



DEGREE PROJECT IN COMPUTER SCIENCE 120 CREDITS, SECOND  
CYCLE

*STOCKHOLM, SWEDEN 2015*

# CluStic - Automatic graph drawing with clusters

VILLIAM ASPEGREN

KTH ROYAL INSTITUTE OF TECHNOLOGY

SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION



**KTH Datavetenskap  
och kommunikation**

## **Master of Science Thesis CSC**

**CluStic – Automatic graph drawing with  
clusters**

**CluStic – Automatisk grafritning med  
avseende på kluster**

Villiam Aspegren

Approved 2015-12-12	Examiner Johan Håstad	Supervisor Michael Minock
	Commissioner Decerno	Contact person Sven Norman

## ***Abstract***

Finding a visually pleasing layout from a set of vertices and edges is the goal of automatic graph drawing. A requirement that has been barely explored however, is that users would like to specify portions of their layouts that are not altered by such algorithms. For example the user may have put a lot of manual effort into fixing a portion of a large layout and, while they would like an automatic layout applied to most of the layout, they do not want their work undone on the portion they manually fixed earlier. CluStic, the system developed and evaluated in this thesis, provides this capability.

CluStic maintain the internal structure of a cluster by giving it priority over other elements in the graph. After high priority element has been positioned, non-priority vertices may be placed at the most appropriate remaining positions. Furthermore CluStic produces layouts which also maintain common aesthetic criteria: edge crossing minimization, layout height and edge straightening.

Our method in this thesis is to first conduct an initial exploration study where we cross compare four industrial tools: Cytogate, GraphDraw, Diagram.Net and GraphNet. A set of layouts are generated with these tools and the user is timed on a task to identify the longest path. Through this exploration study we develop out intuition and determined that Cytogate is the best performing tool for longest path identification.

Given this experience we fully develop CluStic and conduct our main study where we cross compare it with Cytogate and a baseline Breadth-first Search algorithm. Results show that CluStic produces drawings of good quality, Clustic achieves a visualization efficiency score of 1,4 which is an increase compared to the BFS layout (-3,8). CluStic is outperformed by Cytogate which achieves a visualization efficiency score of 1,9 and therefore produces less visually pleasing drawings. However Clustic, unlike Cytogate can preserve initial static structures, thus when a graph contains elements in which their position cannot be altered CluStic is a better choice.

## **Referat**

Målet med automatiserad grafitrning är att utifrån en uppsättning noder och kanter hitta en layout som är visuellt tillfredställande. Ett delområde som inte utforskats nog är möjligheten till att låsa vissa komponenter i grafen som sedan inte får alterneras av grafitrningsalgoritmen. En användare som exempel, strukturerar vissa delar av grafen manuellt och applicerar sedan automatisk layout av resterande element utan att förstöra den struktur som manuellt skapats. CluStic, grafitrningsverktyget som skapats och utvärderats i denna masters uppsats fyller denna funktion.

CluStic bevarar den interna strukturen för ett kluster genom att tilldela en högre prioritet för noder i klustret med avseende på övriga element i grafen. Efter att högprioritets element placerats tilldelas resterande element sina bäst tillgängliga positioner. Utöver detta så uppfyller CluStic några av de vanligaste estetiska mål inom grafitrning: minimera antalet kantkorsningar, minimera höjden, och räta ut kanter.

Metoden som används i denna master uppsatts var att först gör en inledande studie där vi undersöker fyra populära grafitrnings verktyg: Cytogate, GraphDraw, Diagram.Net och GraphNet. En uppsättning grafer genereras av dessa verktyg och vi mäter hur lång tid det tar för en användare att hitta den längsta vägen i grafen. Genom denna studie konstaterar vi att Cytogate presenterade grafer med best kvalitet.

Från kunskap samlad i den inledande studien utvecklar vi CluStic och utför uppsatsens huvud studie där vi jämför CluStic med avseende på Cytogate och en bas layout Bredden-först algoritm. CluStic uppnår ett visualiserings effektivitetsvärde på 1,4 vilket är en ökning jämfört Bredden-först algoritmen (-3,8). CluStic levererar inte layouter som är mer visuellt tillfredställande än de som skapats av Cytogate som får ett visualiserings effektivitetsvärde på 1,9. CluStic tillskillnad från Cytogate bevarar den internt fixa strukturen mellan element med hög prioritet vilket gör CluStic till det bättre verktyget för grafer med statiska element.



# FOREWORD

---

To my family, thank you for being there when times are rough, for the support, and happy moments you provide.

To my supervisor Michael, kept me going when I thought there was no time.

Villiam Aspegren

Pallet town, June 2015

# NOMENCLATURE

---

## Notations

Symbol	Description
$G$	Graph
$V$	Vertices in graph
$E$	Edges in graph
$\delta(v)$	Neighboring edges to $v$

## Abbreviations

DAG	Directed acyclic graph
OSCM	One-sided crossing minimization
BFS	Breadth first search
DFS	Depth first search

# TABLE OF CONTENTS

---

1	INTRODUCTION .....	1
1.1	Problem statement .....	2
1.2	Thesis outline .....	2
2	BACKGROUND .....	3
2.1	General graph drawing .....	3
2.1.1	Layered graph drawing .....	4
2.1.2	Forced directed graph drawing .....	5
2.1.3	Orthogonal graph drawing .....	6
2.2	Aesthetics in graph drawing .....	7
2.3	Layered graph drawing .....	9
2.3.1	Cycle removal .....	9
2.3.2	Layer assignment .....	10
2.3.3	Crossing reduction .....	13
2.3.4	X-coordinate re-assignment .....	14
2.4	State of the art graph drawing tools .....	15
3	CLUSTIC .....	17
3.1	CluStic – preliminary constraints .....	17
3.1.2	Self-loop removal .....	18
3.1.3	Two-loop removal .....	18
3.2	Cycle removal .....	20
3.3	Vertex layering .....	21
3.3.1	Static vertex layering .....	22
3.3.2	Cluster layering .....	22
3.4	Edge crossing minimization .....	24
3.4.1	Edge crossing minimization for static vertices .....	25
3.4.2	Clustered crossing minimization .....	25
3.5	X-coordinate assignment .....	27
4	EVALUATION .....	29
4.1	Initial exploration study .....	29
4.1.1	Input data .....	29
4.1.2	Task .....	29
4.1.3	Evaluation program .....	29
4.1.4	Evaluation .....	30



4.2	Interview.....	30
4.3	Main study.....	30
4.3.1	Input data.....	30
4.3.2	The Task.....	31
4.3.3	Evaluation program .....	31
4.3.4	Evaluation.....	31
5	RESULTS.....	33
5.1	Initial exploration study.....	33
5.2	Main study.....	34
6	DISCUSSION AND CONCLUSION .....	37
6.1	Discussion .....	37
6.1.1	Initial exploration study.....	37
6.1.2	The implementation.....	37
6.1.3	Main study.....	38
6.2	Conclusion.....	39
6.2.1	Initial and main study .....	39
6.2.2	The implementation.....	40
6.3	Recommendations .....	40
7	REFERENCES .....	41
	APPENDIX A: DIAGRAM.NET .....	44
	APPENDIX B: GRAPHWIZ .....	48
	APPENDIX C: GRAPHNET .....	52
	APPENDIX D: CYTOGATE.....	56
	APPENDIX E: CLUSTIC .....	60
	APPENDIX F: BFS.....	64

# 1 INTRODUCTION

The rapid growth of data in modern systems has led to increased system complexity leaving companies with issues of too large and cluttered system architectures. Graph drawing in computer science centers around; class diagrams, interaction diagrams, flow charts, database models, etc. Good drawings could help decrease learning curves while saving resources for companies.

From a set of vertices and edges, automatic graph drawing deals with the problem of finding a visibly pleasing layout. Throughout the years many different approaches to automatic graph drawing have been proposed: (1) orthogonal, (2) force directed and (3) layered. Orthogonal and force directed techniques focus on undirected graphs while layered drawing focuses on directed graphs. Today layered graph drawing algorithms aim to find layouts which are visibly pleasing using relations between vertices in the graph. The term “visibly pleasing” is highly subjective leading to different opinions on what contributes to a good layout. Commonly discussed constraints also known as *aesthetic criteria* are proven to increase the quality of layouts. But not all aesthetic criteria can be satisfied, some counteract others leading into impossible results and bad drawings. The outcome of the final drawing is affected by a combination of the drawing technique and which aesthetic criterion one wishes to fulfill.

Constructing manual layouts require lots of effort and time, particularly maintaining layouts after updates is demanding. Furthermore “the human factor” always plays its part, we tend to forget about details leaving drawings in inconsistent states. This might be tolerable when layouts are only used by a small number of people, but as soon as that number increases misunderstandings arise. Automatic graph drawing solves this problem by creating up-to-date layouts from specification within a reasonable time frame.

A **cluster** is a sub-graph of vertices which poses an internal structure which is always maintained with regards to other components outside the cluster. This means that no vertex within a cluster can be moved from its original position in the cluster. Examples on clusters are: vertices with short distances to each other, strongly connected vertices, etc. In many areas of application clusters appear on a regular basis, drawing tools should therefore be able to handle such components. A cluster containing two vertices which is connected with the rest of the graph by at most one edge is in the thesis referred to as a **static vertex**. These small clusters are treated differently in the partial algorithms used by CluStic due to them being sinks on their own, examples of **static vertices** are fault nodes in a flowchart.

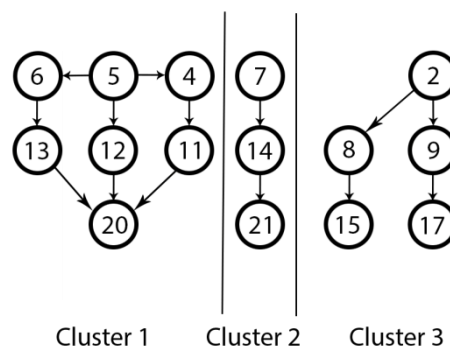


Figure 1 - Illustration of a possible clusters in a graph. Cluster 1 contains three sub-processes (4, 6, and 12) which originate from vertex 5. Vertex 6 and 4 are statically positioned on the left and right of vertex 5, and move in a direction other than the flow of the graph.

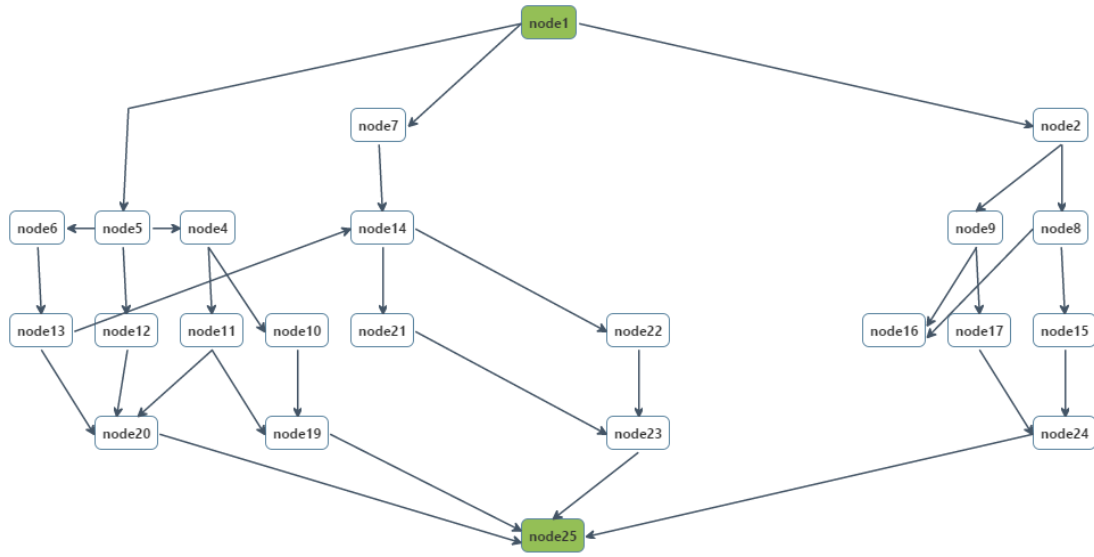


Figure 2 – Layout constructed by CluStic of a graph where clusters from figure 1 are maintained with regards to non-cluster vertices. On the left we see cluster 1, in the middle-left cluster 2 and on the right cluster 3. As shown by the layout, the internal structure of each cluster is kept and non-cluster vertices are placed around clusters.

## 1.1 Problem statement

Given a directed graph structure consisting of at least one cluster or static vertex, the goal is to find a layout of elements in the graph that appears visually pleasing to an interpreter while preserving the internal structure of clusters and static vertices.

## 1.2 Thesis outline

In chapter 2 we review literature references related to layered graph drawing and aesthetic criterion. We further reflect upon problems with current implementations and present the reader with background theory. In chapter 3 we present our method which is then evaluated in chapter 4. In chapter 5, we present the results of the evaluation. In chapter 6, we discuss results and the implementation, furthermore we present interesting extensions to our method which could further improve our work.

## 2 BACKGROUND

### 2.1 General graph drawing

Graph drawing dates back to the 13th century and the work of Ramon Llull. In his work he drew complete graphs in order to analyze pair wise combinations of values (Wilson, Graham, & Watkins, 2013). Modern graph drawing usually involves computers, more precise the task of automatically draw graphs by the use of certain algorithms. The introduction of automatic graph drawing increased the limit on what was possible. No longer did a large number of vertices and edges pose a problem. Today one popular area of graph drawing, often demanding large datasets is the analysis of social networks which is used to produce *Sociograms* (Zhao, 2008). Other areas of application are entity diagrams (UML and database structures), data flow diagrams and flow charts. Usage outside of computer science usually concerns biology and chemistry where layout of molecular maps, protein functions and evolutionary trees are popular (Herman, Melancon, & Marshall, 2000).

A graph drawing problem is simply put: Given a set of nodes and a set of links (edges between nodes) decide on the position of nodes and the appropriate drawing of the links. Usually in the constructed node-link diagram nodes (vertices) are represented by circles, squares or textual boxes whereas links (edges) are represented by straight lines, poly-lines or arcs (Herman, Melancon, & Marshall, 2000). Vertices and edges can be placed in many different positions while still satisfying the graph structure. Each new placement of a vertex/edge corresponds to a new solution of the graph drawing problem. The location where vertices and edges can be placed depends strictly on which type of graph the solution is being applied to.

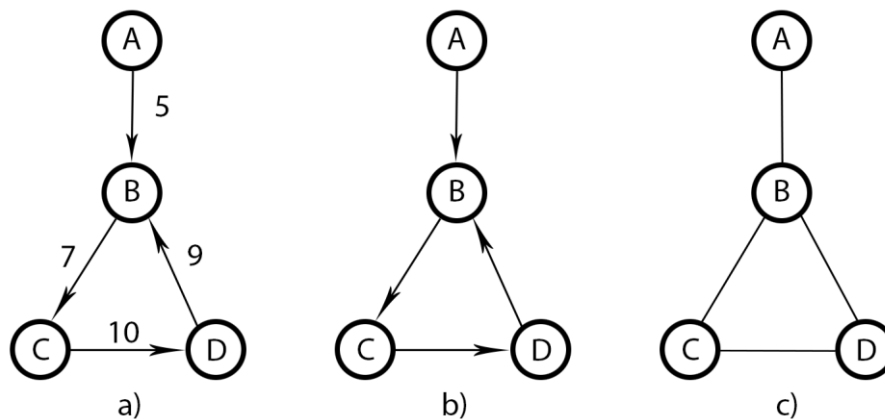


Figure 3 - Illustration of different types of graphs, a) Weighted b) Directed c) Undirected.

Figure 3 shows the most commonly used graph types, each type can be either cyclic or acyclic. In a cyclic graph it is possible to construct a tour of the graph where the start and finish vertex is the same. An acyclic graph contains no cycles and it is therefore not possible to start and end in the same vertex. As an example, figure 3b can be turned into an acyclic graph by reversing the edge between vertex B and D.

There exists no universal drawing technique, which one is of use depends on the graph type and personal taste. Some techniques are more useful for certain types of graphs; *Layered graph drawing* (Sugiyama, Tagawa, & Toda, 1981) is usually used for *directed acyclic graphs (DAG)*. Undirected graphs are drawn using *Force directed graph drawing* (Kobourov, 2012) or *Orthogonal graph drawing* (Eiglsperger, Fekete, & Klau).

### 2.1.1 Layered graph drawing

In the late seventies (Warfield, 1977) and in the early eighties (Carpano, 1980) started the development of layered graph drawing algorithms. Only one year after, the most popular layered algorithm (Sugiyama, Tagawa, & Toda, 1981) was introduced. Layered graph drawing assumes that a general flow exists within the graph, meaning most edges point in the same direction after they have been layered. The direction is usually left-to-right, or top-down, in the end it all depends on the application domain. Layered graph drawing focuses on 4 sub-problems; Cycle removal, layer assignment, crossing reduction and x-coordinate re-assignment, each sub-problem addresses one criteria which affect the quality of the drawing (Bastert & Matuszewski, 2001). Example of systems using Sugiyama scheme are; *dot* (Gansner, Koutsofios, & North, 2006), *AGD graph drawing library* (Paulisch & Tichy, 1990) and *da Vinci* (Fröhlich & Werner, 1995).

Figure 4 illustrates different steps of the Sugiyama scheme. At step one (Figure 4b) of the algorithm directed cycles are removed by reversing a subset of the edges. After the first step the graph has been converted into a *DAG*, a *DAG* allows for all edges to be uniformly directed. The *DAG* is then layered (Figure 4c), each vertex is assigned to a layer so that all edges point in the same direction. Many algorithms used in step three require a *proper leveling* (edges only occur between adjacent layers). By permutation of vertices in each layer the number of edge crossings are minimized (Figure 4d). Finally the x-coordinate of vertices in long edges (spanning more than 1 layer) are adjusted to straighten edges (Figure 4e).

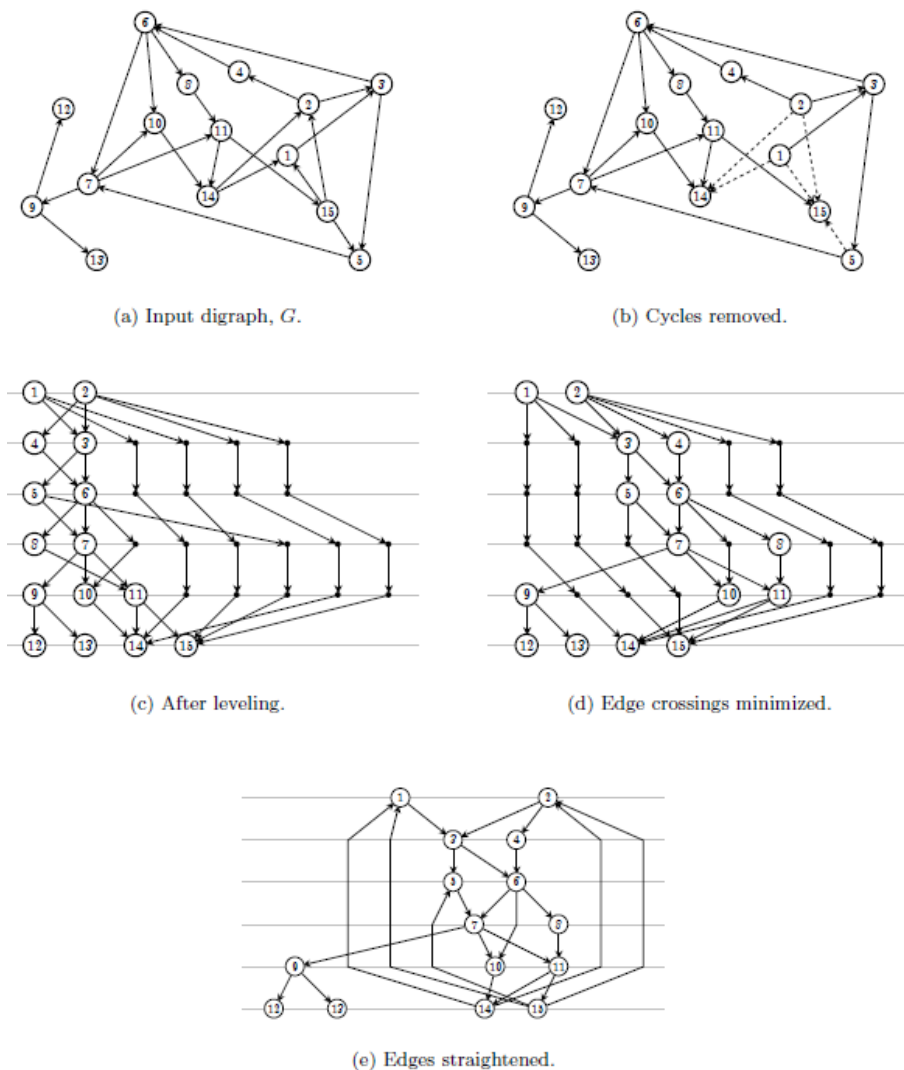


Figure 4 - Illustration of Sugiyama styled drawing (Healy & Nikolov, 2013)

As understood by Figure 4 the Sugiyama scheme preserves the hierarchy of a graph, a hierarchy which is explained by layers of vertices. A more formal approach to explain hierarchy is given in definition 1.2 following from Definition 1.1 (Healy & Nikolov, 2013).

---

**Definition 1.1**

---

A leveled graph  $G = (V, E, \gamma)$  is a DAG with mapping  $\gamma : V \rightarrow \{1, 2, \dots, k\}, k \geq 1$ , that partitions a vertex set  $V$  as  $V = V_1 \cup V_2 \cup \dots \cup V_k, V_j = \gamma^{-1}(j), V_i \cap V_j = \emptyset$  for  $i \neq j$ , such that  $\gamma(v) = \gamma(u) + 1$  for each edge  $(u, v) \in E$

---



---

**Definition 1.2**

---

A hierarchy is a level graph  $G(V, E, \gamma)$  where for every  $v \in V_j, j > 1$ , there exists at least one edge  $(w, v)$  such that  $w \in V_{j-1}$

---

Even though Sugiyama scheme is widely used, some drawbacks still exists. Since edge crossing reduction and layer assignment are independent steps, the final drawing may end up with unnecessary edge crossings due to unfortunate layer assignment. Chimani et al. proposed solution to the unfortunate layer assignment problem in their *Upward planarization layout* (Chimani, Gutwenger, Mutzel, & Wong, 2011). Another problem related to layering is the individual height of vertices. A vertex can stretch further than the height of a layer causing non uniform layering leading to blurring of the final drawing.

