.1971.223083`. URL: `http://dx.doi.org/10.1109/T-C.1971.223083`.

A Source Code

Note: The following links were checked on November 19, 2017.
The source code for the Cytoscape plugin is accessible at:
https://gitlab.cs.uni-duesseldorf.de/spohr/YoshikoWrapper
The latest version of the Yoshiko library can be found at:
https://github.com/spqrPh/yoshiko

B Data Sets

B.1 Synthetic Data Sets

The sets *Twonorm50d* and *Twonorm100d* are part of ClustEval and consist of data in $\mathbb{R}^{50}$ and $\mathbb{R}^{100}$ respectively.

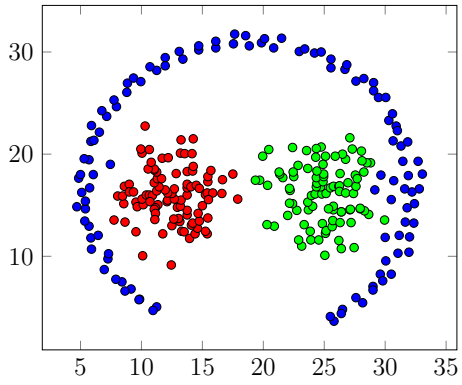


Figure 11: ChangPathbased[4]

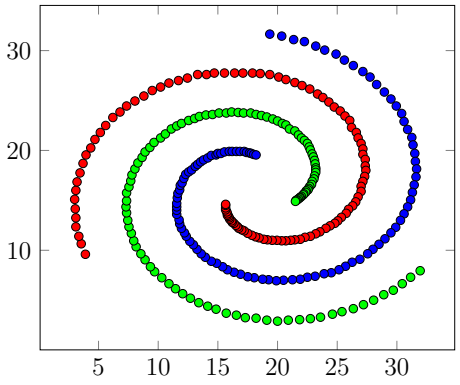


Figure 12: ChangSpiral [4]

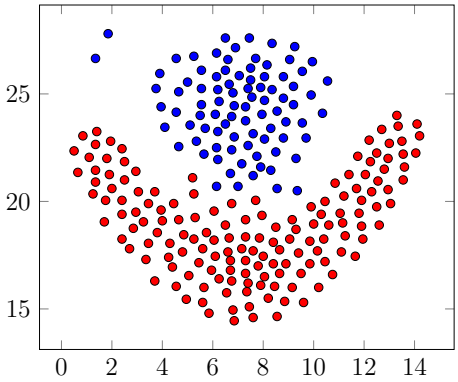


Figure 13: FuFlame [8]

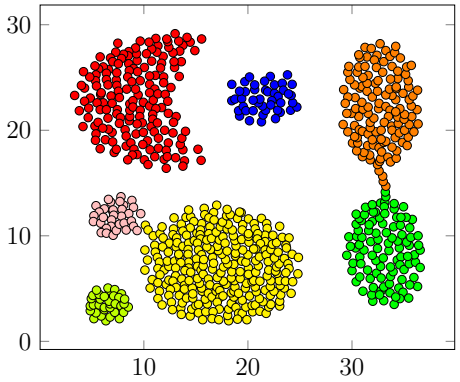


Figure 14: GionisAggregation [9]

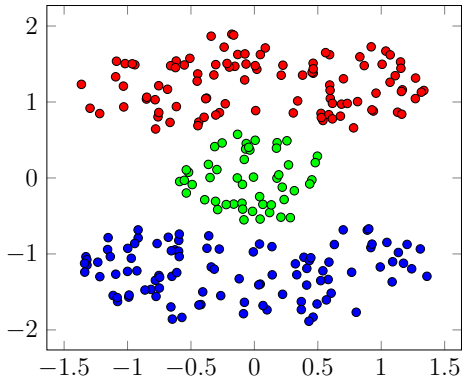


Figure 15: SyntheticCassini [24]

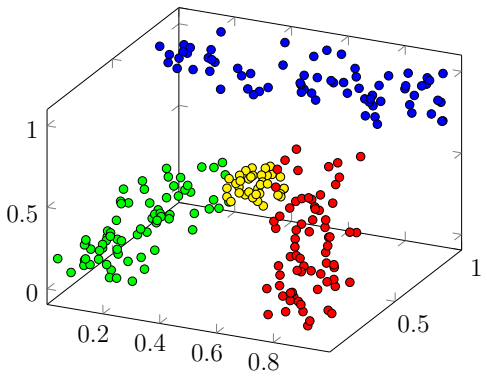


Figure 16: SyntheticCuboid [24]

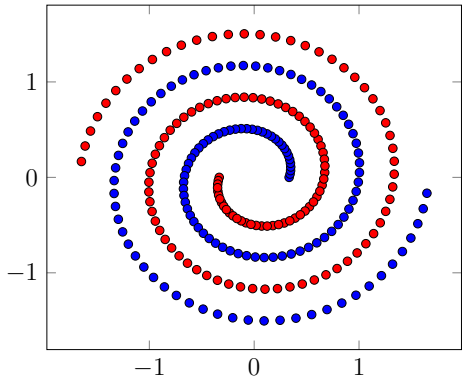


Figure 17: SyntheticSpirals [24]

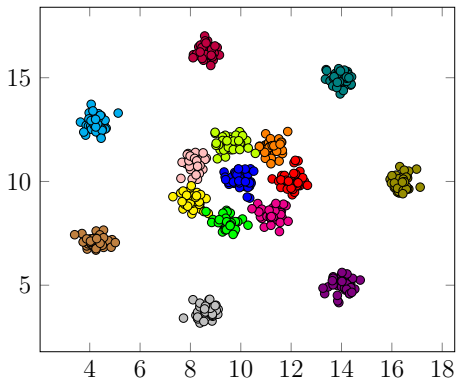


Figure 18: VeenmanR15 [20]

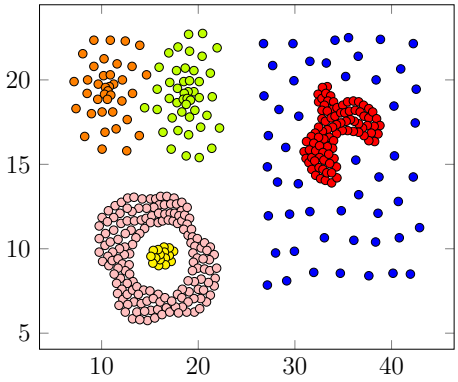


Figure 19: ZahnCompound [25]

B.2 Natural Data Sets

TCGA[18] is a data set that contains information about gene expression in three cancer types and combines three different properties into a combined similarity value for various molecules.

WordNet Data Sets The sets *ColiFind*, *ColiNeed* and *ColiState* are part of WordNet [19]. They analyze the occurrences of specific words in a body of text. By comparing the neighborhood of the words, similarities are calculated. It is then attempted to identify similar context or word meaning.