### 2.1.2 Forced directed graph drawing

Some of the most used techniques for drawing undirected graphs belong to the class of force directed algorithms. This method of drawing, also known as *Spring Embedders* is generally divided into two different categories: *vertex-to-vertex* repulsion and *vertex-to-edge* repulsion (Lin & Yenb, 2012). Eades presented a vertex-to-vertex model which seeks a solution by illustrating a graph as a mechanical system. Each vertex is replaced by steel ring and each edge is replaced by a spring, hence the name *Spring Embedders*. Vertices are initially placed in random positions, when they are released the mechanical system moves to a minimal energy state due to spring forces. An effective modification to Eades mechanical system was introduced by Fruchternab and Reingold. In similarity to Eades, Fruchternab and Reingold uses logarithmic springs due to the overwhelming force of *Hooke's Law springs* for vertices with large distance between each other (Fruchterman & Reingold, 1991). Vertex-to-vertex models often places vertices very close to edges making the resulting drawing unreadable. Vertex-to-edge models therefore not only consider a repulsive force between vertex-vertex and an attractive force between adjacent vertices, but also a repulsive force between vertex-edge.

Whilst force directed algorithms produce drawings which are intuitive, easy to understand and are visually pleasing, they suffer from two major drawbacks. At first, most algorithms run in  $O(n^3)$  where  $n$  is the number of vertices in the graph. Normally the number of iterations in force directed algorithms is  $O(n)$  and in every iteration the force of each pair of vertices is calculated, thus  $O(n^3)$ . Based on running time and aesthetic criteria, force directed algorithms can only handle around 50 vertices. Hadany and Harel addresses the large graph problem (number of vertices larger than 100) by introducing the *multi-scale* technique (Hadany & Harel, 2001). They divide the calculation of vertex positions into coarse-scale (global positions) and fine-scale (local positions in cluster), the algorithm then moves back and forth between these calculations. Traditional force directed algorithms (Eades P., 1984) and (Fruchterman & Reingold, 1991) often run into the problem of local minima and therefore fail to produce a feasible drawing. *Multi-scale*, or *multi-layer* as it is sometimes called, not only reduces running time but also removes the problem of local minima.

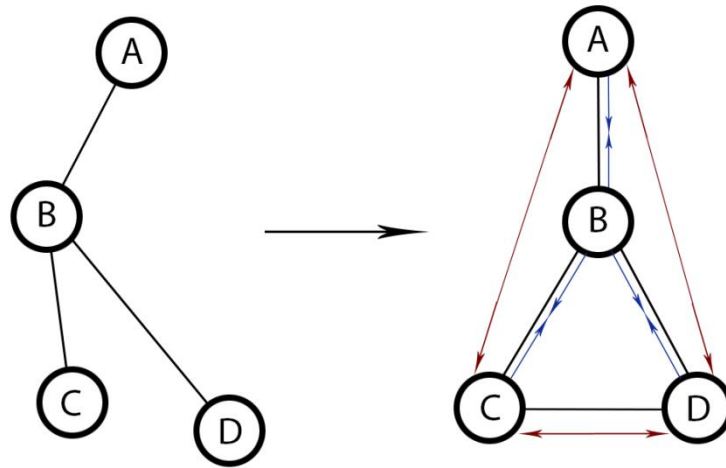


Figure 5 - Initial graph (left) illustrated using a force directed approach (right)

### 2.1.3 Orthogonal graph drawing

*Orthogonal drawing* map edges along rows and columns of an underlying grid. The thought behind orthogonal drawing is that drawings which are clustered with edge- crossings and vertices with low angular resolution are hard to comprehend. Placing vertices at intersections between rows and columns guarantees maximal distinctiveness of adjacent edges at angle multiples of  $\frac{\pi}{2}$ . Thus the maximum degree of a vertex in the graph will be 4, meaning an angular resolution of 90 degree (Biedl, Madden, & Tollis, 1997). Maximum angular resolution introduces bends, therefore bend minimization is a key problem in orthogonal graph drawing. Early algorithms constructed graphs with few bends at the cost of drawing area and running time. However, even for planar graphs (drawings without crossings) it is NP-complete to decide whether a graph has an orthogonal drawing without bends (Eiglsperger, Fekete, & Klau, Orthogonal graph drawing, 2001). In 1987 Tamassia presented his flow network approach which produced drawings with a minimum number of bends and drawing area (Tamassia, 1987). The flow network approach comes with a near quadratic running time, this was later lowered in the algorithm by (Garg & Tamassia, 1996). Due to the time complexity of flow network algorithms, improved heuristics which run in linear time have been proposed. Tamassia and Tollis presented a heuristic which uses a constraint visibility representation and produce drawings with at most  $2n + 4$  bends in linear time. The above solutions focus on 4 graphs, i.e. graphs where vertices have a maximum degree of four. As for graphs with higher degree Tamassia proposed a solution where a vertex with degree larger than four is split into multiple segments. Segments are contained within a virtual box which represents the original vertex, above 4-graph algorithms can then be applied within the virtual box (Tamassia, 1987). Virtual box strategy implies a larger drawing since the area of a virtual box is often larger relative to the degree of a non segmented vertex.

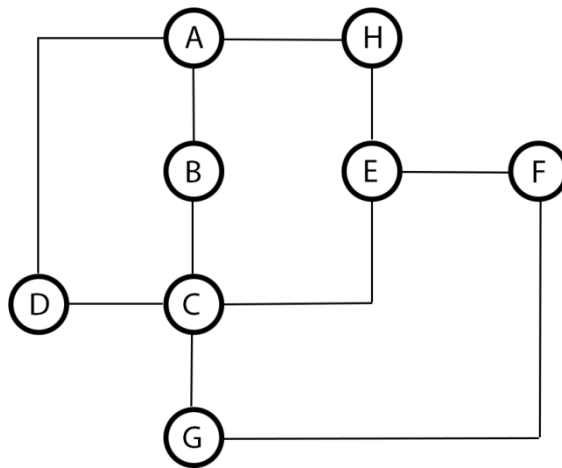


Figure 6 - Orthogonal graph

## 2.2 Aesthetics in graph drawing

In computer science graphs are used to model systems structure, interactions between components and the overall flow of data, focusing on providing an overview, helping with the comprehensiveness of a given system. Comprehensiveness is strongly subjective and what humans understand when given the same task differs. A cognitive model of a task is determined by objective questions, where inference of subjective statements is minimized (Ware, Purchase, Colpoys, & McGill, 2002). In graph drawing a popular model is to measure the accuracy and time of a task, e.g. “The shortest path between two nodes”. The cognitive model is not only about constructing a task but should also consider the complexity of the task, due to limitations of the human mind. Consider the question “Shortest path between two nodes?” The human mind can count up to three objects at a glance (Dehaene, 2011) meaning the path must be longer than three edges otherwise the task provides no measurements (Ware, Purchase, Colpoys, & McGill, 2002). On the other hand, if the number of possible paths is too large it becomes hard to remember which paths have been visited. The complexity of the cognitive model should be an assumption of what *mental effort* (Huang, Eades, & Hong, 2006) a subject put into solving a given task. Given all this the cognitive model provides a solution *how to* measure the quality of a graph, *what effects* quality is answered by the aesthetic criteria. The more common criteria are edge crossings, number of bends, drawing area, edge length, crossing angles, symmetry and even distribution of vertices (Huang W. , Eades, Hong, & Lin, 2013) (Ware, Purchase, Colpoys, & McGill, 2002) (Herman, Melancon, & Marshall, 2000).

Empirical studies on aesthetic criteria in graph drawing are mostly done by Purchase. In *Validating graph drawing aesthetics* (Purchase, Cohen, & James, 1996) she and her colleagues investigated three of the most common aesthetics: edge crossings, number of bends and symmetry. Their experiment follows the cognitive model outlined above, further a graph was drawn three times for each aesthetic criteria where each drawing had more or less focus on eliminating the given criteria. Each subject performed the same task on each drawing measuring the accuracy and response. Their results show an increase in error rate when the number of edge crossings is large. In *Cognitive measurements of graph aesthetics* Purchase et al. once again prove the negative effect on readability given by a large number of edge crossings. Purchase et al furthermore investigates the effect of *path continuity (bendiness)* on readability. Each edge crossing contributes 0.65 seconds to the final time which corresponds to approximately 38 degrees of bendiness (Ware, Purchase, Colpoys, & McGill, 2002). Their results prove it is worth allowing for some extra edge crossings as long as it reduces the bendiness of paths. Purchase et al. also show that it is in fact the number of crossings in the task (the shortest path) of the cognitive model which decreases readability. This has no effect on the overall results since their algorithm does not consider a special path when constructing graphs.



As mentioned above, paths have been a widely used objective measurement in graph drawing studies, shortest path (Ware, Purchase, Colpoys, & McGill, 2002) and (Ware & Mitchell, 2008) and (Huang W. , Eades, Hong, & Lin, 2013) in particular is popular. Shortest path is convenient in studies about force directed drawings where the flow of a graph is non uniform or non-existent. For layered graph drawing the flow mostly moves in one direction making it easier for humans to track short paths instantly. Therefore the path must not only be of length greater than three but must be so substantially (Dehaene, 2011).

Due to the complexity of aesthetic search problems it is not possible to consider all available aesthetics when constructing a graph drawing algorithm. The complexity issue is somewhat solved by heuristics, which takes the aesthetic criteria in consideration but gives no guarantee on optimization. Many aesthetics criteria are mutually exclusive (Huang W. , Eades, Hong, & Lin, 2013), solving one will contradict the other resulting in non feasible graphs. Consider figure 7, maximizing symmetry (left) results in more edge crossings. Minimizing edge crossings (right) reduces symmetry.

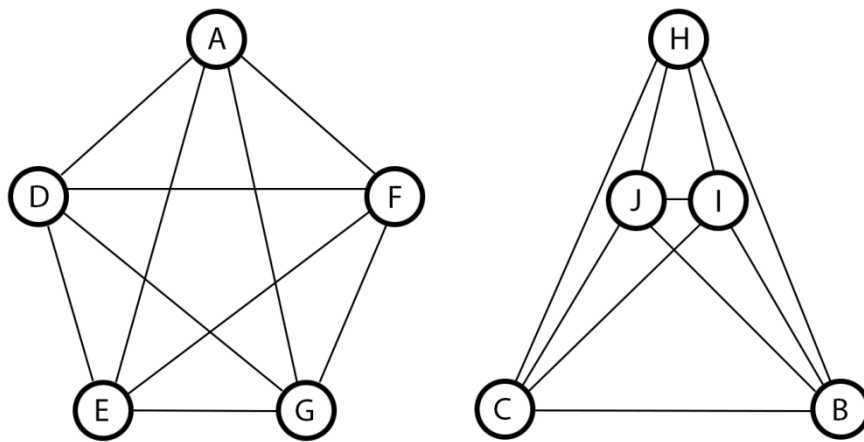


Figure 7 - Drawing of same graph. Left: Maximized on symmetry; right: minimum number of bends

## 2.3 Layered graph drawing

The aim of this thesis is an algorithm which constructs a layered drawing of a given graph based on existing clusters. We therefore research and present existing heuristics and algorithms used to solve partial problems of layered graph drawing.

### 2.3.1 Cycle removal

Cycle removal is the first step of the Sugiyama method, cycle removal aims to reverse a subset of edges turning the original digraph (directed graph) into a *DAG* (*directed acyclic graph*). In Sugiyama method no *two-cycles* exist, *two-cycles* consist of a pair of edges  $(u,v)$  and  $(v,u)$  which forms a cycle between two nodes. If *two-cycles* are present, one of the edges is removed before continuation. Edges which are reversed to create a *DAG* will have a direction against the flow in the final drawing. Reversal of edges is not the only option, edges can also be removed and reintroduced. However, removing edges leaves a sub-graph of the original *DAG* this may lead to undesirable results in step 2 of the Sugiyama model.

The problem of finding a minimum set  $E_a \subset E$  (FAS set) of edges which removal turns a digraph  $G(V, E_a)$  into a *DAG* is often stated as the *Minimum feedback arch set problem* (*minimum FAS*). The minimum *FAS* problem is unfortunately known to be NP-hard (Karp, 1972). Below we present popular heuristics for solving the minimum *FAS* problem, some of these heuristics have been proposed to also solve the complimentary problem of *FAS*, *The maximum acyclic sub-graph problem*.

#### 2.3.1.1 Vertex ordering

The *un-weighted linear ordering problem* is equivalent to the maximum acyclic sub-graph problem (Bastert & Matuszewski, 2001). The un-weighted linear ordering problem is defined as: find an ordering of all nodes in a graph  $G(V, E)$  i.e. a mapping  $o : V \rightarrow \{1, 2, \dots, |V|\}$  such that the number of edges  $(u, v) \in E : o(u) > o(v)$  is minimized. From this definition it follows that the easiest heuristic approach would be to take any ordering of  $G$  and remove edges  $(u, v)$  where  $o(u) > o(v)$ . The ordering might be constructed from a breath-first search, a depth-first search or as Eades et al. suggested, from an approach where the node with largest out degree are ordered first and then the node with second largest out degree and so on (Eades, Lin, & Smyth, 1989). These types of heuristics are fast but do not give any quality guarantees.

#### 2.3.1.2 Berger and Shor

In 1990 Berger and Shor presented their algorithm which guarantees an acyclic set of size at least  $\frac{1}{2}|E|$  (Healy & Nikolov, 2013). Imagine a graph  $(V, E)$ , let  $\delta^+(v)$  and  $\delta^-(v)$  denote the sets of outgoing and incoming edges of a node  $v$ , and  $\delta(v) = \delta^+(v) \cup \delta^-(v)$  the set of incident edges to  $v : v \in V$ .

---

**Algorithm 1: Berger – Shor algorithm**

---

```
 $S_t \leftarrow \emptyset$ 
foreach  $v \in V$  do
  if  $|\delta^+(v)| \geq |\delta^-(v)|$  then
    appends  $\delta^+(v)$  to  $E_a$ ;
  else
    append  $\delta^-(v)$  to  $E_a$ ;
  delete  $\delta(v)$  from  $G$ ;
```

---

Algorithm 1 computes the acyclic set  $E_a$  with size  $|E_a| \geq \frac{1}{2}|E|$  and runs with a time complexity  $O(|V| + |E|)$ . The sub-graph  $G'(V, E_a)$  is a *DAG* and thus a set  $F = E \setminus E_a$  is a *FAS* (Healy & Nikolov, 2013).

### 2.3.1.3 Greedy cycle removal

In 1993 Eades et al. observed the fact that edges incident to sinks or sources of a digraph could not be part of a cycle. By making this observation Eades et al. constructed algorithm GR (Eades, Lin, & Smyth, 1993) which improved the results of Berger and Shor (Berger & Dhor, 1990).

---

#### Algorithm 2: GR – Enhanced greedy algorithm

---

```

 $E_a \leftarrow \emptyset$ 
 $G \leftarrow (V, E)$ 
while  $V \neq \emptyset$  do
    while  $v \in V$  and  $v$  is a sink in  $G$  do
        appends  $\delta^-(v)$  to  $E_a$  and delete  $v$  and  $\delta^-(v)$  from  $G$ ;
        remove all isolated vertices in  $G$ ;
    while  $v \in V$  and  $v$  is a source in  $G$ 
        appends  $\delta^+(v)$  to  $E_a$  and delete  $v$  and  $\delta^+(v)$  from  $G$ ;
    if  $V \neq \emptyset$  then
        let  $v$  be a vertex in  $V$  with maximum value  $|\delta^+(v)| - |\delta^-(v)|$ ;
        append  $\delta^+(v)$  to  $E_a$  and delete  $v$  and  $\delta^+(v)$  from  $G$ ;

```

---

As done by Berger and Shor, the algorithm by Eades et al. removes edges after a vertex is processed. However, algorithm GR builds the *FAS* by computing a linear ordering of vertices in the digraph and takes those edges with direction against the flow/ ordering as the *FAS*. The algorithm computes a  $FAS \leq \frac{|E|}{2} - \frac{|V|}{6}$ . Algorithm 2 has been altered to directly compute the acyclic set  $E_a$  leaving the resulting  $FAS = E \setminus E_a$ .

### 2.3.2 Layer assignment

A layering of a DAG  $G = (V, E)$  is a partitioning  $\mathbb{L}$  of the vertex set  $V$  where  $\mathbb{L} = \{L_1, L_2, \dots, L_h\}$ ,  $h \geq 1$  such that  $\forall (u, v) \in E$  with  $u \in L_j$  and  $v \in L_i$  then  $i < j$ . The separate sets  $\{L_1, L_2, \dots, L_h\}$  of  $\mathbb{L}$  are called the layers. The height of a layering is determined by the number of existing layers whereas the width is determined by the number of vertices in a specific layer. Let a function  $l(u, \mathbb{L})$  denote the number  $i$  of the layer containing vertex  $u \in V$ . The span  $s$  of an edge  $(u, v)$  is the height distance between the vertices, i.e.  $s(e, \mathbb{L}) = l(u, \mathbb{L}) - l(v, \mathbb{L})$ . A layering of  $G$  is called proper if there is no edge  $e \in E$  with  $s(e, \mathbb{L}) > 1$ . Finding a proper layering of  $G$  is crucial since the crossing reduction step assumes a proper layered DAG as input. In the algorithms below we assume a proper layer, meaning that an edge  $(u, v)$  with a span larger than 1 will be replaced by a path  $(u = v_1, v_2, \dots, v_k = v)$  where a dummy vertex is placed in each layer between  $l(u, \mathbb{L})$  and  $l(v, \mathbb{L})$ . The running time of algorithms in the later parts of the Sugiyama method depends on the number of original vertices plus the number of dummy vertices, hence minimizing the number of dummy vertices reduces time complexity (Bastert & Matuszewski, 2001). Running time is not the only reason for dummy vertex minimization, bends are introduced at the position of dummy vertices in the final drawing. A large number of dummy vertices also mean a large number of long edges. As explained in section 1.2 long edges and the number of bends are both aesthetics which affects the readability of the drawing.

#### 2.3.2.1 The longest-path algorithm

A layering that minimizes the height with no regards on width can be found in linear time (Mehlhorn, 1984). The algorithm starts out with two empty sets,  $U$  and  $Z$ , and a variable called *currentLayer* which represent the current layer being built. As soon as a vertex is assigned to a layer it is added to  $U$ , meaning  $U$  is a set containing all assigned vertices. After all nodes of a layer have been assigned the set  $U$  is unified with set  $Z$ , meaning  $Z$  contains all vertices in layers below. All sinks are assigned to layer zero, a new vertex  $v$  is picked based on the following criterion: The vertex cannot have been assigned to a previous layer  $v \in V \setminus U$ , and all immediate successors must have been assigned to layers below  $currentLayer \cap N_G^+(v) \subseteq Z$ .

---

**Algorithm 3: The longest path algorithm**

---

```
DAG  $G \leftarrow (V, E)$ 
 $U \leftarrow \emptyset$ 
 $Z \leftarrow \emptyset$ 
 $currentLayer \leftarrow 1$ 
while  $U \neq V$  do
  select  $v \in V$  with  $N_G^+(v) \subseteq Z$ ;
  if a vertex  $v$  is selected;
    Assign  $v$  to the layer with number  $currentLayer$ ;
     $U \leftarrow U \cup \{v\}$ 
  if no vertex is selected
     $currentLayer \leftarrow currentLayer + 1$ ;
     $Z \leftarrow Z \cup U$ 
```

---

### 2.3.2.2 Coffman-Graham algorithm

Finding a layering which is neither too wide nor too high is NP-hard (Eades & Lin, 1989) even if all original vertices have the same width and the contribution of dummy vertices is left out. Consider the possibility that each vertex in a *DAG* represents a task to be performed on one processor in a multiprocessor. Suppose each task requires the same amount of time, i.e. 1 unit of time. Edges in the *DAG* are precedence constraints between tasks. The *precedence-constraint multiprocessor scheduling problem (PCMS)* is the problem of assigning all tasks to  $w$  processors so that all tasks are done by time  $h$ . This can only be achieved if a layering of width  $w$  and height  $h$  can be found. Since *PCMS* is NP-hard (Garey & Johnson, 1979) and can be reduced into the *layering assignment problem*, it follows that it is NP-hard as well (Eades & Lin, 1989). Therefore some heuristics of *PCMS* have been proposed as approximate solutions to the layering problem as well.

The input of the Coffman-Graham algorithm is a *DAG* without transitive edges and a width  $w$  which is the number of nodes in each layer. A transitive edge  $e = (u, v) \in E$  is an edge where there exists a path with length greater than 1 with start vertex  $u$  and end vertex  $v$  other than edge  $e$ . A *DAG* can be reduced into its transitive reduction in  $O(M(|V|))$  time, where the time for computing the product of two arbitrary  $n \times n$  matrices is  $M(n)$  (Aho, Garay, & Ullman, 1972). The Coffman-Graham algorithm is a two step algorithm, in the first step vertices are given a unique label  $\pi(v): v \in V$  (based mostly on their distance to source nodes) which will be used in step two for priority placement in layers. Vertices with high label will be placed in lower layers, vertices with low label will be placed in the upper layers. When picking vertices the algorithm uses comparison between finite sets based on a lexicographical ordering  $\prec$  defined as follows:

If  $S_1$  and  $S_2$  are two finite sets then  $S_1 \prec S_2$  if either

- $S_1 = \emptyset$  and  $S_2 \neq \emptyset$ , or
- $S_1 \neq \emptyset$  and  $S_2 \neq \emptyset$  and  $\max(S_1) < \max(S_2)$ , or
- $S_1 \neq \emptyset$  and  $S_2 \neq \emptyset$ ,  $\max(S_1) = \max(S_2)$  and  $S_1 \setminus \{\max(S_1)\} \prec S_2 \setminus \{\max(S_2)\}$ .