C Parameter Optimization Overview

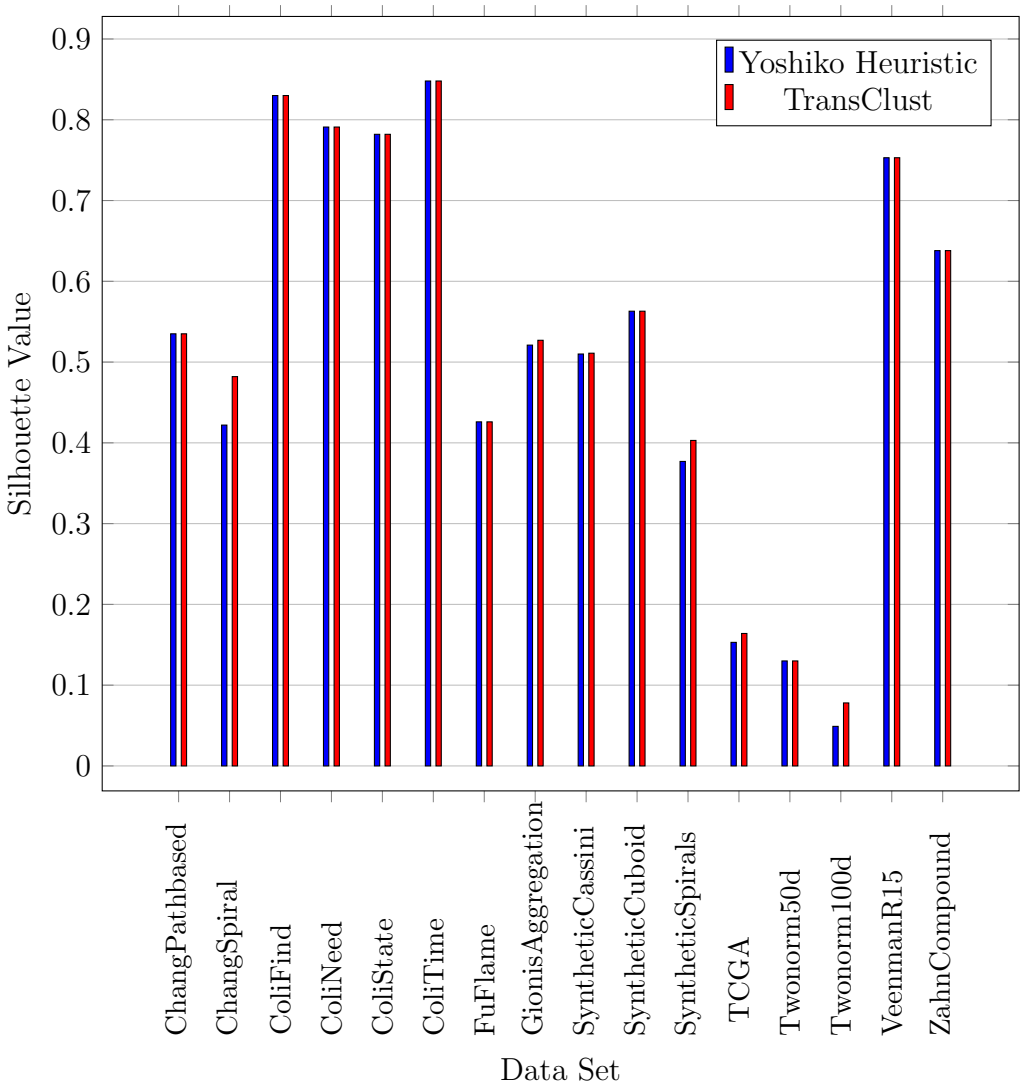