---

**Algorithm 4: Coffman – Graham**

---

DAG  $G = (V, E)$ , without transitive edges and integer  $W > 0$   
**for all**  $v \in V$  **do**  $\pi(v) \leftarrow \infty$   
**for**  $i = 1$  **to**  $|V|$  **do**  
    Choose  $v \in V$  with  $\pi(v) = \infty$  such that  $N_G^-(v)$  is minimized according to  $\prec$ ;  
     $\pi(v) \leftarrow i$ ;  
**end for**  
 $k \leftarrow 1, L_1 \leftarrow \emptyset, U \leftarrow V$   
**while**  $U \neq \emptyset$  **do**  
    choose  $v \in V$  such that  $N_G^+(v) \subseteq U$  and  $\pi(v)$  such that is maximized according to  $\prec$ ;  
    **if**  $|L_k| \leq W$  and  $N_G^+(v) \subseteq L_1 \cup L_2 \cup \dots \cup L_{k-1}$  **then**  
         $L_k \leftarrow L_k \cup \{v\}$ ;  
    **else**  
         $k \leftarrow k + 1; L_k \leftarrow \{v\}$   
    delete  $v$  from  $U$

---

### 2.3.2.3 Network-Simplex layering algorithm

The Coffman-Graham algorithm tends to produce a large number of dummy vertices, often to such extent that the area of the final drawing is worse than the area of the *longest-path algorithm* (Hai & Zhihui, 2013). The goal of minimizing the number of dummy vertices is the same as the goal of minimizing the total edge span (edge length) of a graph. Shown by Eades and Sugiyama in 1990, minimizing the number of dummy vertices guarantees a layering with minimal height (Eades & Sugiyama, 1990). Minimizing the total edge span can be modeled as an integer linear program defined as follows:

$$\begin{aligned} \min \quad & \sum_{(u,v) \in E} l(u, \mathbf{L}) - l(v, \mathbf{L}) \\ \text{subject to constraints: } & l(u, \mathbf{L}) - l(v, \mathbf{L}) \geq 1, \forall (u, v) \in E \\ & l(u, \mathbf{L}) \in \mathbb{Z}^+, \forall u \in V \end{aligned}$$

Due to the constraint matrix being totally unimodular (Schrijver, 1986) this problem has an integer solution which could be discovered using any of the available polynomial-time algorithms for solving linear programs. Gansner et al. proposed another way of solving the linear program using their *network-simplex algorithm* (Gansner E. , Koutsofios, North, & Vo, 1993). In their algorithm they introduce 2 extra criterions. Firstly, by assigning priority  $\omega$  on edges they can control length of the edge. Secondly, some edges might need to be at least of length  $\lambda$ :

$$\begin{aligned} \min \quad & \sum_{(u,v) \in E} \omega(u, v) l(u, \mathbf{L}) - l(v, \mathbf{L}) \\ \text{subject to constraints: } & l(u, \mathbf{L}) - l(v, \mathbf{L}) \geq \lambda(u, v), \forall (u, v) \in E \\ & l(u, \mathbf{L}) \in \mathbb{Z}^+, \forall u \in V \end{aligned}$$

The *network-simplex algorithm* does not guarantee polynomial running time, but has been tested to deliver very efficient result (small number of iterations) thus resulting in low running time.

### 2.3.3 Crossing reduction

Revisiting section 1.2 it is clear that the number of edge crossings affects the comprehensiveness of a drawing and should be eliminated. The number of edge crossings depends only on the relative ordering of vertices. This makes the task of edge crossing minimization a little easier since the exact x-coordinate of vertices can be ignored as long as the relative positioning is kept. Unfortunately finding orderings of vertices which minimize edge crossings in a layered graph is NP-hard, even if the graph is bipartite (only two layers) (Garey & Johnson, 1983). If the ordering of vertices in one layer is fixed the problem still remains NP-hard (Eades & Wormald, 1994).

We assume a properly layered graph  $G = (V, E)$  with partitions  $V = V_1 \cup \dots \cup V_k, V_i \cap V_j = \emptyset, 1 \leq i \neq j \leq k$ . A set of edges  $E = \{(u, v) | u \in V_i, v \in V_{i+1}, 1 \leq i \leq k-1\}$ . We also denote the ordering of a layer  $V_i$  by the permutation  $\pi_i$ , i.e. the ordering of  $V_1$  is explained by the permutation  $\pi_1$ . Since the number of edge crossings is determined by the relative positioning of vertices in different layers each permutation  $\pi_i$  of a layer  $V_i$  provides a solution to the *crossing minimization problem*. In conclusion we seek a set of permutations  $\Psi = \{\pi_i | 1 \leq i \leq k\}$  which minimizes the number of edge crossings  $C(G, \pi_1, \pi_2)$ .

#### 2.3.3.1 One-sided crossing minimization

The *one-sided crossing minimization problem (OSCM)* requires a graph  $G$  and a given permutation  $\pi_1$ , problem is to find a permutation  $\pi_2$  which minimizes  $C(G, \pi_1, \pi_2)$ . The notion of *crossing number* is important to many heuristics. Given a fixed permutation  $\pi_1$  of  $V_1$  the *crossing number*  $c_{uv}$  is given by the number of edge crossings between edges incident to  $u, v \in V_2$  when  $\pi_2(u) < \pi_2(v)$ . The sum of *crossing numbers*  $\sum_{u,v} \min(c_{uv}, c_{vu})$  provides a simple lower bound on the optimal number of crossings (Healy & Nikolov, 2013). Finding *crossing numbers* can be done by the sweep line algorithm (Sander, 1994) in time  $O(|V_1| + |V_2| + |E| + c)$  where  $c$  is the number of crossings. Many heuristics for solving OSCM have been proposed, some of the more popular are *Barycenter*, *Median*, *Greedy switch*, *Split* and *Greedy insertion*.

##### 2.3.3.1.1 Barycenter heuristic

The Barycenter averaging heuristic (Sugiyama, Tagawa, & Toda, 1981) build on the assumption that nodes which are close to their neighbors produces less edge crossings. Let  $N(u)$  denote the set of neighbors to vertex  $u$ , i.e.  $N(u) := \{v : \{u, v\} \in E\}$ . The position of a vertex is determined by the Barycenter (average) of its neighbors:

$$B(u) = \frac{\sum_{v \in N(u)} \pi_1(v)}{\deg(u)}$$

This computation is done  $\forall u \in V_2$ , if two vertices would end up with the same Barycenter value, either their original order is kept or they are separated by a small value. Due to its simplicity, running time and good results the Barycenter is one of the most used heuristics. Interestingly the algorithm will always find a crossing free ordering if one exists. The Barycenters of all vertices can be found in linear time, but the vertices need to be sorted meaning the total running time becomes  $O(|V_2| \log |V_2|)$  (Bastert & Matuszewski, 2001).

##### 2.3.3.1.2 Median heuristic

Just like the Barycenter heuristic the running time for the Median heuristic is  $O(|V_2| \log |V_2|)$ . In this heuristic the vertex  $u$  is assigned the median of its neighbors  $N(u)$ , if there exists two medians (as in the case of even number of vertices) the leftmost is chosen. In the case of two vertices being assigned the same median they are separated with a small number and a vertex with odd degree is always placed on the left of the two. Just like the Barycenter heuristic, the Median heuristic will always find a crossing free ordering if one exists. Median heuristic however can give a guarantee on an ordering  $\pi_2$  such that  $C(G, \pi_1, \pi_2) \leq 3C_{opt}(G, \pi_1, \pi_2)$ , where  $C_{opt}$  is the optimal number of crossings, i.e. minimal number of crossings (Eades & Wormald, 1994).

### 2.3.3.1.3 Greedy heuristic

Greedy insertion (Eades & Kelly, 1986) starts from the left with a vertex  $u$  and chooses the next vertex  $v$  to be the one which minimizes the number of crossings between edges incident to  $u$  and edges incident to  $v$ . Running time of greedy insertion is  $O(|V_2|^2)$ .

### 2.3.3.1.4 Greedy switch

Greedy switch has a large reminiscence with bubble sort, each consecutive pair of vertices  $u, v$  in a ordering  $V_2$  is tested. A switch of the vertices  $u, v$  in layer  $V_2$  produces a change in the total number of crossings by  $c_{vu} - c_{uv}$ . Greedy switch heuristic (Eades & Kelly, 1986) continues to swap consecutive vertices as long as the number of crossing is reduced. If a *crossing number matrix* is pre-computed the running time is  $O(|V_2|^2)$ .

---

#### Algorithm 5: Greedy switch

---

```

while moreEdges
  for  $u \leftarrow 1$  to  $|V_2| - 1$ 
    if  $c_{u(u+1)} > c_{(u+1)u}$ 
      switch vertices  $u$  and  $u + 1$ 
    if noMoreEdges()
      moreEdges  $\leftarrow$  false

```

---

### 2.3.3.1.5 Split

Eades and Kelly presented a third heuristic which work in the same sorting fashion as *greedy switch*, *split* heuristic however resembles quicksort and not bubble sort. A “pivot” element is chosen from the set, i.e. a vertex  $v$  is picked from layer  $V_2$ . After a “pivot” element has been picked the heuristic creates two sets  $L$  and  $R$ , containing vertices where  $L = c_{uv} < c_{vu}$  and  $R = c_{vu} \leq c_{uv}$ . This procedure is repeated until no further swaps can be made. The running time of the heuristic is at worst  $O(|V_2|^2)$ .

### 2.3.3.2 Multi-layered crossing minimization

The problem of finding an ordering  $V_i$  of a layer that minimizes the number of edge crossings does not become easier if the number of layers increase. One of the more usual multi-layer heuristics is to repeatedly use a *layer-by-layer sweep* of an *OSCM* heuristic (Healy & Nikolov, 2013). Another approach without the use of a *layer-by-layer sweep* was suggested by Eades and Sugiyama in 1990. The authors suggest the fixture of vertices in layer  $V_1$  and  $V_k$ , for any other vertex  $u$  in sub sequent layers the position of  $u$  is calculated as a weighted average (similar to Barycenter) of the x-coordinates of both  $N_G^+(u)$  and  $N_G^-(u)$ .

$$\pi_i(u) = \frac{\sum_{v \in N_G^+(u)} \pi_i(v)}{N_G^+(u)} + \frac{\sum_{w \in N_G^-(u)} \pi_i(w)}{N_G^-(u)}$$

### 2.3.4 X-coordinate re-assignment

*X-coordinate re-assignment* finds final positions for vertices in each layer with regards to the entire drawing, in other words x-coordinate re-assignment has the goal of minimizing the number of bent edges. A drawing with straight edges is more readable, however moving vertices around with respect to other layers often comes with the drawback of wider drawings. Consider an edge from layer  $i$  to layer  $j$ , the path  $(v_i, v_{i+1}, \dots, v_j)$  then consist of the original vertices  $v_i, v_j$  and  $(v_{i+1}, \dots, v_{j-1})$  which are the introduced dummy vertices. Unlike original vertices which often has an out and indegree of size  $> 1$ , dummy vertices has an out and in degree of 1. This makes it possible to keep the path of dummy vertices vertical. Ganser et al. proposed horizontal assignment as an integer optimization problem:

$$\begin{aligned} \min \quad & \sum_{e=(u,v) \in E} \Omega(e) \omega(e) |x(v) - x(u)| \\ \text{subject to: } & x(b) - x(a) \geq \delta(a, b) \end{aligned}$$

Where  $\Omega(e)$  denotes the importance of that edge being drawn vertically, this weight is not to be confused by the edge weight  $\omega(e)$ . The measurement  $\delta(a, b)$  is the minimum distance between consecutive vertices  $(a, b)$ . Gansner et al. defines three types of edges: (1) Both original vertices, (2) One original and one dummy vertex, and (3) Both dummy vertices. Each type gets assigned a weight  $\Omega(e)$ , (1) = 1, (2) = 2, and (3) = 8 as chosen by the authors but could be defined as other values. Choosing the  $\Omega(e)$  wisely could limit the “spaghetti effect”, i.e. limit the number of bends in long edges (Eiglsperger, Siebenhaller, & Kaufmann, 2005). The implementation (Gansner E. , Koutsofios, North, & Vo, 1993) can be transformed into a linear program by introducing extra constraints and additional variables. Thus increasing the constraint matrix and therefore negatively affect memory and time complexity.

Brandes and Köpf presented an algorithm (Brandes & Köpf, 2002) which has a linear running time in the number of vertices and edges. Their algorithm produces drawings which guarantee at most two bends, where no bends occur between dummy vertices. The measurement  $|x(v) - x(u)|$  of edge length is used just as in the algorithm by Gansner et al. Another similarity is the usage of median as the best measurement of vertex alignment. Given a set  $X = \{x_i\}$ , the sum  $\sum_{i=1}^k |x - x(i)|$  is minimized if  $x$  is the median of  $X$ . The first step of the algorithm computes four different drawings where the vertex position is assigned with regards to upper and lower neighbors, for each of these drawings the vertices are ordered *left-to-right* or *right-to-left* within each layer. During this phase Brandes and Köpf introduce three different measurements of conflict: Type 0, if a pair of non-inner segments cross or share a vertex. Type 1, when non-inner segment crosses an inner segment, since the algorithm tries to align inner-segments vertically all conflicts are solved in favor of inner segment. Type 2, when inner segment crosses each other, hopefully these conflicts are resolved in the *crossing minimization* step. In step two *blocks* of maximum sets of vertically aligned vertices are positioned, *blocks* can then be compacted with regards to the separation constraint  $\delta(u, v)$ . As a last step the 4 drawings are combined into a final drawing using the *average median* measurement on each layer.

## 2.4 State of the art graph drawing tools

We briefly describe four open source graph drawing tools which can be used for layer styled graph drawing. Each tool is evaluated in the initial exploration study and the best performing tool will be used as a benchmark in the main study.

### 2.4.1 Graphviz

Graphviz is an open source graph visualization tool. Graphviz has the capability to visualize layouts of many different formats; SVG, PDF, Postscript and common image formats. This makes Graphviz one of the most widely used graph drawing libraries on the market today. Apart from a wide variety of exportable formats Graphviz also offers a substantial amount of possible layout algorithms, dot – layered graph drawing, neato – spring model layouts, used for small graphs, fdp – similar to neato but reduces forces of spring instead of energy, and sfdp – same as fdp but works for larger graphs.

### 2.4.2 Cytogate

Cytogate is a opensource platform for visualizing complex networks, Cytogate was originally constructed to handle biological research but is today a general graph drawing platform comprised of algorithms for layer styled drawing, force-directed methods, and data analysis tools. Just like Graphviz, Cytogate has the capability to produce a wide variety of output formats making it useful for many applications.



### *2.4.3 Graphsharp*

Graphsharp is an desktop application offering graph layout algorithms for WPF applications, among the many offered drawing styles; Sugiyama, Compound-graph layout, and force directed are the most common with regards to above mentioned graphs drawing tools. Graph sharp build on the generic, directed and undirected data-structures provided by QuickGraph as a library for .NET. Like Graphviz and Cytogate, GraphSharp offers a wide variety of outputs format including an extension which exports the entire data-structure to your own .NET application.

### *2.4.4 Diagram.NET*

Diagram.NET is an open source drawing tool written in C#, it provides options to drag and drop components creating and initial layout which could later be manipulated using popular automatic graph drawing algorithms. It does not contain the same amount of exportable formats as mentioned tools above but can still provide exports to common image extensions: JPEG, PNG and many others.

In this chapter we explain in detail our main contribution, CluStic, the graph drawing tool for static vertices and clusters (drawings found in appendix E). We explain how CluStic works, how interactions between static and non-static components are handle furthermore we show solutions used by CluStic to solve the most common aesthetic properties. The chapter is divided into five parts, each part covers a step in the main algorithm and is constructed from a set of partial heuristics.

---

**Algorithm 6:** *Main algorithm*( $V \leftarrow \text{Vertices}, E \leftarrow \text{Edges}, C \leftarrow \text{Clusters}$ )

---

```

 $G \leftarrow (V, E, C)$ 
  removeSelfLoopEdges( $G$ );
  removeTwoLoopEdges( $G$ );

 $DAG \leftarrow \text{cycleRemoval}(G)$ ;
  longestPathLayering( $DAG$ );
  makeLayeringProper( $DAG$ );
  edgeCrossingMinimization( $DAG$ );
  xCoordinateAssignment( $DAG$ );

 $G_2 \leftarrow \text{remove all dummy vertices and edges from } DAG$ ;
  re – introduce all self and two loop edges removed from  $G$  to  $G_2$ ;

  return layered positioning of vertices in } G_2

```

---

### 3.1 CluStic – preliminary constraints

Preliminary constraints end execution of partial heuristic if not removed, mostly due to infinity loops in the heuristics used by CluStic. Such constraints are: *self-loops*, edges starting and ending in the same vertex, and *two-loops*, a pair of edges in which one edge in the pair goes from vertex  $a$  to  $b$  and the other goes from  $b$  to  $a$  instead.



Figure 8 - Self loop

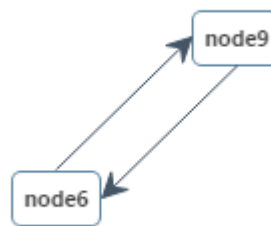


Figure 9 - Two loop

### 3.1.2 Self-loop removal

Removing self-loops is done in linear time with regards the number of edges in the graph, the algorithm scans through each edge in the graph and compares the start and ending node. If equal, the edge is removed from the original set of edges and stored in a temporary set which will be merged with the original set in a later phase of the main algorithm. Removing self loops is an easy step since introducing them back to the original set later on does not affect the drawing or any of the partial algorithms.

### 3.1.3 Two-loop removal

Removing two-loop edges is not as straight forward as self-loop edges, depending on which of the two edges that is removed affects the outcome of the final drawing. Two-loop edge removal combines two strategies: the first checks the possible layering of the two edges, an edge with a starting vertex placed in the layer furthest from the source vertex is the most probable to remove. The second strategy checks if the removal of an edge causes an introduction of a new source or sink. Graphs used in hierarchical graph drawings are called ST-graphs. ST-graphs contain a start vertex  $s$  (source) and an ending vertex  $t$  (sink), all other vertices are placed in layers between the source and the sink. Since a source and sink should be placed in the highest and lowest layer respectively none of them can exist in a layer in-between. When one of two edges about to be removed the algorithm must check so the removal of the current edges will not turn one of its two connected vertices into a new source or sink. The graph data-structure of CluStic consists of three lists, the first contains all vertices, the second contains all edges and the third is an adjacency list containing arrays of in and out neighbors of each vertex. In and out neighbor arrays are separated, storing them separately constructs easy “new sink and source” checks. Such check consists of counting the degree, i.e. the length of each in and out neighbor array. As an example consider figure 10, if we were to remove the edge going from vertex *node9* to *node6*, we have to check if the removal causes *node9* to become a sink (since the edge start in this vertex) or causes *node6* to become a source (ends in this vertex). Meaning a lookup on the length of the out neighbor array of *node9* and a lookup on the in neighbor of *node6* will be conducted, if the length is larger than 1 it is fine to remove such edge.

The second problem occurs if no layering of vertices is calculated, therefore to which layer a certain vertex belongs is not determined. Removing one of two edges may cause the remaining edge to go against the flow in the final ordering leaving a less feasible drawing. This can be solved by a preordering which is decided by a breadth-first search placing vertices involved in a two loop edge in different layers if possible. The algorithm can then determine which of the two vertices should be the start vertex and end vertex by checking which of the two vertices have a larger distance to the source vertex.

Algorithm 7, takes as input a *Digraph* containing two lists, one with all vertices  $V$  and one with all the edges  $E$ . Each vertex is an object containing an attribute *label* which defines the name of a vertex in the graph. Each edge is an object which contains *from* and *to* attribute stating which vertices the edge goes from and to. The algorithm produces a set containing all the edges removed by the algorithm, at the end of algorithm 6 (above) removed edges are reintroduced to the graph. The algorithm starts by picking one edge  $e$  from the edge list, for the *from* vertex of  $e$  retrieves all incoming neighbors  $N_G^-(from)$ . If the *to* vertex of  $e$  is in  $N_G^-(from)$  and fulfill all external constraints,  $e$  is one of two edges in a two-loop. External constraints are:

- $distanceMatrix(source, from) \geq distanceMatrix(source, to)$ .
- $e$  has not already been added.
- Removal of the edge does not create a source or a sink.

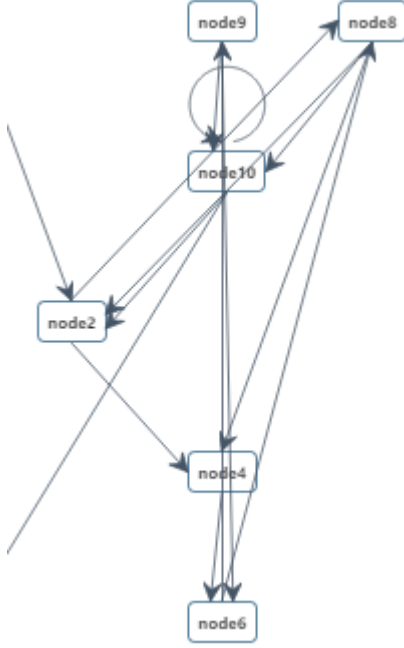


Figure 10 – Part of drawing without two loop constraints.

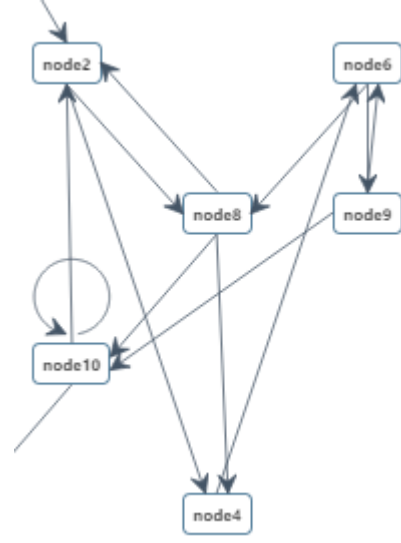


Figure 11 – Part of drawing with two loop constraints.

As shown in the illustrations above, two-loop constraints have a positive effect on the final drawing.

---

**Algorithm 7:** Two loop removal

---

```

 $G \leftarrow (V, E);$ 
 $twoLoopEdges \leftarrow \emptyset;$ 
for each edge  $e \subseteq E$ 
   $edgePartOfTwoLoop \leftarrow false;$ 
   $toVertex \leftarrow e.toVertex;$ 
   $fromVertex \leftarrow e.fromVertex;$ 
  if  $N_G^-(fromVertex) > 1$ 
    for each vertex  $v$  in  $N_G^-(fromVertex)$ 
      if  $v = toVertex$ 
         $edgePartOfTwoLoop \leftarrow true;$ 
        remove  $v$  from  $N_G^+(fromVertex);$ 
  if  $edgePartOfTwoLoop = true$ 
    remove  $fromVertex$  from  $N_G^-(toVertex);$ 
    remove edge  $e$  from the set  $E;$ 
     $twoLoopEdges \leftarrow twoLoopEdges \cup e;$ 

```

---

## 3.2 Cycle removal

At this step of the main algorithm the graph is a *digraph* (directed graph) and may therefore still contain an arbitrary number of cycles. Removing cycles relates to the task of finding a minimal *FAS* (*feedback arch set*), the *FAS* is a set of edges, when removed from the original edge set of the graph leave a *DAG* (*directed acyclic graph*). What we really seek is the minimal *FS* (*feedback set*), which is a set of edges, when reversed turns the digraph into a *DAG*. Due to the fact that a minimal *FAS* is always a *FS* (Healy & Nikolov, 2013) we can use a heuristic for the minimal *FAS* problem in order to find a *FS*.

The easiest way to remove cycles is to build a linear ordering (left - right) of vertices in the graph and pick each edge which moves against the flow (right - left). CluStic started out with this approach; first it calculated the linear ordering using a *BFS* (breadth – first search). This places vertices in a *x* position, there may be many vertices with the same *x* position, the algorithm then searches through the edge list and placing edges which has a *from* vertex with larger *x* position than their *to* vertex in the *FS*.

---

### Algorithm 8: Easy cycle removal

---

```

 $G \leftarrow (V, E)$ , where  $V$  contains no isolated vertices;
positionArray  $\leftarrow$  doBreathFirstSearch( $G$ ), store  $x$  position for each vertex;
 $fs \leftarrow \emptyset$ ;
for each  $e \in E$ 
    if ( $e.from > e.to$ )
         $fs \leftarrow fs \cup e$ ;

```

---

The *easy cycle removal* algorithm, though easy to follow has its drawbacks in the created *FS*. The ordering in which the algorithm traverses vertices in the graph affects the outcome of the final drawing. As an example, figure 13 is the resulting linear ordering of a *BFS* for a given graph structure with 12 vertices and 15 edges. Red arrows indicate edges with a direction against the flow in the linear ordering, i.e. edges placed in the *FS* by the algorithm. However, in this graph structure there is no need for any edges at all in the *FS*, due to unfortunate vertex transversal the produced ordering forces two edges against the flow. Figure 12 presents an ordering of the same graph structure produced by the *enhanced greedy algorithm (EHG)*, as shown there exists a linear ordering which produces no edges in the *FS*.

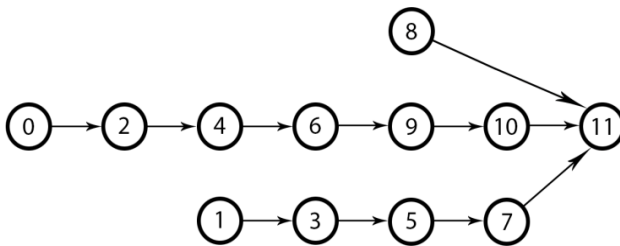


Figure 12 - Linear ordering produces by *EHG*

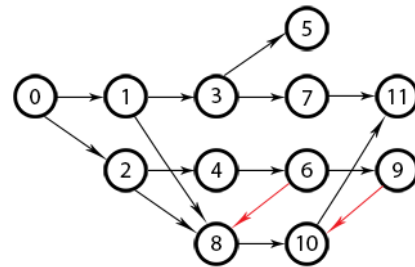


Figure 13 - Linear ordering produce by *BFS*, red arrow indicate edges in *FS*.

The *EHG* (algorithm 2) (Eades, Lin, & Smyth, 1993) makes use of the fact that sources and sinks cannot be part of cycles in a graph. If a graph contains no cycles *EHG* would never produce a linear ordering containing edges with a direction against the flow. *EHG* transverses the graph vertex by vertex starting with the sink, after a vertex been visited it is removed from the set of vertices and all

edges going into it are removed from the set of edges. This procedure loops as long as there are sinks in the graph, notice that removing a sink may create new ones, as in the case (figure 12 above) with no cycles. After all sinks have been removed *EHG* continues with the sources in a similar manner but instead remove edges going out from the source. If there are neither sources nor sinks left, the next vertex considered is the one with largest total degree  $N_G^+ - N_G^-$ , edges are removed as if they were source edges. Removing a “largest degree vertex” may create a new source or sink, therefore the algorithm starts once again from the top searching for sinks. A drawback of the *EHG* implementation used by CluStic is that edges placed in the *FS* may not be edges with a direction against the flow in the final drawing. In this aspect *BFS* outshines *EHG*, but *EHG* comes with a guaranteed upper bound on the *FS* size leaving *EHG* as a better pick. As show in figure 14(left), the drawing made by *BFS* contains no edge crossings due to edges in *FS* being those which are against the flow unlike the drawing for *EHG* which contains four edge crossings.

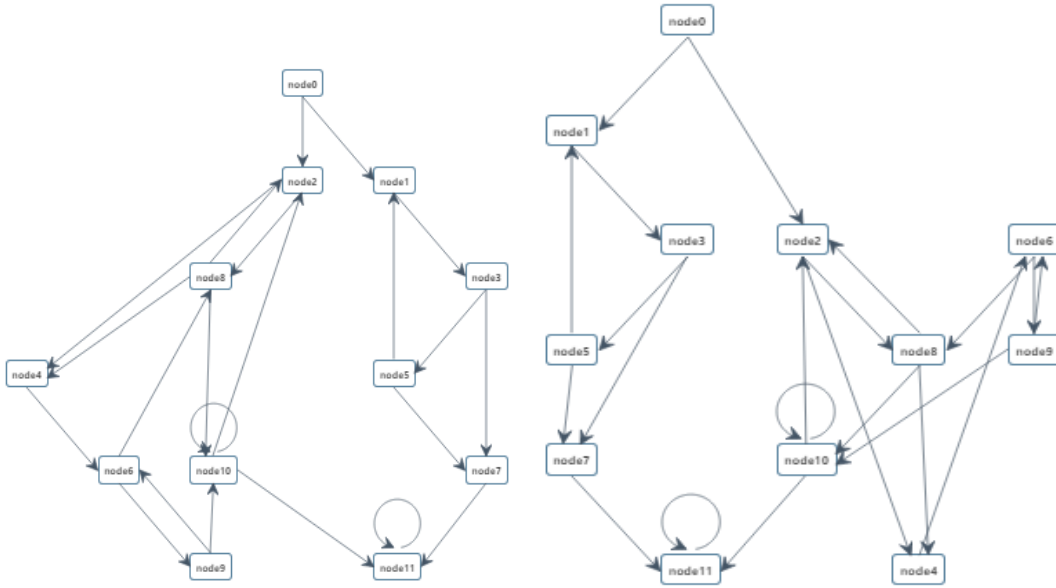


Figure 14 - Left: drawing produced with a linear ordering from BFS. Right: drawing produced with a linear ordering from EHG.

### 3.3 Vertex layering

Vertex layering assigns each vertex in the graph to a layer where many vertices may be assigned to the same layer with no regards to internal layer ordering. Naturally the source vertex is placed at the top layer and the sink is placed at the bottom starting with layer 0. The layering is determined by the *longest path algorithm* (Healy & Nikolov, 2013) which also guarantees a minimum height of the graph i.e. the longest path from source to sink strictly going in the direction of the flow. CluStic uses a modified version of the *longest path algorithm* which has the option to consider a static vertex or a cluster of vertices. The *longest path algorithm* starts by assigning sink vertices to layer 0, vertices placed in above layers are determined by the outgoing edges of vertices placed in a previous layer. If a vertex has all of its outgoing edges incident to vertices in previous layers it is placed in the current layer. This procedure continues for all the layers until source vertices are placed in the top layer.

### 3.3.1 Static vertex layering

This modification to the *longest path algorithm* gives the possibility to keep the position of certain vertices static in regards to other vertices. The *longest path algorithm* normally pick vertices for a given layer depending on the adjacent vertices of its outgoing edges, meaning static vertices must be ignored in this step. CluStic removes all static vertices from the original vertex list and store them separately. The *longest path algorithm* then proceeds as normal only considering vertices in the original vertex list. Figure 15 shows a drawing of a graph when vertex *node4* have been removed. Figure 16 shows the same drawing when vertex *node4* has been re-introduced one layer above vertex *node8*. Notice the that re-introducing *node4* eliminate the four edge crossings which exists for the drawing in figure 14 (above), this is one of the benefits of static vertices, but static vertices must be used with care since they might as well make the drawing worse.

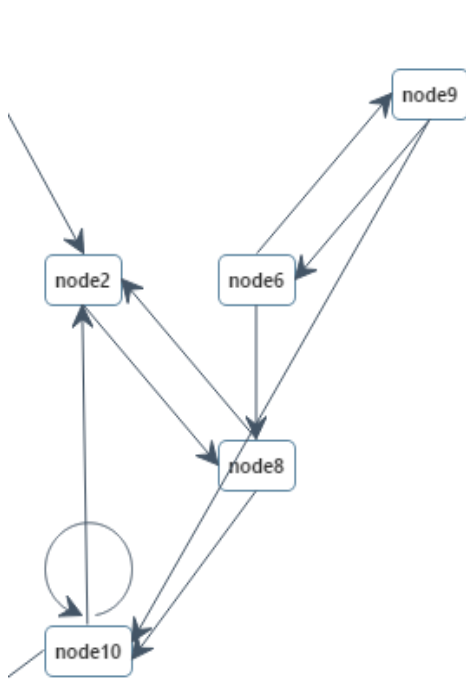


Figure 15 - Drawing where vertex *node4* have been removed.

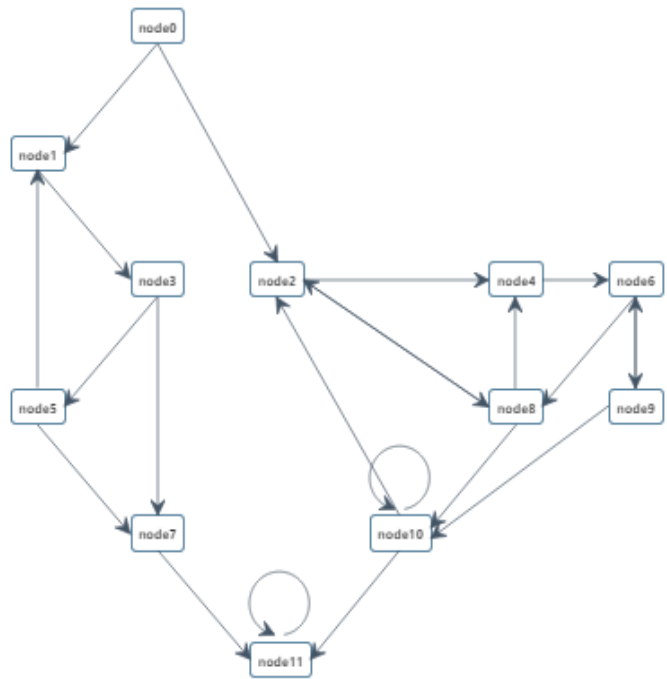


Figure 16 – Same drawing where vertex *node4* have been re-introduced.

### 3.3.2 Cluster layering

Clustering is one of the main features CluStic, a cluster in a graph can be seen as a set of vertices which are tightly coupled or share a close connection between each other i.e. sub-graph. Examples of clusters could range from anything, such as a cycle in the graph or a number of vertices being connected by shared edges. When clusters are present each cluster is in need of a layering algorithm which maintains their internal structure with regards to the remaining vertices in the original graph. CluStic assigns each vertex of a cluster to a layer when the position of the cluster sink/ sinks is determined. The algorithm builds a layering starting at layer 0, each sink of the graph is placed at this layer. If such a sink is part of a cluster, each vertex in the cluster is assigned to a layer relative to the internal layering of the cluster and layer 0. The algorithm then builds layer 1, each remaining vertex which has all of its edges incident to vertices in a layer below current are assigned to layer 1. As before, if a vertex is part of a cluster all vertices in that cluster is layered. This procedure continues until source vertices have been assigned a layer. As a last step, each static vertex removed in previous step is reintroduced to their respective layer.

---

**Algorithm 11:** The longest path algorithm for clusters
 

---

```

DAG  $G \leftarrow (V, E)$ 
Remove all vertices with static constraints
Assign all sink vertices to layer 0
 $U \leftarrow \{\text{all sink vertices}\}$ 
 $Z \leftarrow Z \cup U$ 
currentLayer  $\leftarrow 1$ 
while  $U \neq V$  do
  select  $v \in V$  with  $N_G^+(v) \subseteq Z$ ;
  if vertex  $v$  is not assign a layer
    Assign  $v$  to the layer with number currentLayer;
    if  $v$  is part of cluster
      Assign all vertices in cluster to currentLayer + internalLayer
       $U \leftarrow U \cup \{v\}$ 
  if  $v$  is assigned to layer
     $U \leftarrow U \cup \{v\}$ 
  if no vertex is selected
    currentLayer  $\leftarrow$  currentLayer + 1;
     $Z \leftarrow Z \cup U$ 
Re – introduce all vertices with static constraints
  
```

---

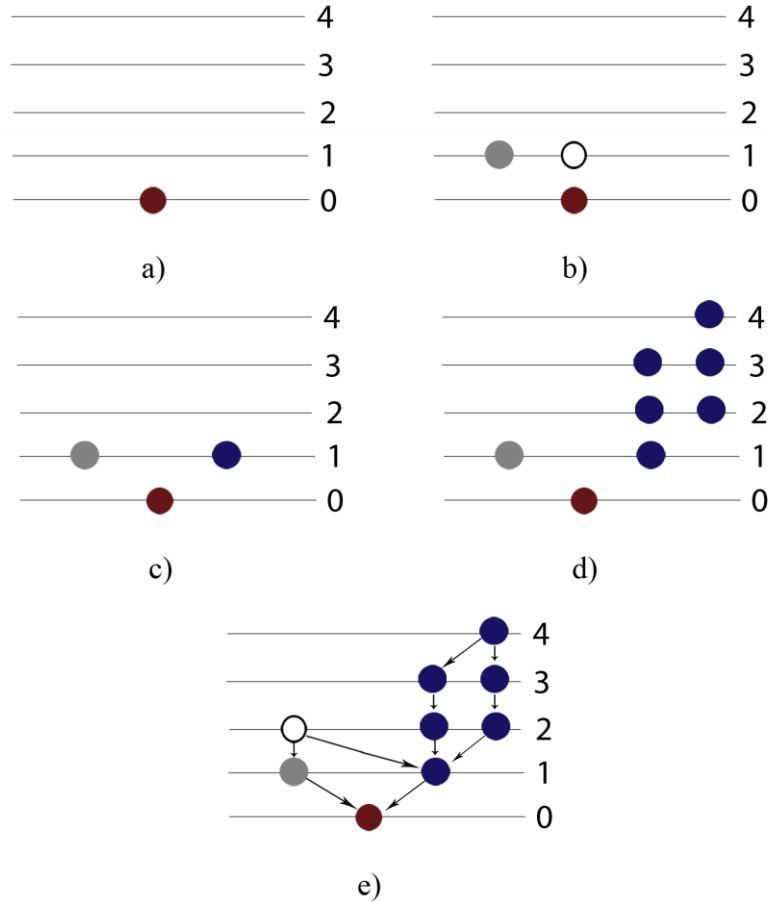


Figure 17 - Illustration of the cluster layering. A) The sink (red) of the graph is assigned to layer 0. B) A non-cluster vertex (gray) is assigned to layer 0, a non-cluster vertex (white) is not assigned to the layer since it has edges going to a vertex not in layer 0. C) A cluster vertex (blue) is assigned to layer 1, leading to D) where all vertices in the cluster are layered. E) The last non-cluster vertex is also assigned to layer 2 since all its outgoing neighbors are assigned to lower layers.



### 3.4 Edge crossing minimization

Step four of the CluStic main algorithm is edge crossing minimization, up until this step the vertices have been assigned to a layer but currently all of them have the same position within that layer. Randomly placing vertices within the layer with some specific distance between each other most certainly leads to edge crossings. The edge crossing minimization step places vertices within a layer whilst fulfilling certain criterions: vertices are placed with a minimum separation value between each other, and edge crossings are kept at a minimum by placing vertices with regard to their neighboring layers. CluStic builds vertex positions within layers using the *one-sided crossing minimization (OSCM)* (Healy & Nikolov, 2013) model repeatedly. OSCM calculates the minimum  $C(G, \pi_1, \pi_2)$  of a graph  $G$  where  $C$  is the crossing number of edges going from a fixed permutation  $\pi_1$  of top layer and a permutation  $\pi_2$  of bottom layer. CluStic uses the Barycenter heuristic (Sugiyama, Tagawa, & Toda, 1981) to find the minimum crossing number between layers of the graph. The Barycenter heuristic calculates the average of each incoming or outgoing neighboring set (depending on which sweep). A number of vertices may be assigned the same Barycentric value, if so the order of which they are added to the layer will be kept. Assigning many vertices to the same position and then just arranging them after some internal ordering may at first look like it may lead to additional edge crossings. However this issue is dealt by the layer by layer sweep (up – down, down – up) which guarantees that each layer permutation  $\pi_2$  is consider once when  $\pi_1$  is the layer above and once when  $\pi_1$  is a layer below (except for source and sink layers).

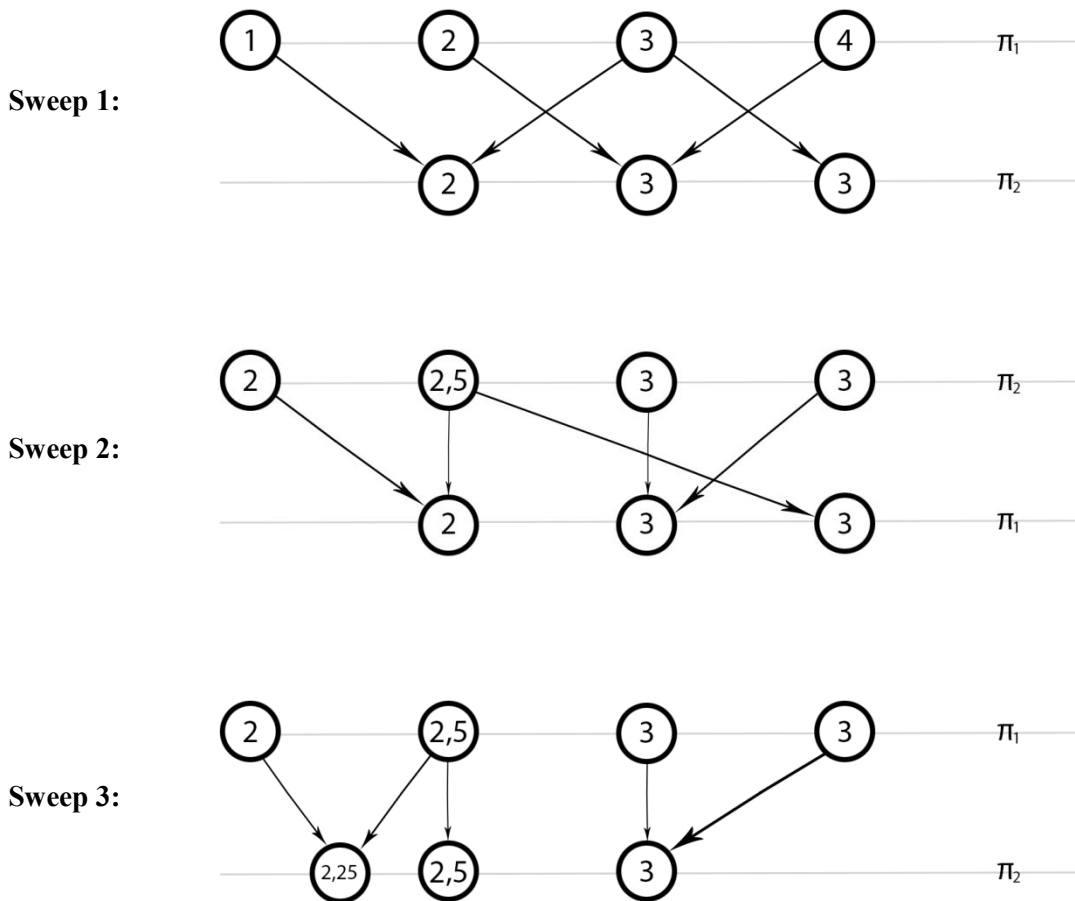


Figure 18 - Illustration of the Barycenter sweep.

Figure 18 illustrates the Barycenter method; Sweep 1 (up - down) shows the Barycentric values and the permutation  $\pi_2$  of the bottom layer with regards to its incoming edges. Sweep 2 (down - up) shows the Barycentric values when top layer is being permuted with regards to outgoing edges. Sweep 3 (up - down) shows the layer position of vertices in the graph. As shown after the third sweep no edge crossings exist between top and bottom layers. The Barycentric method normally runs for a given number of sweeps or until convergence of some specified value.

### 3.4.1 Edge crossing minimization for static vertices

In the Barycentric model, layers are permuted until convergence to a specified value or a number of sweeps have been reached, leading to an edge crossing free positioning of vertices if one exists. Given this, if static vertices are introduced as normal vertices to the Barycenter method their specified constraint might be broken. Therefore static vertices are removed from the edge crossing minimization step and re-introduced after positioning of normal vertices has been completed, meaning static vertices will be assigned a value equal to the Barycentric value of the vertex they are statically placed against.

### 3.4.2 Clustered crossing minimization

Positions of cluster vertices in the internal layering may not be altered by the positioning of the remaining vertices in the original graph. It is assumed that when clusters are positioned in the graph each cluster vertex has already been assigned its final x-coordinate within the cluster. Using the internal x-coordinate of vertices in a cluster means the source or sink of a cluster can be used to position the cluster relatively to remaining vertices. Each cluster vertex which is not a source or sink will be placed in the original graph at a position equal to their internal cluster position aligned against their cluster source and sink. Since the source or sink of the cluster has a Barycenter in the original graph they will update Barycenter values for the entire cluster. However in order to not alter positions of vertices in the cluster, CluStic places non clustered vertices which are intermediate with cluster vertices on a safe Barycentric position outside of the cluster vertex span of the layer.