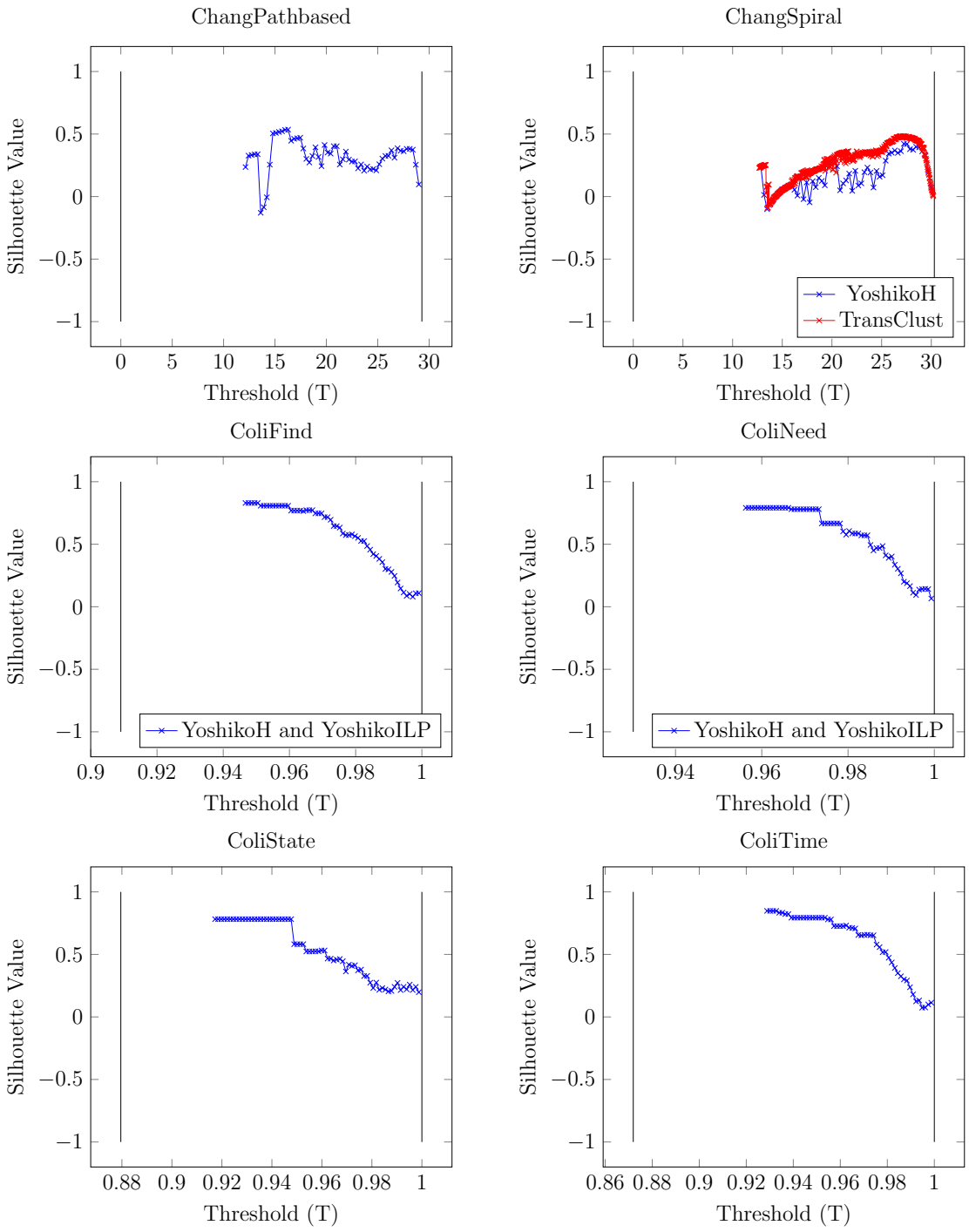