---

**Algorithm 13:** Edge crossing minimization for clusters, **input** ( $G \leftarrow \text{graph}, \text{sweep}$ )

---

**if** sweep is **up – down**

**for each** layer **from** height of  $G - 1$  to 0

    barycenter(layer + 1, layer)

**for each**  $v \in \text{layer}$

**if**  $v$  is source in cluster

        set barycentric value  $B(u)$  for all vertices  $u$  in cluster to:  $(\text{clusterX}(u) - \text{clusterX}(v)) + B(v)$

**if**  $v$  is not in cluster and  $B(v)$  is between two vertices from same cluster

        Place  $v$  as close to  $B(v)$  as possible without being between two vertices from same cluster

**if** sweep is **down – up**

**for each** layer **from** 1 to height of  $G$

    barycenter(layer + 1, layer)

**for each**  $v \in \text{layer}$

**if**  $v$  is sink in cluster

        set barycentric value  $B(u)$  for all vertices  $u$  in cluster to:  $(\text{clusterX}(u) - \text{clusterX}(v)) + B(v)$

**if**  $v$  is not in cluster and  $B(v)$  is between two vertices from same cluster

        Place  $v$  as close to  $B(v)$  as possible without being between two vertices from same cluster

---

As seen in algorithm 13 all vertices which are part of a cluster are positioned with regards to the source and sink vertices of the cluster. Algorithm 14 shows the interaction between the different sweeps needed in the Barycentric positioning of clusters.

---

**Algorithm 14:** Edge crossing minimization for clusters

---

$DAG\ G \leftarrow (V, E)$

$U \leftarrow \{All\ clusters\ G_i = (V_i, E_i) in\ G\}$

$convergence \leftarrow convergence\ constraint\ or\ some\ specific\ value$

*Remove all vertices with static constraints*

**Until** *convergence constraint is met*

$edgeCrossingMinimizationForClusters(G, up - down)$

$edgeCrossingMinimizationForClusters(G, down - up)$

*Re – introduce vertices with static constraints back to the cluster*

---

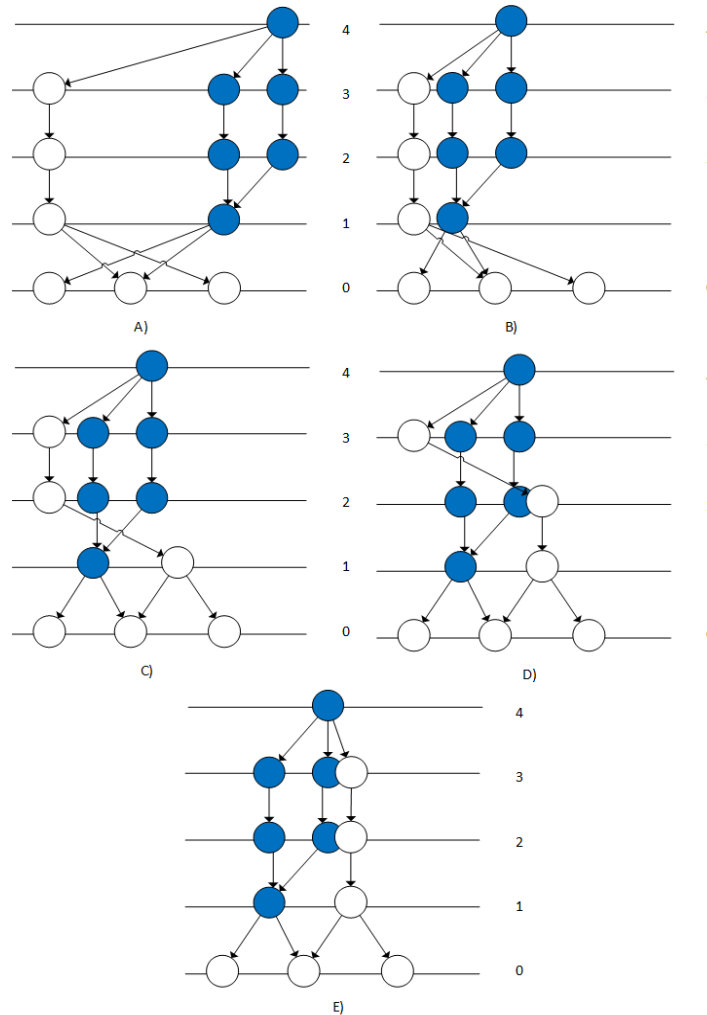


Figure 19 – A) Illustrates the initial positioning of the vertices. Vertices in blue are part of the same cluster, vertices in white are other vertices not part of any cluster. B) Show when the sink of the cluster is aligned to the Barycentric value of its neighbors. As illustrated the entire cluster is moved when the sink is aligned. In C, D and E we can observe how non-cluster vertices are aligned separately.

### 3.5 X-coordinate assignment

In the previous step, vertices were assigned a layered specific position which reduces or even eliminates the number of crossing edges. In this step of the CluStic algorithm vertices are assigned their final layer position. A vertex may not be assigned a position which breaks the internal ordering of a layer determined in previous step. X-coordinate assignment is also known as edge straightening due to the result of this step being a layout with an increased amount of straight edges. Studies show (Purchase, Cohen, & James, 1996) (Ware, Purchase, Colpoys, & McGill, 2002) that straight edges are easier to follow than edges which contain bends, thus the real problem are edges with a span larger than 1. Long edges should gain priority over short edges if two edges start in the same vertex, thus the dummy vertex should gain priority over original vertices in the layer. The edge straightening step can be solved by a slight modification of the Barycenter method in previous step. The following constraints are required when modifying the Barycenter method; first, each vertex is assigned a priority which is defined by the degree of the vertex. Depending on which sweep the algorithm is in the degree is either the number of incoming or outgoing edges. Secondly, a vertex is only allowed to push other vertices if those vertices have less priority than the vertex itself. Finally, vertices are considered in descending order from highest priority to the lowest meaning dummy vertices are assigned x-coordinate first (Sugiyama K. , 2002). Figure 20 illustrates the x-coordinate placement,  $\pi_2$  represents the layer in which vertices are assigned to a layer. The priority is represented by the number inside the vertex, red means dummy vertex and arbitrary high priority, in this case an 8. In the top figure vertices are aligned in the layer by their Barycentric order, in the bottom figure the dummy vertex is aligned first by its Barycentric value, this pushes vertices on the right side away two step. Next is the vertex with the second highest priority, vertex 3, this pushes vertices on the right side of vertex 3 one step to the right. Finally the two remaining vertices with priority 2 are assigned to their Barycentric value, the one furthest to the right cannot be moved since there are no open positions or vertices with less priority on the left of it.

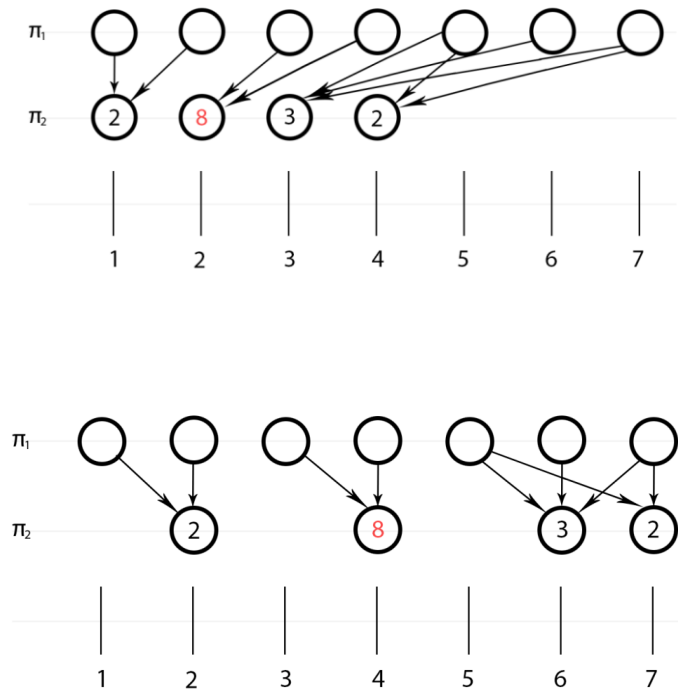


Figure 20 - Illustration of the x-coordinate assignment step. Above, vertices are order by their Barycentric value. Below, vertices after they have been aligned by their Barycentric value in priority descending order.

Since the Barycentric ordering is kept when assigning vertices their x-coordinate, static vertices poses no problem to the layout algorithm. However, clusters need to preserve their internal layout in the final drawing meaning no vertices in the cluster may be pushed around by vertices outside of the cluster. Using the same technique as in the Barycentric cluster placement step will not work in x-coordinate assignment. Suppose vertices in clusters are assigned a higher priority than dummy vertices. This would lead to a cluster source / sink is picked, afterwards all vertices in the cluster are relatively assigned with regards to the source / sink. If a second cluster is considered which relative placement of some vertex in the cluster has the same x-coordinate as any of the vertices in the first cluster it would lead to bad placement. Suppose this is solved by allowing cluster vertices to push all other vertices around, even those with the same priority. This would lead to malformed cluster structures since one layer containing two clusters may be pushed out of position whilst all other cluster layers stay the same. To solve this problem we present the *breathing step*. The *breathing step* is just an extension to the normal x-coordinate assignment step. Phase one of the *breathing step* is *inhaling*, each cluster is compressed into a single line cluster with one node in each layer starting at the source of a cluster. *Inhaling* clusters creates a new graph structure comprised from all single line clusters and all non cluster vertices and edge. The new graph structure is then assigned x-coordinate after the priority Barycenter method with highest priority for vertices in single line clusters. When all vertices have been assigned an x-coordinate the algorithm proceeds into to the *exhale* phase. *Exhaling* clusters transforms them back into their internal structure from before the *inhale* phase and places them back into the graph. By using the internal *box-size* value, a cluster is placed back into the graph by pushing multiple layers at once if any intersections are discovered between clusters. The *box-size* value is constructed by calculating the area of a cluster, i.e. *height x width* of a cluster. This provides enough space in the original graph to fit the entire *box-size* value of the cluster. Algorithm 15 explains the interaction between *breathing* clusters and the priority Barycenter calculations.

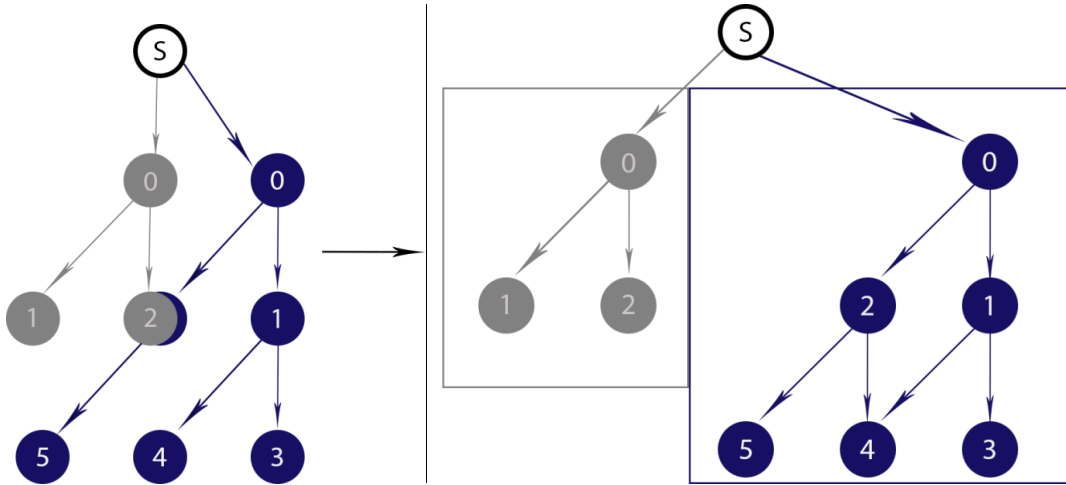


Figure 21 - Left: illustration of the problematic situation with overlapping clusters blue - gray. Right: After the *breathing* phase has been applied, borders indicate the *box-size* value.

---

**Algorithm 15:** *X – coordinate assignment for clusters*

---

```

DAG  $G \leftarrow (V, E)$ 
 $U \leftarrow \{ \text{All clusters } G_i = (V_i, E_i) \text{ in } G \}$ 
convergence  $\leftarrow$  convergence constraint or some specific value
for each cluster  $\in U$  do
    inhale(cluster)
doPriorityBarycenter(DAG)
for each cluster  $\in U$  do
    exhale(cluster)

```

---

## 4 EVALUATION

In this chapter we describe the methods used when evaluation CluStic and other graph drawing tools, we describe the two studies used, similarities as well as differences between them. Further we describe which subjects are used, how they test each graph drawing tool and what variables are used for evaluation.

### 4.1 Initial exploration study

One goal of the thesis is to investigate whether clustering effects the quality of a layout in a positive way. We therefore design a initial study which aims to compare CluStic with the best performing graph drawing tool from the set of tools presented in chapter 2.4. In the initial study each tool present 5 drawings of the same set of graphs to a subject, the subject traces the longest path from start-finish and fills in each answer. The evaluation program records: the time spent on each graph, the answer, and the assessment of mental effort by the subject. From this data the best performing tool is selected for comparison with CluStic in the main study.

#### 4.1.1 Input data

From a set of 30 workflows graphs, we choose five graphs (evaluation set) which represent the wide variety of possible layouts in the original set. Graphs in the original set vary in size from three vertices up to 24 whereas graphs in the evaluation set ranged from 12 vertices to 24. For every graph in the evaluation set a corresponding layout is calculated by each of the 4 programs, giving a total of 20 layouts (appendix A, B, C and D). The graphs are chosen in such fashion that no two vertices overlap with each other, eliminating potential misinterpretations and therefore reduces cognitive load.

#### 4.1.2 Task

The task was to determine the longest path between two vertices, since the graph structure is layered these vertices were naturally source  $s$  and sink  $t$ . Paths in the study had to pass under the following constraints.

- A path originates in the source vertex of the graph and ends in the sink vertex.
- A path is any set of edges which makes it possible to construct a walk from the source vertex to the sink vertex.
- The longest path is a path which has the largest edge set.
- Path length between nine and thirty, this was to ensure a reasonable level of difficulty.
- All subjects were presented the same set of graphs.

#### 4.1.3 Evaluation program

The evaluation program, figure 22, highlights a source vertex  $s$  and a sink vertex  $t$  in either a special color or by a unique shape. Doing so reduces time spent locating start and end of longest path and also reduces cognitive load. At startup subjects was presented with an empty drawing area, the only visible elements were answer boxes at the right hand side. This way subjects could become familiar with the layout of the program before they started the test. After subjects pressed *start* in the lower right corner the test began. In each iteration a layout is drawn, simultaneously a timer starts which records response time. When subjects were done typing their answer into the *answer* input area and switched focus to the *mental effort* input area. When subjects switches focus the timer stops and response time for current iteration was stored in memory. *Mental effort* is an assessment of task difficulty measured on a 7-point scale ranging from (1) *very low mental effort* to (7) *very high mental effort* (Zugal, Pinggera, Reijers, Reichert, & Weber, 2012) and (Mayer & Chandler, 2001). After subjects assessed mental effort they proceed to the next drawing by clicking *next*. Focus then switched back to the *answer* input field eliminating time spent moving the cursor, resulting in more accurate response times. Every

subject moved through twenty iterations of drawings, filling in answer and mental effort each time. Drawings are the same for all subjects but they are presented in random order each time.

The evaluation program ran locally on an Acer timeline 3820TG with an Intel i5 M450 2.40 GHz processor and an external 24" monitor. Subjects are computer science or electrical engineering students in year three to five with none to basic knowledge of graphs. Subjects participating in the study are placed in a room with the local system setup, at most one subject at a time could participate in the study. Before starting subjects were introduced to basic graph theory and the task was introduced. All subjects did a think-out-loud test run before the start of the actual test, during this think-out-loud run subject could ask question or the supervisor could step in if they did anything wrong. Estimated running time was around 15 minutes, during which the subject was not allowed any breaks.

#### **4.1.4 Evaluation**

All results: accuracy, mental effort and response time were averaged for each of the five graph drawing tools. The tool which produced the best drawings was decided by a ranking: (1) accuracy, (2) response time, (3) mental effort. If two or more tools shared the same average accuracy we looked at the response time, if response time was similar within a 3 seconds we looked at mental effort, if two or more tools once again produced the same averages we looked at strict response times. The graph drawing tool with the best results were picked as measurement for the newly constructed algorithm in the main study.

### **4.2 Interview**

An interview provides a subjective, more personal point of view. From an interview it is possible to reach a conclusion on which aspects contributes to a good graph layout for a particular person. This is of importance since a drawing is only as good as it is perceived by its observers. After a subject finished the study, they were presented with the drawing of the graph on which they achieved worst accuracy. Following this drawing the subject was asked three questions investigating the source of their performance drop and possible solutions.

- What made this drawing harder to interpret?
- What parts made this graph easy to interpret?
- If you could change anything, what would it be?

Each subject were then presented with a manually drawn representation of the same graph and asked two questions.

- Which graph is the easiest to interpret?
- If manual is the answer, which parts of the manually drawn graph make it easier to interpret?

### **4.3 Main study**

The goal of this study is to evaluate the new graph drawing tool CluStic against the best performing drawing tool (Cytogate) from the Initial exploration study. Just like before improvements and limitations of the method are discussed and presented in chapter 5.

#### **4.3.1 Input data**

From the original set of workflows in the initial exploration study the same 5 workflows with varying complexity is used. A subject spends around 2-3 times longer on graphs drawn by BFS (appendix F), this means that the amount of time a subject puts into doing the test is roughly the same as in the initial exploration study. A subject spends roughly the same amount of time taking both studies, leading to fewer differences in energy level. Subjects were presented with layouts from three different graph drawing tools which cover layouts from: the best performing tool in the Initial exploration study

*Cytogate*, a basic *BFS-layout* and finally the CluStic graph drawing tool. Using both good and bad measurements for evaluation help determine the quality of the CluStic graph drawing tool.

### 4.3.2 The Task

The task used is the same as in the Initial exploration study.

### 4.3.3 Evaluation program

The Evaluation program used is the same as in the Initial exploration study but drawings presented to subjects are different. Subjects are 10 new students with the similar knowledge and background to those subjects in the Initial exploration study. Using 10 new subjects removes the possibility of recognizing drawings and improving their own performance leading to skewed results and a bad evaluation. Each subject uses the evaluation program in a similar environment as in the Initial study.

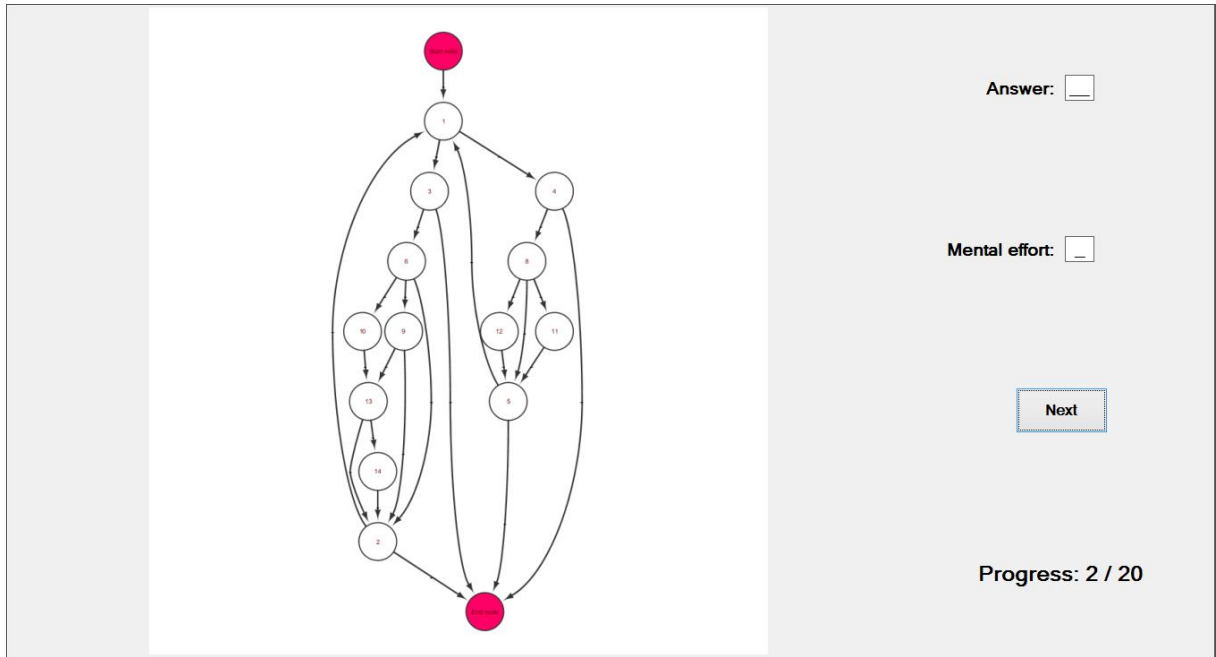


Figure 22 - Screenshot of the evaluation program.

### 4.3.4 Evaluation

The aim of this master thesis is to find out if clusters have a positive effect on the quality of a graph layout. We evaluate the algorithms in the main study by looking at the mean and standard deviation of their respective sample distribution. Low standard deviations together with a larger difference in means will indicate that the observed differences in sample distributions are more likely to occur. On the contrary, large standard deviations means non-uniform sample values and would indicate a higher amount of uncertainty in the sample distributions.

We will use two different types of variables when retrieving sample data, *dependent* and *independent*. The dependent variable will not change throughout the sampling phase of each subject. The independent variable will change during the sampling phase providing a different distribution for each variable. Independent variables of the main study are: Cytogate, CluStic, and BFS. Dependent variable is the measurements of visualization efficiency (Huang, Eades, & Hong, 2006) which is computed by the equation:

$$\text{visualization efficiency} = \frac{Z_{RA} - Z_{ME} - Z_{RT}}{\sqrt{3}}$$

In the above equation,  $Z_{RA}$ ,  $Z_{ME}$ , and  $Z_{RT}$  are standardized z-scores of the response accuracy, mental effort, and response time. The term visualization efficiency measures the cognitive gain (response



accuracy) versus the cognitive cost (mental effort, response time), thus giving the reading easiness of a drawing.

### 5.1 Initial exploration study

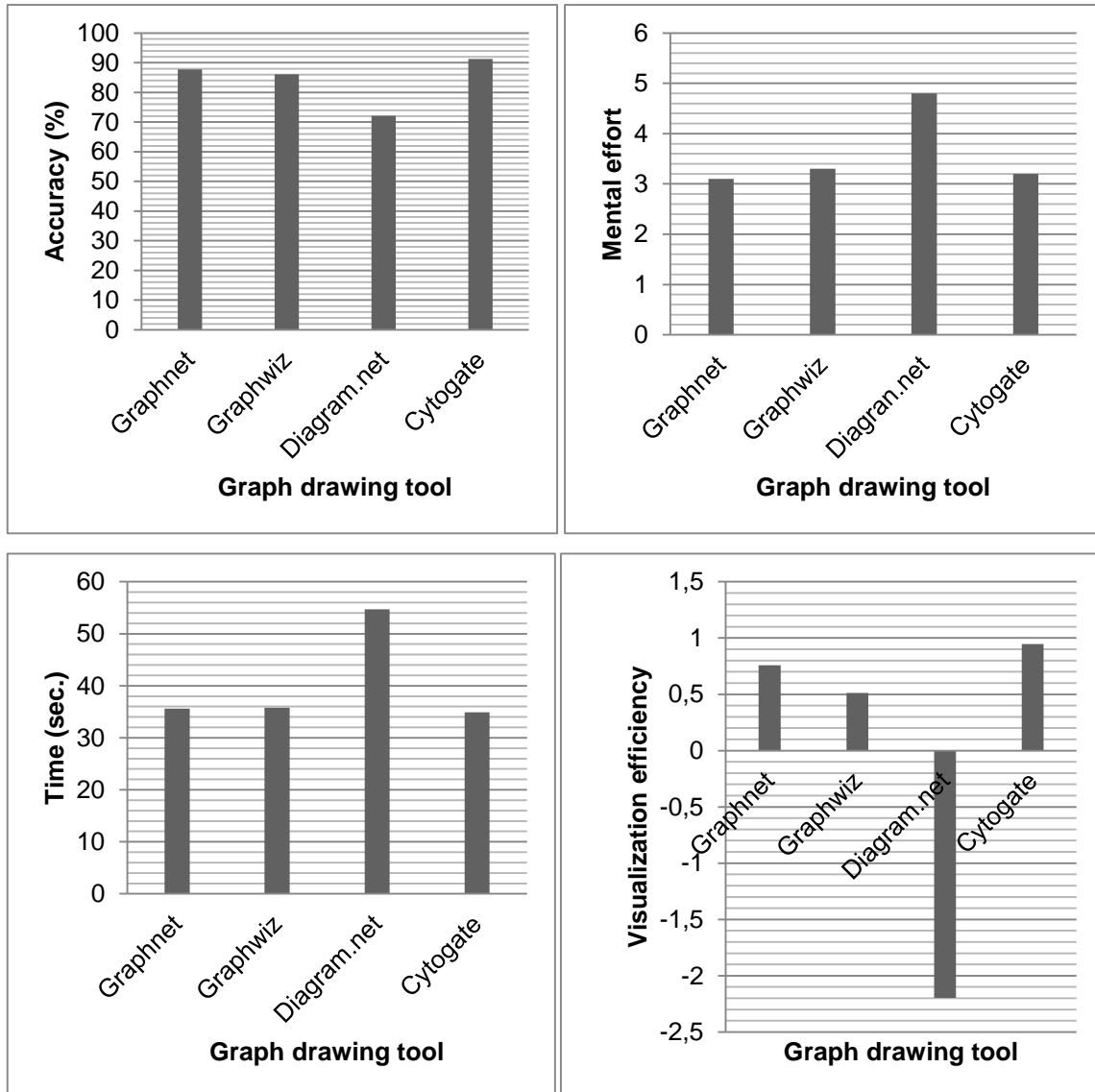


Figure 23 – Averages of accuracy, mental effort, response time and visualization efficiency.

Figure 23 show averages of each measured variable for every graph tool used in the initial exploration study. Graph drawings from the initial exploration study are presented in Appendix A, B, C, and D. The lower-right figure represents *visualization efficiency*. As seen, Cytogate produces drawings which results in higher visualization efficiency, i.e. a higher cognitive gain with regards to cognitive cost. The visualization efficiency result is not surprising given that Cytogate produces best values in 2 out of 3 tested variables. Subjects spent almost identical time, around 35 seconds on drawings made by top three performing tools, yet Cytogate produced drawings which on average resulted in 5,7% better accuracy. Diagram.net produces the least satisfying results out the four tools. Cross checking with images in appendix A of drawings produced by Diagram.net shows an obvious correlation between number of edge crossings and worse drawings.

## 5.2 Main study

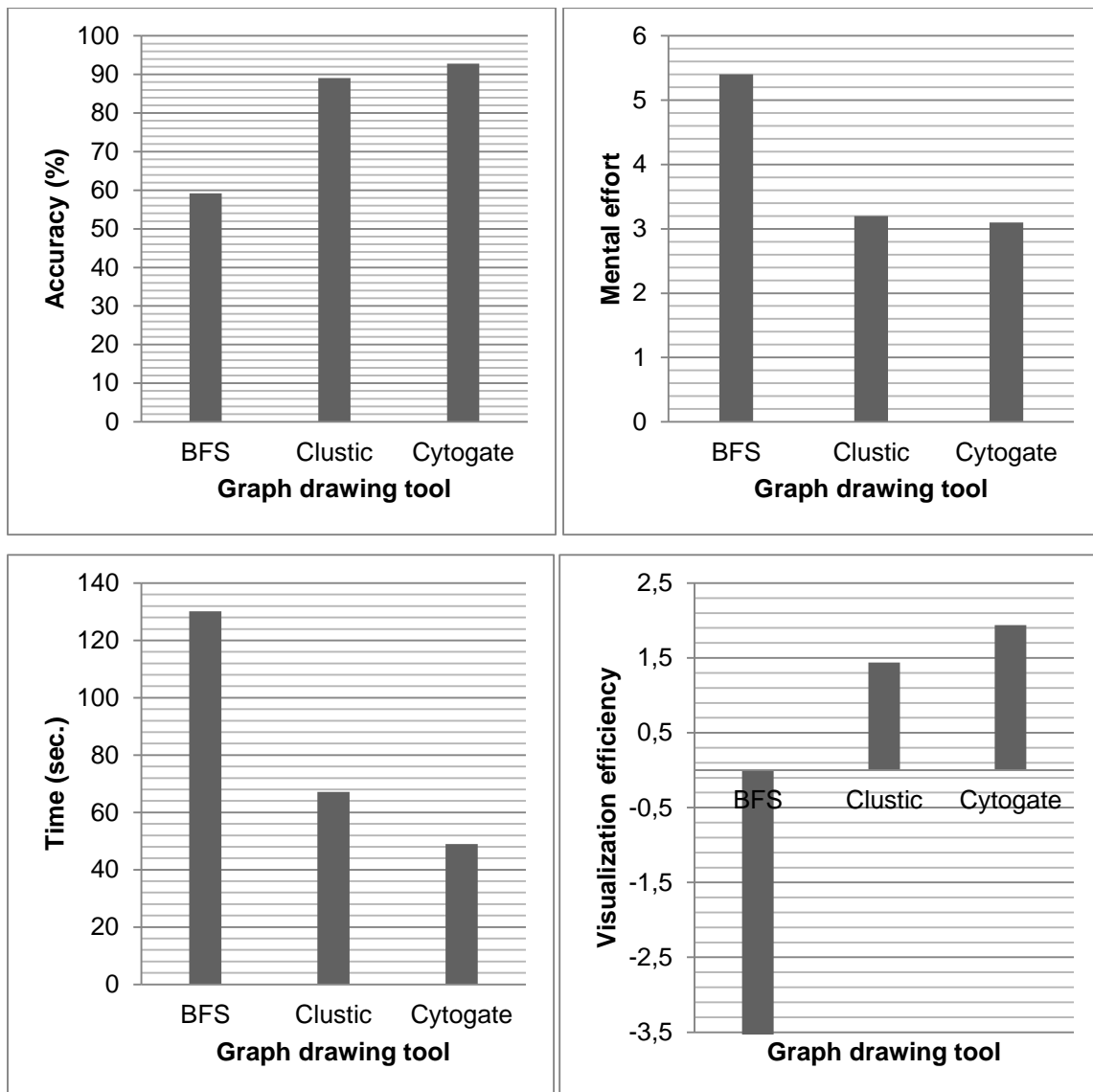


Figure 24 - Averages of accuracy, mental effort, response time and visualization efficiency from main study.

Illustrations in figure 24 display averages of each variable for every graph drawing tools used in the main study. Drawings from the main study are presented in Appendix D, E, and F. It can be seen that all measurements are in favor of CluStic in comparison to BFS, both are however outperformed by Cytogate. In particular subjects received an average accuracy of 89% with drawings made by CluStic compared to the 59% received from BFS, which is 34% higher accuracy in favor of CluStic. Subjects spent 67.1 seconds on average with drawings made by CluStic, which is 48.4% faster compared to the average time of BFS. On average, the amount of mental effort used on drawings made by CluStic were 3.1, meaning drawings made by CluStic were assessed to be 41% easier to interpret than drawings made by BFS. The cognitive gain with regards to cognitive cost (visualization efficiency) was on average 1.4 for CluStic and -3.8 for BFS.

On average, drawings made by Cytogate were 3.1% easier to interpret using 36% less time while achieving 4.3% better accuracy. The average visualization efficiency for Cytogate was 1.9 compared to 1.4 for CluStic.

Standard deviations for visualization efficiency were: BFS 1.1, CluStic 0.6, and Cytogate 0.8, small values across all graph drawing tools. This leads us to believe that the observed sample distributions have a low amount of uncertainty and that the differences in means are more likely to occur.

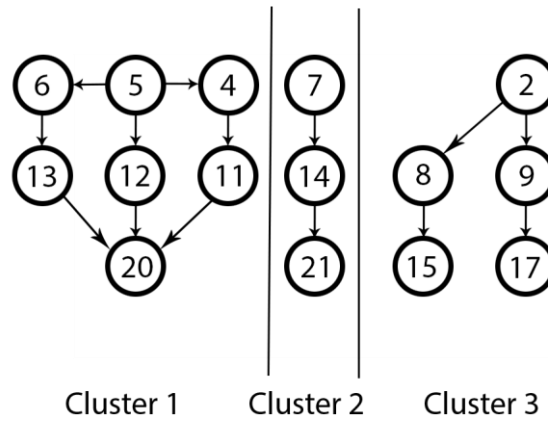


Figure 25 - Pre-existing clusters that must be preserved.

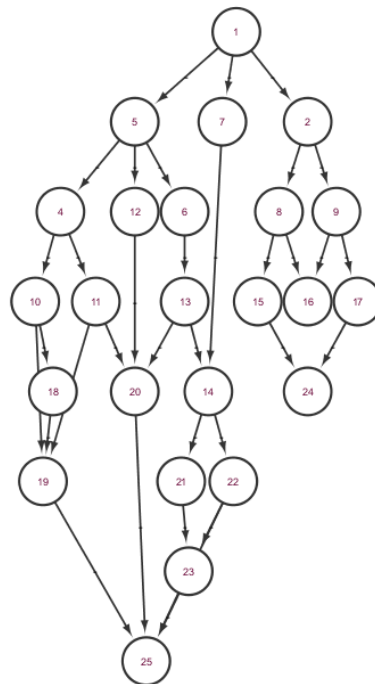


Figure 26 - Cytogate cannot preserve clusters

In figure 26 we can observe that clusters in figure 25 are not preserved as Cytogates optimizes the placement of vertices with regards to aesthetic criterion. This causes vertex 7 and 5 to be placed in different layers in regards to their position within their cluster. A second side effect is the introduction of vertex 16 into the internal structure of *Cluster 3*(figure 25). Optimal placement of vertex 16 with regards to edge crossing minimization criteria will move vertex 16 in between vertex 15 and 17 even though it is not a part of *Cluster 3*.

In figure 2 we see the benefits of CluStic, the algorithm preserves the clusters, without introducing changes to them as in the drawing by Cytogate. The drawing produced by Cytogate contains no edge crossings, thus reaching a lower visualization efficiency score. As seen in figure 2 neither vertex 5 or 7 are introduced to different layers compared to their cluster layering. Furthermore the placement of vertex 16 is outside of *Cluster 3* keeping the internal undisrupted.

## 6 DISCUSSION AND CONCLUSION

---

### 6.1 Discussion

In this section we will discuss the method used when implementing CluStic and designing the user studies. Furthermore we discuss the results retrieved from testing of both the CluStic implementation and popular public drawing tools.

#### 6.1.1 Initial exploration study

A subject spent around 15 min participating in the test, given they were not allowed any breaks we assess this duration to be reasonable. If the study were to present 40 drawings instead of 20, doubling the test duration we believe more students would hesitate to participate in the test. An increase in data due to an increase in the number of drawings is beneficial but not crucial. The sampled values of 5 drawings per tool (appendix A, B, C, and D) provided sufficient information about the quality of drawings.

The evaluation program worked great due to its simple design, with no extra features subjects were focused on the actual task. Subjects were not informed of the timer running between different drawings, by doing so we hoped to remove the element of time pressure. We believe this led to subjects taking the actual time needed for a task. One noticeable defect arose during the testing phase, textboxes (boxes in the GUI, in which subjects types values) differed in their layout; the accuracy text box could take 2 numbers as input whereas the box for mental effort could only take 1 number. When filling in the answer for accuracy subjects were accidentally focusing the second number position of the box instead of the first. This led to subjects moving the cursor to the correct position losing them around 2 seconds each time.

We did not include any type of progress indication in the evaluation program, by doing so we assumed that a subject would be more focused on the task. However, this was something that many subjects mentioned, subjects felt that it is important to know how many graphs are left, quoting “since even around 15 min may feel like an eternity when you are in a stressful environment”. Thus the effect of not displaying any progress indication may be the opposite of what we expected at design time.

Drawings from each individual tool almost always use a unique design scheme, leading to drawings with different colors and shapes for vertices and edges. We assess that there is a possibility that some individual design schemes are more preferable than others leaving some drawings easier to interpret, thus skewing results in their favor.

Before the study took place we assumed a victory for GraphWiz (appendix B) and a clear loss for Diagram.net (appendix A). Given the results we conclude they were not quite what we expected, having three tools with such similar results were surprising. Following from this, results may only differ because of the subjective interpretation of each individual design scheme. Results show in correlation with the drawings presented in appendix A, B, C, D, E, and F that drawings which are cluttered e.g. edge crossings, closely placed vertices and edges are much harder to interpret.

#### 6.1.2 The implementation

CluStic is constructed from set of heuristics which aims to satisfy aesthetic criteria (Purchase, Cohen, & James, 1996). Apart from those criteria CluStic aims to compute layouts of static components first and then place the remaining elements around their positions. Combining aesthetic criteria with static components is counterproductive. The aesthetic criteria heuristics depend on the constraints put on vertices by their edges. Forcing these heuristics to ignore some of the constraints breaks the normal structure leading to less satisfying layouts. CluStic solves this by re-constructing the important components (original clusters) into straight-line clusters which internally remembers the original cluster structure while letting other components be placed around it. This solution is somewhat good, since CluStic ignores the original position of elements in a cluster after they have been “inhaled”,

surrounding elements may be wrongly positioned. However, this will always be the scenario, keeping a component structure static will force the position of other elements to more unsatisfying positions.

The “exhale” phase of the breathing step expands a cluster back to its original structure forcing the repositioning of components which share the same layers. The *box-size* value used in the exhale phase is a rectangle area and thus not optimal with regards to vertex placement inside a cluster. Even though the *box-size* property makes sure a cluster never expands beyond its own limits, free positions in a layer maybe occupied by the *box-size* area. One improvement would be to change the *box-size* area to a custom shape, formed after the internal structure of a cluster leading to a more optimal shape and thus less likely to occupy free positions in the graph. This would further improve the alignment of vertices not in clusters since free positions may not be blocked by the *box-size* area anymore. At the moment CluStic only positions the source and sink of a cluster, ignoring a more dynamic placement of vertices in-between the source and sink of a cluster. If a cluster could maintain its original shape (no *breathing* step) alignment of non-cluster vertices would be better.

CluStic forces repositioning of elements in the graph depending on internal structures leaving the resulting drawings (appendix E) of less quality than drawings produced by normal graph drawing tools. We started with the idea that static components could improve the overall layout. The implementation and the results shows that CluStic can produce drawings of good quality (appendix E), CluStic can however not compete with a normal layered graph drawing tool like Cytogate (appendix D). Since ST-graphs are usually directed, a hierarchical structure is enforced leaving cluster vertices somewhat related to each other initially. This means the difference between normal graph drawing tools (Cytogate etc) and CluStic is less, leading to normal graph drawing tools being accepted over layouts which keep cluster structure (CluStic).

CluStic produces drawings of lower quality when the number of clusters is large and the clusters are of small size compared to fewer in number and of large size. This is however not surprising since more clusters will enforce a larger number of non-movable components in graphs leading to more aesthetic properties being ignored.

### 6.1.3 Main study

The choice of task to use in the evaluation of the implementations started out with three possible tasks; aesthetic criteria, cluster identification, and the longest path. To only consider the results from aesthetic criteria is not fair since aesthetic criteria are ignoring the “human factor” which is a big part of CluStic. A second approach was tried, in which two test-subjects were given the task of identifying clusters in a graph. When non-cluster vertices were introduced to the graph the results of the two test subjects differed greatly and it was deemed that this task would be too subjective to provide qualitative data. The main goal of the thesis was to evaluate whether or not cluster effect the quality of a graph layout. Since “the longest path” is a well-tested task and graph transversal is a common measurement to test the readability of a graph, this seemed like the most appropriate task to use.

One could then ask, “Does not this evaluation task ignore clusters?” As stated in previous paragraph the initial thought of a task was to have a subject identify clusters in the graph. Evaluation of CluStic would in such way certainly highlight its strengths, but since this type of evaluation method were too subjective they were deemed inappropriate. The task of finding the longest path provides subjects with the opportunity to assess their own ability as well as providing objective measures to the author about cognitive gain and cognitive cost. Furthermore it can be concluded that “mental effort” which is a subjectively measured value introduces, if assessed badly noise into the visualization efficiency formula. Given the individual results most subjects assessed their mental effort close to the distribution mean. “Response time” results also confirm that the estimated time for each task was properly chosen leading to a total test duration which did not cause subjects to lose focus.

The main study was a copy of the initial exploration study, however, in the main study we changed the evaluation program to also display the progress of the test. Introducing a progress counter may have reduced noise due to subjects thinking about how long they have left. On the downside subjects still complained about the textboxes which still contained the “bad focus problem” from the initial exploration study. At the introduction of new graph drawings (appendix E, and F) subjects highlighted

that vertices of round shape are easier to interpret than squared shaped. This falls under the “uniform color and design scheme” problem of the initial exploration study, since CluStic uses squared shaped vertices it may have impacted the result of CluStic negatively.

Looking at the results of the main study we could see, not to surprisingly, how bad BFS (appendix F) performed. Our expectations of CluStic in the beginning made us believe the tool achieved bad results. But throughout the implementation part of this thesis, given the constraints static components adds to a layout we think CluStic (appendix D) produced drawings of satisfying quality.

This becomes apparent when we look the results, CluStic scored a 89% accuracy closely following Cytogate at 92%. What is more surprising is perhaps the fact that there was only a 0.1 difference in the mean of mental effort. This means subject assessed drawings of CluStic and Cytogate to be almost equally difficult. On the other hand, the fact that Cytogate produce drawings which on average have 0,5 better visualization efficiency proves that subjective measurements such as mental effort have obvious drawbacks. How easy subjects “think” a task is compared to how hard it “really” is may differ greatly. The largest difference in sample distribution means between CluStic and Cytogate occurred in the “response time” chart, Cytogate outshined CluStic with 36% lower response time. Since the accuracy of both algorithms was high, a low response time would give an objective measure on how difficult a task actually was, ignoring the self-assessment (mental effort) of a user. A low standard deviation means less spread of values, leading to more uniform sample distributions. Low standard deviation could therefore indicate that the chance of drawing a sample value which largely differs from the mean is less. The low standard deviation of both Cytogate and CluStic together with a 0,5 increase of visualization efficiency in favor of Cytogate, leaves us with the conclusion that the difference in sample distribution means between Cytogate and CluStic is likely enough to be considered real. We can therefore with good confidence say that Cytogate produces drawings with higher quality than CluStic and that clusters does not affect the layout in a positive way.

Looking at the results from BFS, there is not really much to be said about the quality of the drawings. CluStic produced drawings with a visualization efficiency of 1,4 compared to the -3,8 of BFS, the difference was 0,5 between CluStic and Cytogate. We cannot be certain the observed difference was all due to the impact of clusters. Since CluStic is designed to compete with the best algorithms, it implements some of the most popular aesthetic criteria solving heuristics. This may introduce noise in the results of the drawing tool and we cannot ignore this when reaching a conclusion. But we observe a higher mean in terms of visualization efficiency in layouts made by CluStic than BFS, we just cannot assume this is all due to Clusters. As with Cytogate and Clustic, the standard deviation is low for BFS leading us to believe the values are likely to occur and that chance did not play a part in the bad results of BFS.

## **6.2 Conclusion**

In this part of the thesis we present our conclusions on the result, method, implementation and discussion. We highlight important findings and drawbacks of the different techniques used throughout the thesis.

### **6.2.1 Initial and main study**

From the results we could see that the task of longest path used in the evaluation works in the sense of not being too biased towards CluStic. As discussed there are other evaluation methods which could highlight the benefits of CluStic, these methods are however subjective and does not give enough qualitative data.

Design of the evaluation program was too simple, subjects expressed the need to see progress throughout the testing phase. Further the color and shape scheme of vertices and edges in each individual graph drawing tool must be uniform in order to fully remove the noise which different designs may introduce.



### **6.2.2 The implementation**

CluStic was outperformed by Cytogate, given the extra constraints static components bring to a layout CluStic performs well. Results show that BFS should never be considered an alternative, a layout from CluStic or Cytogate outperforms BFS in every dependent variable of the main study. Obvious flaws of CluStic are closely related to the partial heuristics used, but not all is caused by the use of heuristics. Many problems arise due to non-optimal placement of clusters, and the introduction of the breathing step which causes some non-cluster vertices to be wrongly positioned. The results show that static components have a negative effect on the quality of graph drawings. If internal structures of clusters are a necessity CluStic produces feasible layouts for sparse graphs. However for graphs which are directed and where the constraint of clusters can be ignored we recommended using a normal layered graph drawing tool.

## **6.3 Recommendations**

For the main study we propose 2 changes: the first would be to use a uniform color and design scheme. This would make it less beneficial for the tools using a more interpretable color and design scheme leading to reduced noise in the recorded data. Secondly, in order to achieve more accurate results the study should increase the number of test subjects.