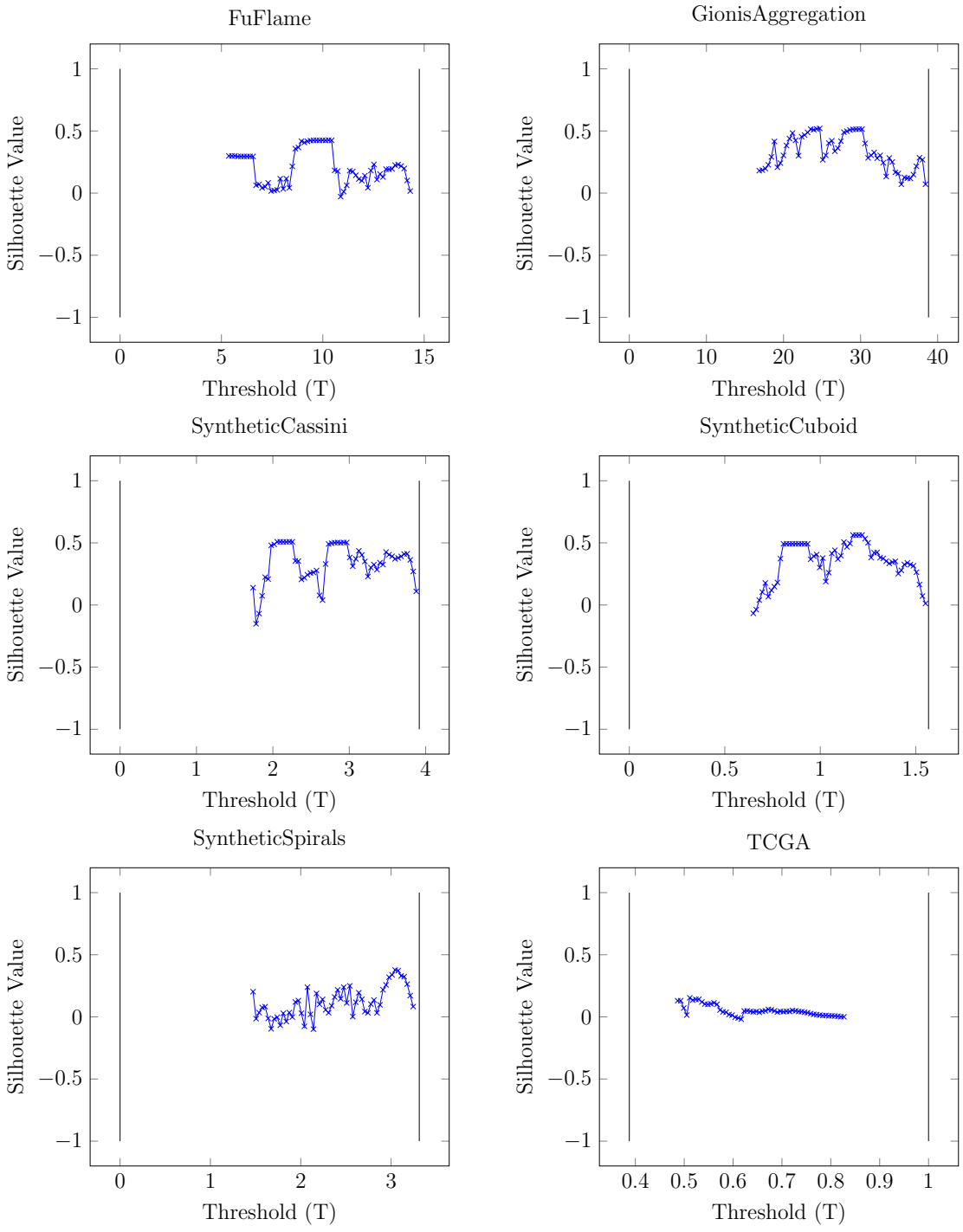
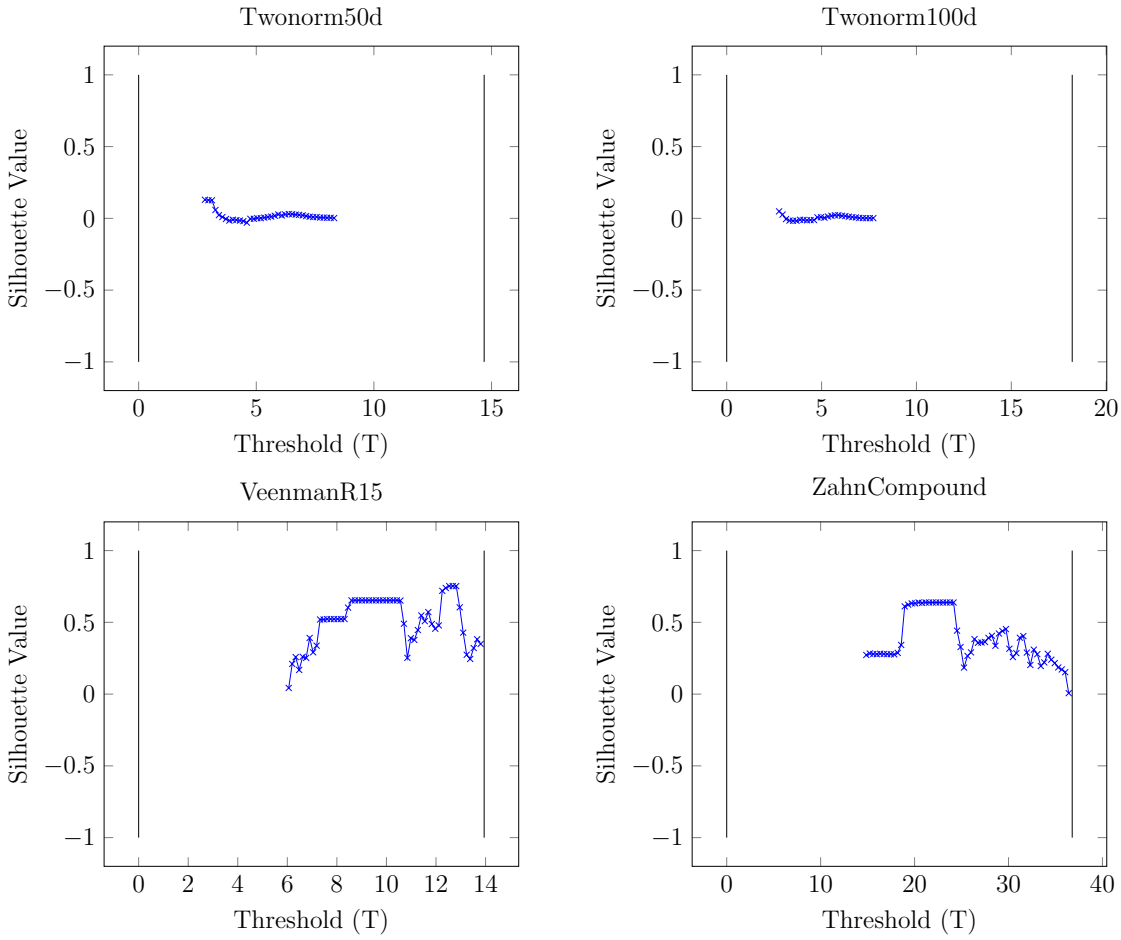


Figure 20: Optimized Parameters

D Parameter Optimization Curves




FuFlame
GionisAggregation
SyntheticCassini
SyntheticCuboid
SyntheticSpirals
TCGA
Silhouette Value
Threshold (T)



If not specified otherwise, the curves refer to Yoshiko in heuristic mode.

Figure 20: Visualization of the optimization process

E Running Time Analysis

E.1 Reduction Rules

Config / DataSet	None	CR	AC	CC	HE	PD	SN
ChangPathbased	3.099(0%)	3.120(0%)	3.126(0%)	3.129(0%)	3.522(0%)	5.977(0%)	34.565(2.333%)
ChangSpiral	0.564(0%)	0.581(0%)	0.643(0%)	0.568(0%)	0.436(1.608%)	0.981(0%)	2.815(0.892%)
ColiFind	6.281(0%)	6.167(0%)	6.202(0%)	6.298(0%)	7.302(0%)	10.109(99.523%)	467.804(5%)
ColiNeed	0.107(0%)	0.104(0%)	0.104(0%)	0.106(0%)	0.122(0%)	0.159(98.095%)	0.160(54.286%)
ColiState	0.552(0%)	0.552(0%)	0.551(0%)	0.549(0%)	0.659(0%)	0.881(98.947%)	9.936(22.105%)
ColiTime	12.117(0%)	11.548(0%)	11.706(0%)	11.72(0%)	14.128(0%)	19.575(99.412%)	1550.2(2.544%)
FuFlame	2.167(0%)	2.189(0%)	2.789(0%)	2.180(0%)	2.384(0%)	4.255(0%)	5.788(0%)
GionisAggregation	93.291(0%)	93.396(0%)	93.024(0%)	91.929(0%)	101.777(0%)	186.609(0%)	1045.78(0.003%)
SyntheticCassini	1.929(0%)	2.006(0%)	1.937(0%)	2.036(0%)	2.174(0%)	3.837(0%)	5.458(3.2%)
SyntheticCuboid	0.977(0%)	0.984(0%)	1.237(0%)	0.977(0%)	1.069(0%)	1.856(0%)	2.567(0%)
SyntheticSpirals	0.598(0%)	0.596(0%)	0.652(0%)	0.605(0%)	0.388(0%)	1.090(0%)	0.857(0%)
TCGA	2.097(0%)	2.170(0%)	2.109(0%)	2.135(0%)	2.495(0%)	4.100(0%)	2.313(0%)
Twonorm50d	0.626(0%)	0.624(0%)	0.631(0%)	0.624(0%)	0.753(0%)	1.145(10%)	5.107(0%)
Twonorm100d	0.657(0%)	0.635(0%)	0.637(0%)	0.639(0%)	0.755(0%)	1.203(2%)	1.794(0%)
VeenmanR15	5.881(0%)	5.926(0%)	7.088(0%)	5.882(0%)	6.494(0%)	11.362(45.5%)	23.098(45.833%)
ZahnCompound	6.602(0%)	6.578(0%)	6.625(0%)	6.571(0%)	7.613(0%)	12.920(0%)	387.413(0%)