The CluStic graph drawing tool can be improved if the following changes are made: CluStic should be able to automatically recognize clusters which are not pre-defined, this could lead to better focus on closely coupled elements in the graph. As a second improvement CluStic could make use of a more optimal ILP algorithm (section 2.3.2.3 Network Simplex algorithm), this would lower the number of dummy elements and possibly affect the quality and complexity of a layout. Finally, CluStic could use a hybrid approach which means that cluster could initially be placed in arbitrary sized drawing area and aligned by a force directed algorithm. Clusters are then mapped to their closest layer and remaining vertices are assigned their positions based on a layered algorithm.

## 7 REFERENCES

---

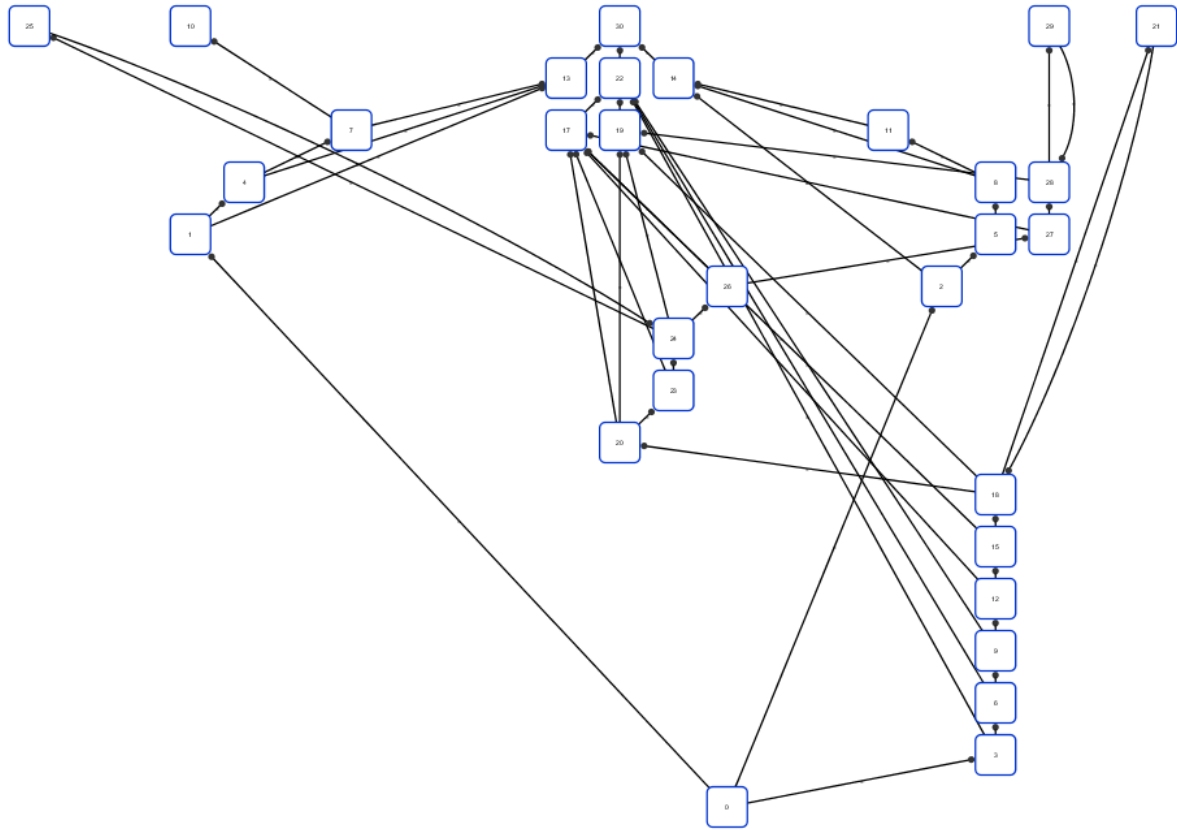
- 1 Aho, A. V., Garay, M. R., & Ullman, J. D. (1972). the transitive reduction of a directed graph. *SIAM Journal on Computing* , 1 (2), 131-137.
- 2 Bastert, L., & Matuszewski, C. (2001). Layered Drawings of Digraphs. In K. M., & D. Wagner, *Drawing Graphs*, (pp. 87-120). Berlin Heidelberg: Springer-verlag.
- 3 Berger, B., & Dhor, P. W. (1990). Approximation algorithms for the maximum acyclic subgraph problem. *Proceedings of the first Anual ACM-SIAM symposium on Distrete Algorithms*, (pp. 263-243).
- 4 Biedl, T. C., Madden, B. P., & Tollis, I. G. (1997). The three-phase method: A unified approach to orthogonal graph drawing. *Proceedings of Graph Drawing 97* (pp. 392-401). Springer-Verlag.
- 5 Brandes, U., & Köpf, B. (2002). Fast and simple horizontal coordinate assignment. *Graph Drawing: Proceedings of 9th Internation symposium, GD* , 2265, 31-44.
- 6 Carpano, M.-J. (1980). Automatic display of hierarchized graphs for computer-aided decision analysis. *IEEE Transactions on Systems, Man and Cybernetics* , SMC-10 (11), 705-715.
- 7 Chimani, M., Gutwenger, C., Mutzel, P., & Wong, H.-M. (2011). Upward planarization layout. *Journal of Graph Algorithms and Applications* , 15 (1), 127-55.
- 8 Dehaene, S. (2011). *The Number Sense: How the Mind Creates Mathematics, revised and updated*. Oxford, USA: Oxford University Press Inc.
- 9 Eades, P. (1984). A heuristic for graph drawing. *Congress Numerantium*, 42, (pp. 149-160).
- 10 Eades, P., & Kelly, D. (1986). Heuristics for reducing crossing in 2-layered networks. *Ars combinatoria* , 21-A:89-98.
- 11 Eades, P., & Lin, X. (1989). How to draw a directed graph. *1989 IEEE Workshop on Visual Languages* (pp. 13-17). Rome, Italy: IEEE Comput. Soc. Press.
- 12 Eades, P., & Sugiyama, K. (1990). How to draw a directed graph. *Journal of Information Processing* , 13 (4), 424-437.
- 13 Eades, P., & Wormald, N. C. (1994). Edge crossings in drawings of bipartite graphs. *Algorithmica* , 11 (4), 379-403.
- 14 Eades, P., Lin, X., & Smyth, W. (1993). A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters* , 47 (6), 319-323.
- 15 Eades, P., Lin, X., & Smyth, W. F. (1989). *Heuristics for the feedback arc set problem*. Perth, Australia: Curtin University of Technology, School of.
- 16 Eiglsperger, M., Fekete, S., & Klau, G. Orthogonal graph drawing. *Drawing graphs. Methods and models (Lecture Notes in Computer Science Vol.2025)* , 2025, 121-171.
- 17 Eiglsperger, M., Fekete, S., & Klau, G. (2001). Orthogonal graph drawing. In M. Kaufmann, & D. Wagner, *Drawing graphs, methods and models* (Vol. 2025, pp. 121-171). Berlin, Germany: Springer-Verlag.
- 18 Eiglsperger, M., Siebenhaller, M., & Kaufmann, M. (2005). An Efficient Implementation of Sugiyama's algorithm for layered graph drawing. *Journal of Graph Algorithms and Applications* , 9 (3), 305-325.
- 19 Fruchterman, T., & Reingold, E. (1991). Graph drawing by force directed placement. *Software—Practice and Experience* , 1123-1164.

- 20 Fröhlich, M., & Werner, M. (1995). Demonstration of the interactive graph visualization system da Vinci. In M. Fröhlich, & M. Werner, *Graph Drawing* (Vol. 894, pp. 266-269). Springer Berlin Heidelberg.
- 21 Gansner, E., Koutsofios, E., & North, S. (2006). *Drawing graphs with dot*. Graphwiz.
- 22 Gansner, E., Koutsofios, E., North, S., & Vo, K.-P. (1993). A technique for drawing directed graphs. *IEEE Transactions on Software Engineerin* , 19 (3), 214-230.
- 23 Garey, M. R., & Johnson, D. S. (1979). *Computers and intaractability. A guide to the theory of NP-completeness*. Oxford, UK: Freeman.
- 24 Garey, M. R., & Johnson, D. S. (1983). Crossing Number is NP-Complete. *SIAM. J. on Algebraic and Discrete Methods* , 4 (3), 312-316.
- 25 Garg, A., & Tamassia, R. (1996). A new minimum cost flow algorithm with applications to graph drawing. *Proceedings of Graph Drawing '96* (pp. 201-216). Berlin, Germany: Springer-Verlag.
- 26 Hadany, R., & Harel, D. (2001). A multi-scale algorithm for drawing graphs nicely. *Discrete Applied Mathematics* , 113 (1), 3-21.
- 27 Hai, T., & Zhihui, H. (2013). Network simplex algorithm for DAG layering. *2013 Fifth International Conference on Computational and Information Sciences (ICCIS 2013)* (pp. 1525-1528). Shiyang,China: IEEE.
- 28 Healy, P., & Nikolov, S. (2013). Hierarchical Drawing Algorithms. In R. Tamassia, *Handbook of Graph Drawing and Visualization* (pp. 409-453). CRC Press.
- 29 Herman, I., Melancon, G., & Marshall, M. (2000). Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics* , 6 (1), 24-43.
- 30 Herman, I., Melancon, G., & Marshall, S. (2000). Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS* , 6 (1), 24-43.
- 31 Huang, W., Eades, P., & Hong, S.-H. (2006). Measuring effectiveness of graph visualization: A cognitive load perspective. *Information Visualization* , 8 (3), 139-152.
- 32 Huang, W., Eades, P., Hong, S.-H., & Lin, C.-C. (2013). Improving multiple aesthetics produces better graph drawings. *Journal of Visual Languages and Computing* , 24 (4), 262-272.
- 33 Karp, R. M. (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations* , 85-103.
- 34 Kobourov, S. G. (2012). Spring Embedders and Force Directed Graph Drawing Algorithms. *arXiv:1201.3011* .
- 35 Lin, C.-C., & Yenb, H.-C. (2012). A new force-directed graph drawing method based on edge-edge repulsion. *Journal of Visual Languages & Computing* , 23 (1), 29-42.
- 36 Mayer, R., & Chandler, P. (2001). When learning is just a click away: Does simple user interaction fooster deeper understanding of multimedia messages. *Journal of psychology* , 95 (2), 390-397.
- 37 Mehlhorn, K. (1984). *Graph algortihms and NP-completness* (Vol. 2). Heidelberg, Germany: Springer-Verlag.
- 38 Paulisch, F., & Tichy, W. (1990). EDGE: an extendible graph editor. *Software - Practice and Experience* , S1/63-88.
- 39 Purchase, H., Cohen, R., & James, M. (1996). Validating graph drawing aesthetics . *proceedings of symposium on graph drawing, GD 95, Lecture notes in Computer Science*, (pp. 435-446). Springer-Verlag.

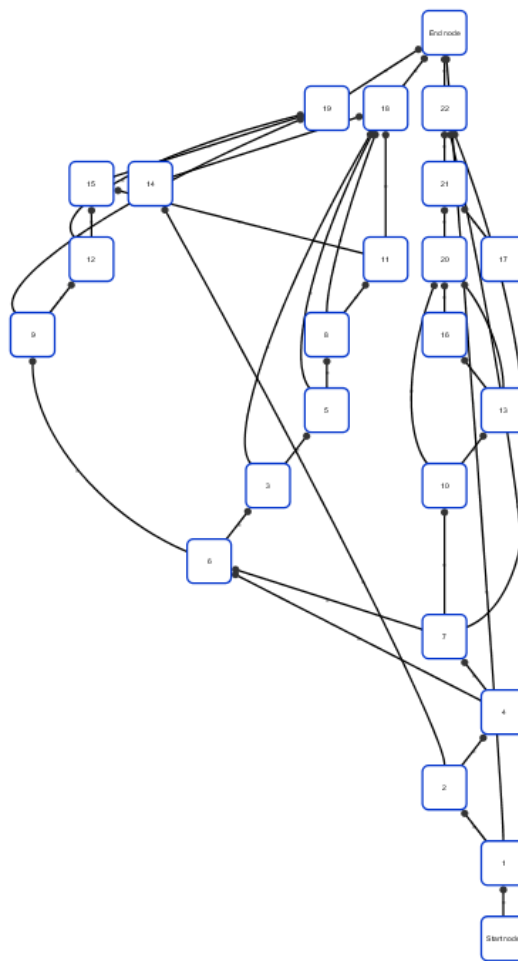
- 40 Sander, G. (1994). *Graph layout through the VCG tool, Technical Report A03/94*. Universität des Saarlandes.
- 41 Schrijver, A. (1986). Polyhedral proof methods in combinatorial optimization. *Discrete applied mathematics* , 14 (2), 111-133.
- 42 Sugiyama, K. (2002). Graph drawing and application for software and knowledge engineers. *Series on software engineering and knowledge engineerign* , 59-80.
- 43 Sugiyama, K., Tagawa, S., & Toda, M. (1981). Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetic* , SMC-11 (2), 109-125.
- 44 Tamassia, R. (1987). On Embedding a Graph in the Grid with the Minimum Number of Bends. *SIAM J. Computing* , 16 (3), 421-444.
- 45 Ware, C., & Mitchell, P. (2008). Visualizing graphs in three dimensions. *ACM Transactions on Applied Perception (TAP)* , 5 (1), 15.
- 46 Ware, C., Purchase, H., Colpoys, L., & McGill, M. (2002). Cognitive measurements of graph aesthetics. *Information visualization* , 1 (2), 103-110.
- 47 Warfield, J. (1977). Crossing theory and hierarchy mapping. *IEEE Transactions on Systems, Man and Cybernetics* , SMC-7 (7), 505-523.
- 48 Wilson, R., Graham, R., & Watkins, J. J. (2013). *Combinatorics: Ancient & Modern*. Oxford univeristy press.
- 49 Zhao, J. (2008). A sociogram analysis on group interaction in an online discussion forum. *Advances in Web Based Learning - ICWL 2008. 7th International Conference* (pp. 377-389). Jinhua, China: Springer-Verlag.
- 50 Zugal, S., Pinggera, J., Reijers, H., Reichert, M., & Weber, B. (2012). Making the Case for Measuring Mental Effort. *Proceedings of the Second Edition of the International Workshop on Experiences and Empirical Studies in Software Modelling*. New York, USA: ACM.