CR = Clique Rule, **AC** = Almost Clique Rule, **CC** = Critical Clique Rule, **HE** = Heavy Edge Rule,
PD = Parameter-Dependent Rule,
SN = Similar Neighborhood Rule

All entries are in the format $t(r)$ where t is the running time in seconds and r the input reduction in %.
Note: We used no scaling factor for **SN**.

Table 1: Reduction rules running time and efficiency

E.2 Experimental Callbacks

Configuration / DataSet	None	TC	PC
ChangPathbased	28130	Timeout	27324
ChangSpiral	28	1	16
FuFlame	31831	14883	21167
SyntheticCassini	16926	Timeout	17585
SyntheticCuboid	141	6	138
SyntheticSpirals	5	0.737	4
TCGA	Timeout	52	Timeout
Twonorm50d	107	13	106
Twonorm100d	960	8	952
VeenmanR15	481	35	465
ZahnCompound	Timeout	2840	Timeout

TC = Triangle Cuts - callback
PC = Partition Cuts - callback

All entries describe the running time in seconds. For this experiment all reduction rules were turned on. The running time is calculated as total running time on all cores meaning that the actual clock time might be lower. A 'Timeout' entry means the actual running time exceeded at least 6 hours real-time. We omitted sets where all modes timed out as well as sets where the algorithm did not enter the ILP phase (as a solution was already found through reduction).

Table 2: Experimental callbacks running time

E.3 Comparison ILP, Heuristic and TransClust

Configuration / DataSet	TransClust	ILP	Heuristic
ChangPathbased	4.202	28130	3.099
ChangSpiral	3.385	28.254	0.781
ColiFind	6.238	11.659	6.281
ColiNeed	2.352	0.176	0.106
ColiState	2.997	0.993	0.552
ColiTime	8	22	12
FuFlame	3.500	31831	2.167
GionisAggregation	11	Timeout	93
SyntheticCassini	1.280	16926	1.929
SyntheticCuboid	1.319	141	0.977
SyntheticSpirals	2.781	5	0.598
TCGA	3.525	Timeout	2.097
Twonorm50d	3.080	108	0.626
Twonorm100d	3.683	960	0.657
VeenmanR15	4.321	481	5.881
ZahnCompound	4.732	Timeout	6.602

All entries describe the running time in seconds. For this experiment all reduction rules were turned on for the ILP and off for the heuristic. We measured TransClust using the unix command 'time' while we used the C++ 'clock()' command to measure Yoshiko. This was due to the fact that ClustEval did not support shared walltime monitoring yet. TransClust might be disadvantaged by this method as it was measured on its own, causing overhead for each measurement on the HPC system, whereas all Yoshiko measurements were taken in one run. A 'Timeout' entry means the actual running time exceeded at least six hours.

Table 3: Comparison of running time between TransClust, Yoshiko (ILP) and Yoshiko (Heuristic)