## APPENDIX A: DIAGRAM.NET

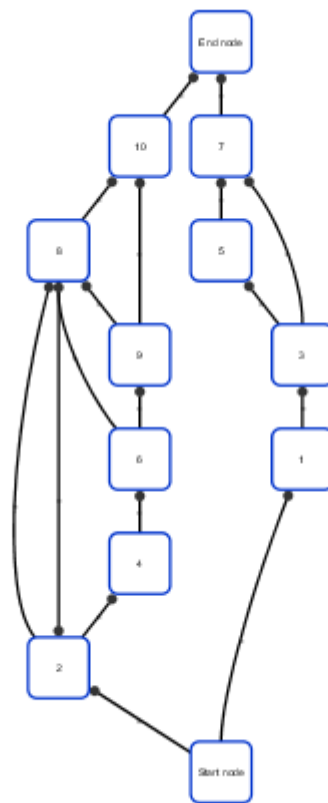
Graph 1



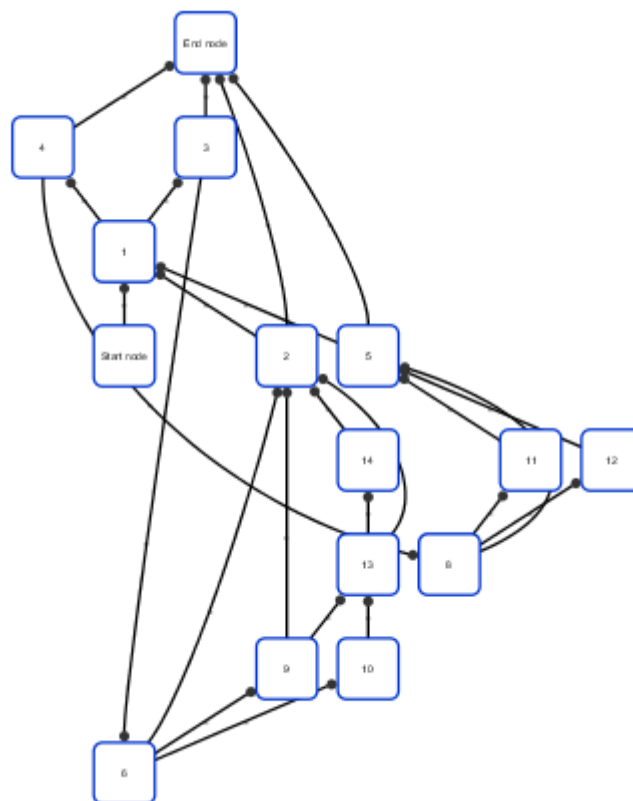
Graph 2



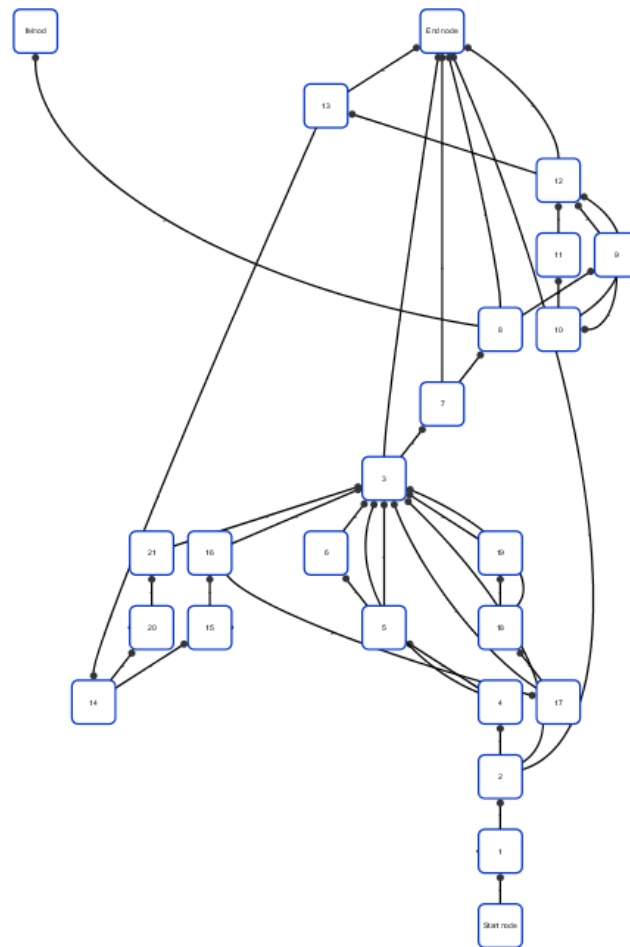
Graph 3



Graph 4



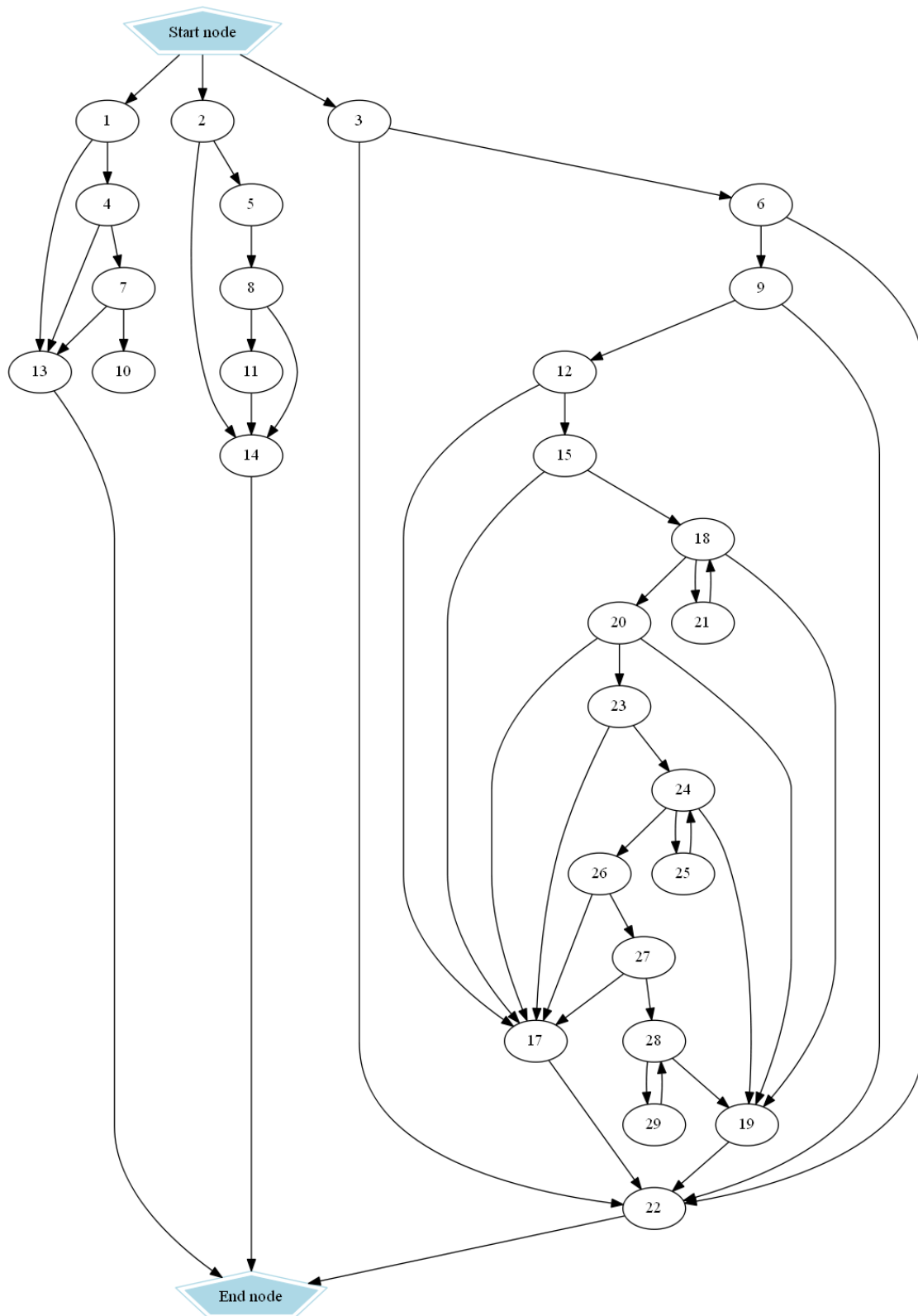
Graph 5



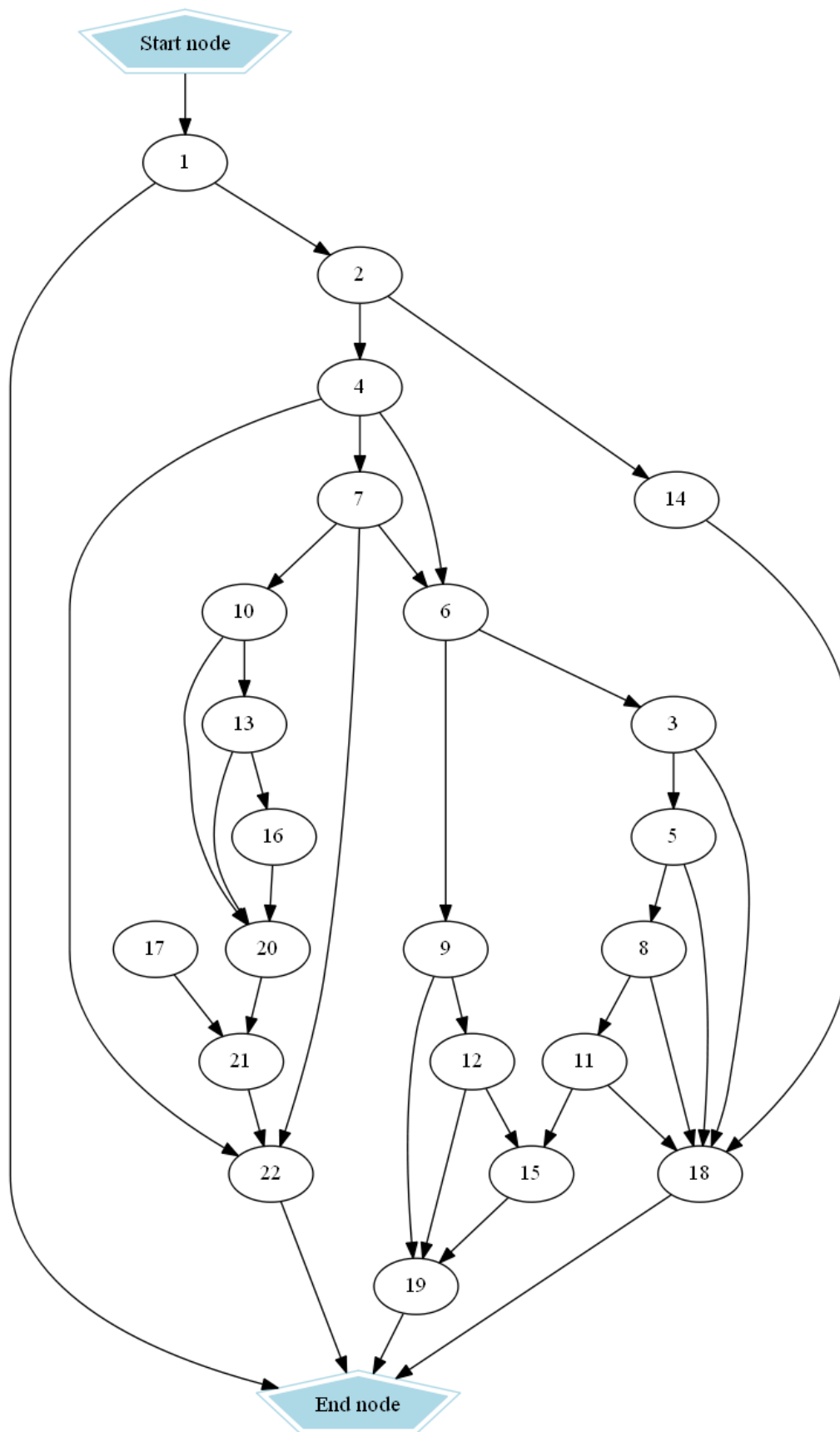


## APPENDIX B: GRAPHWIZ

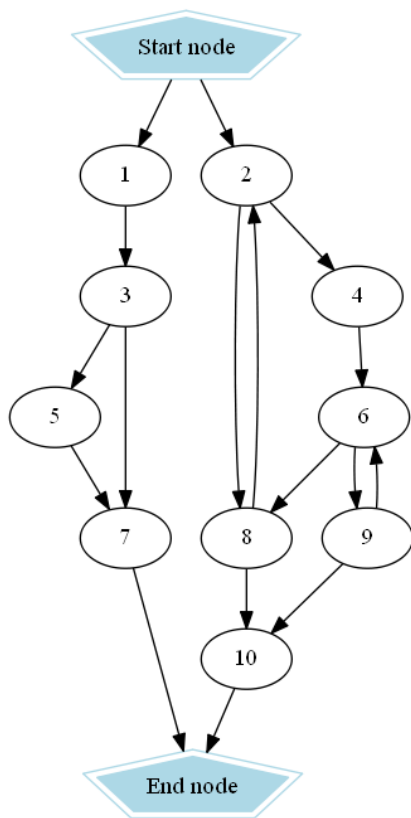
Graph 1



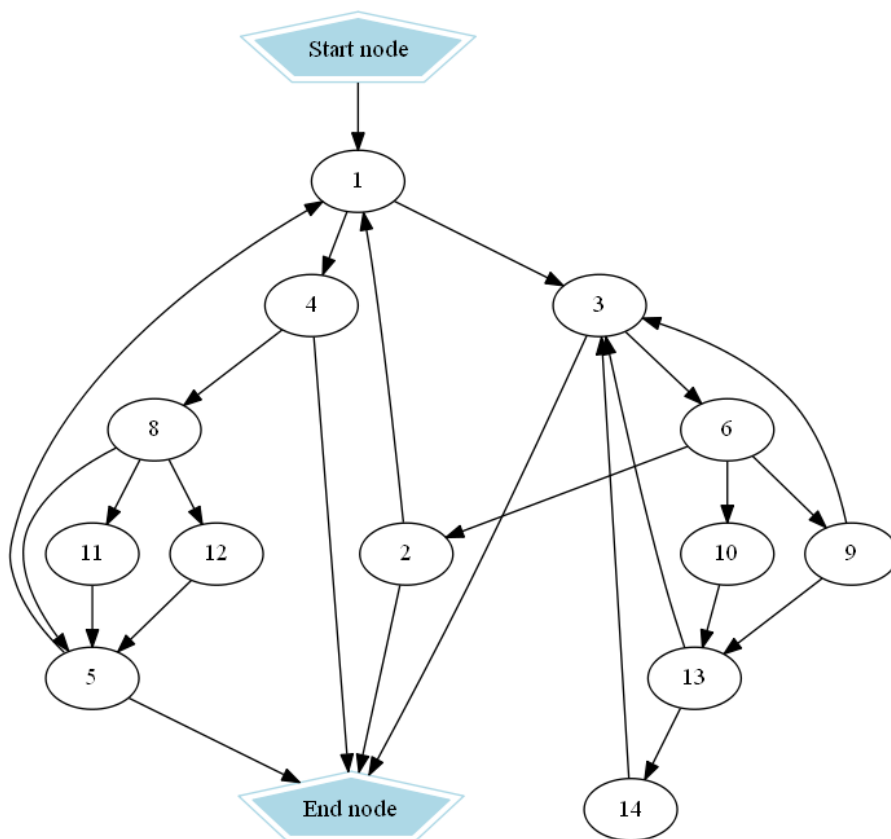
Graph 2



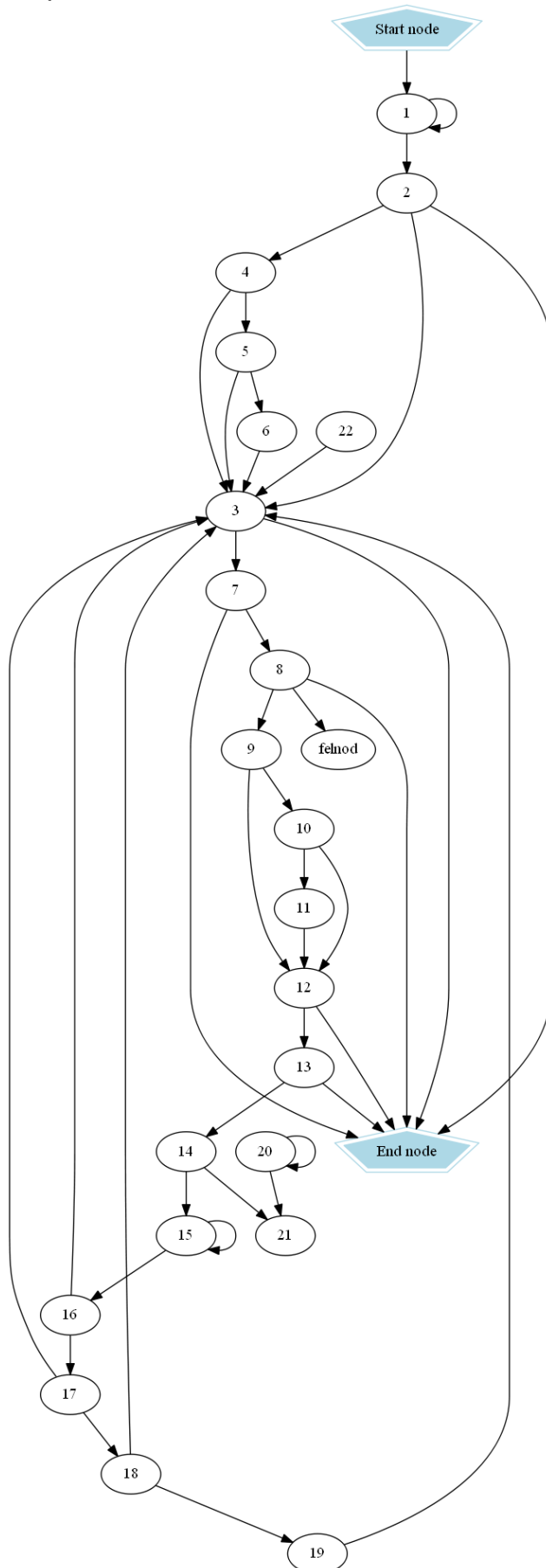
Graph 3



Graph 4

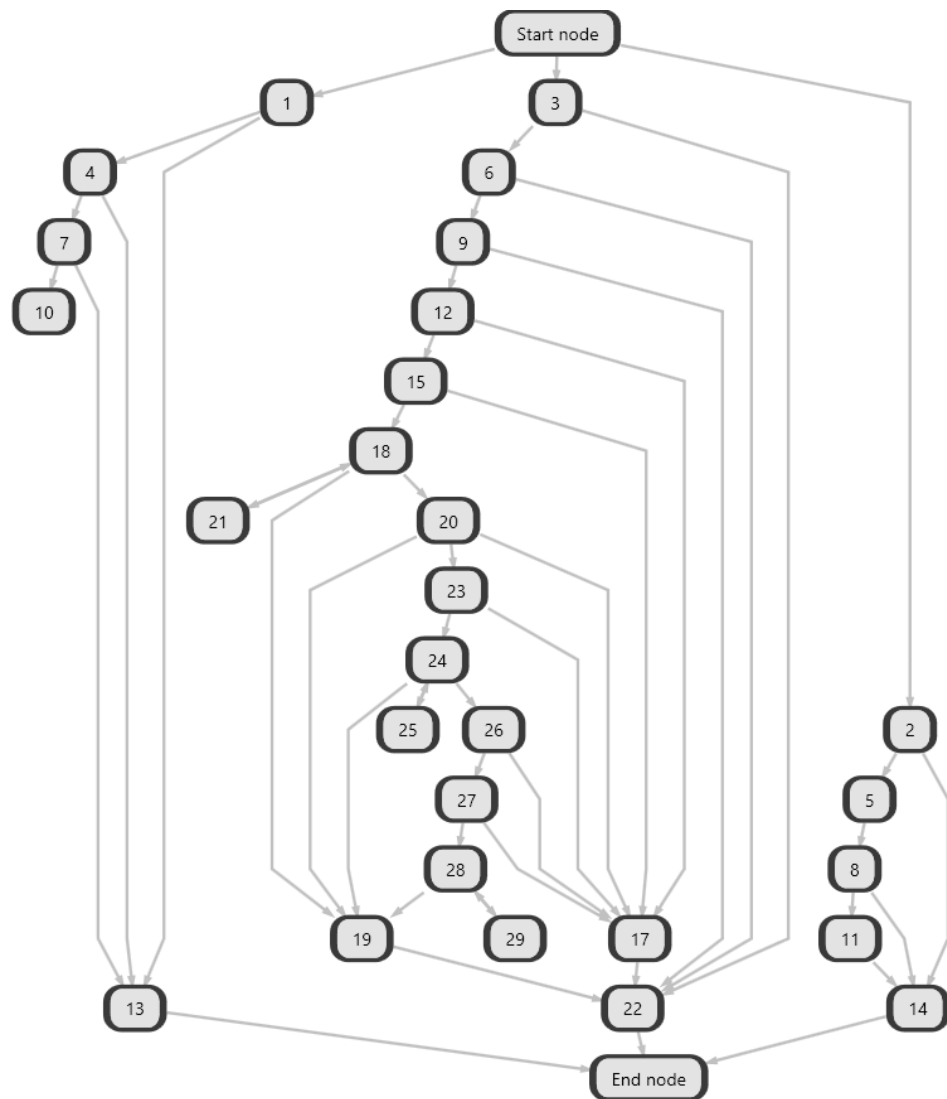


Graph 5

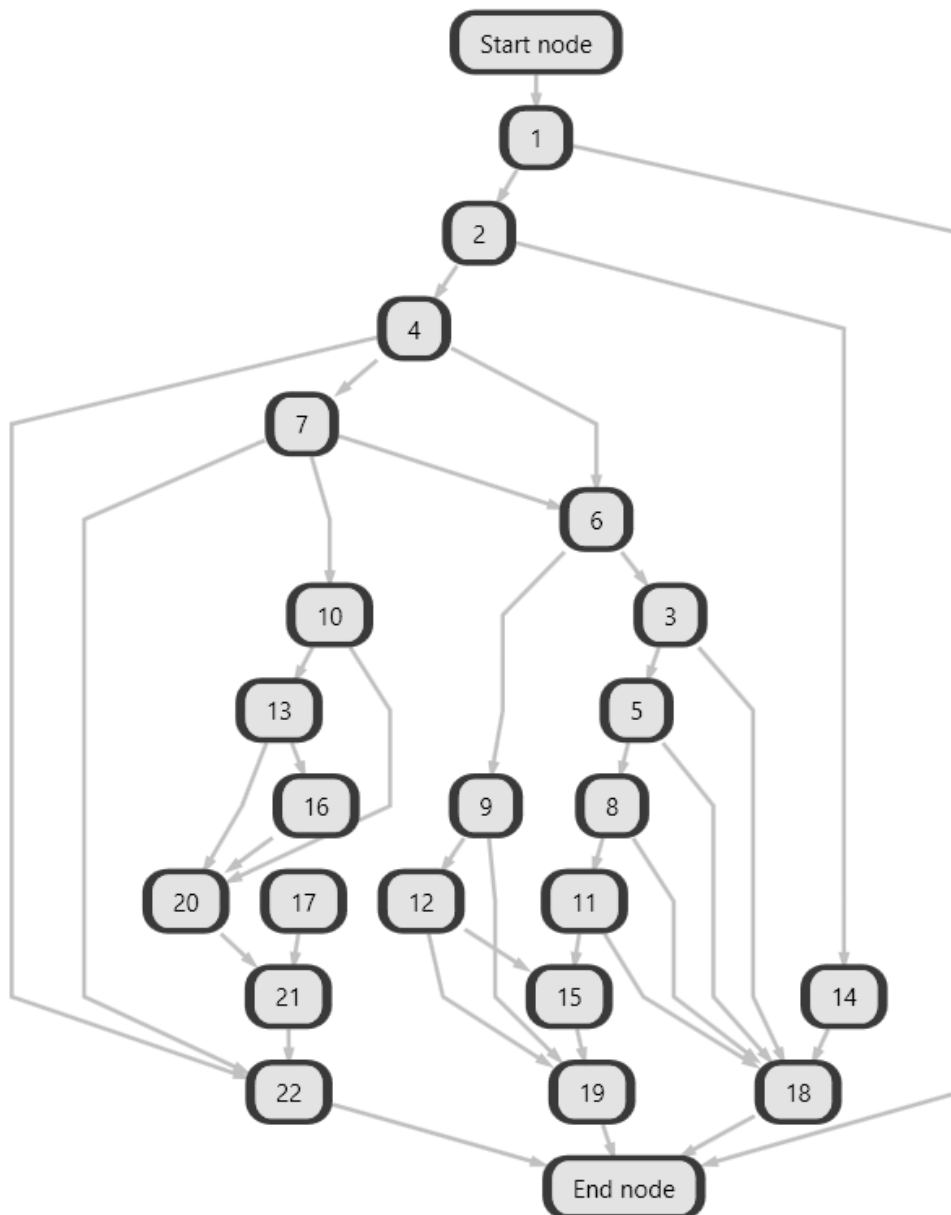


## APPENDIX C: GRAPHNET

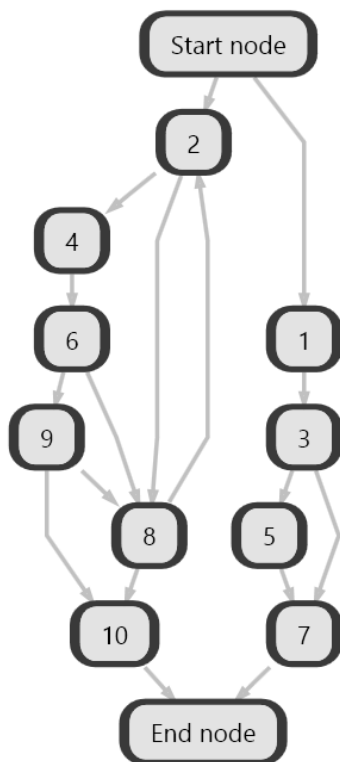
Graph 1



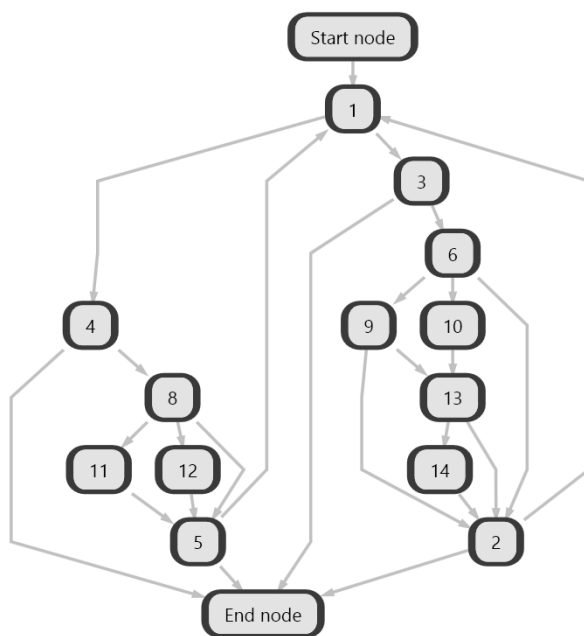
Graph 2



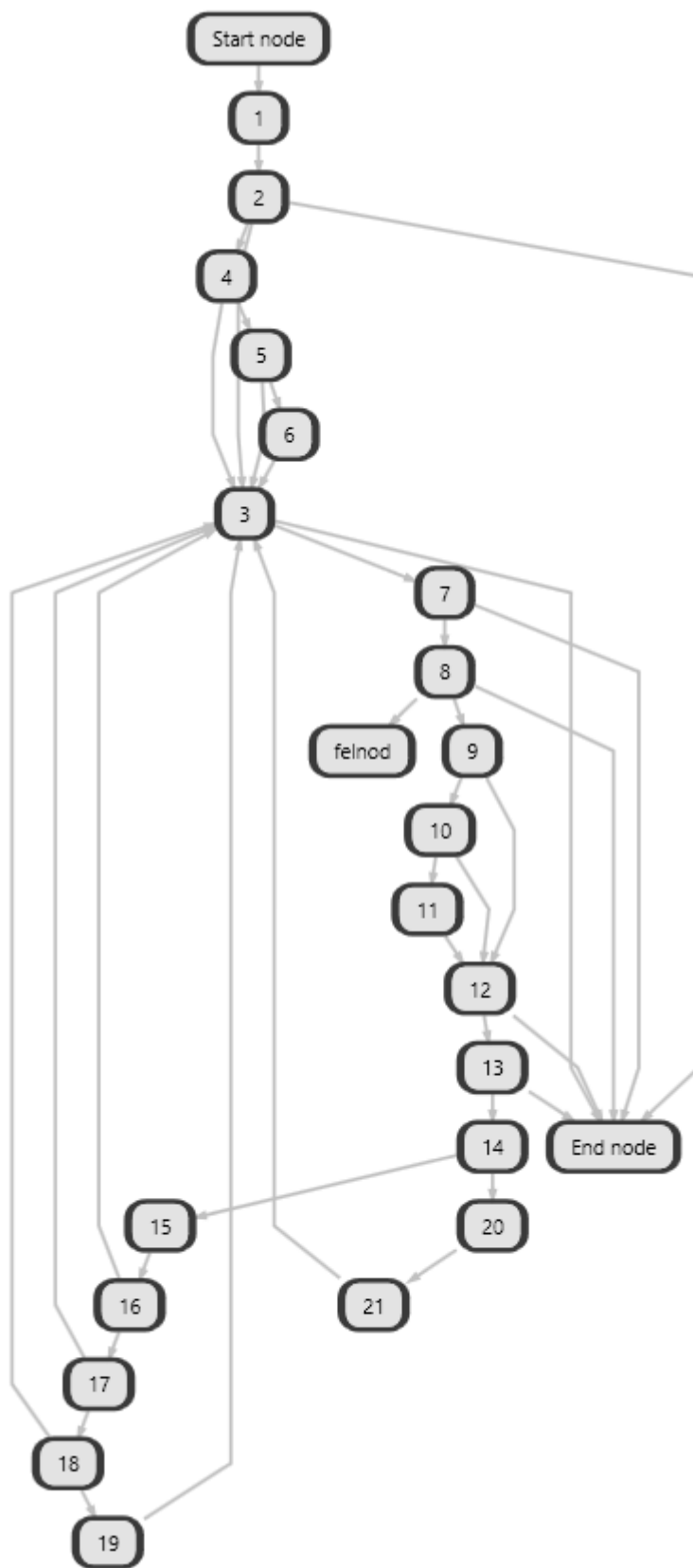
Graph 3



Graph 4



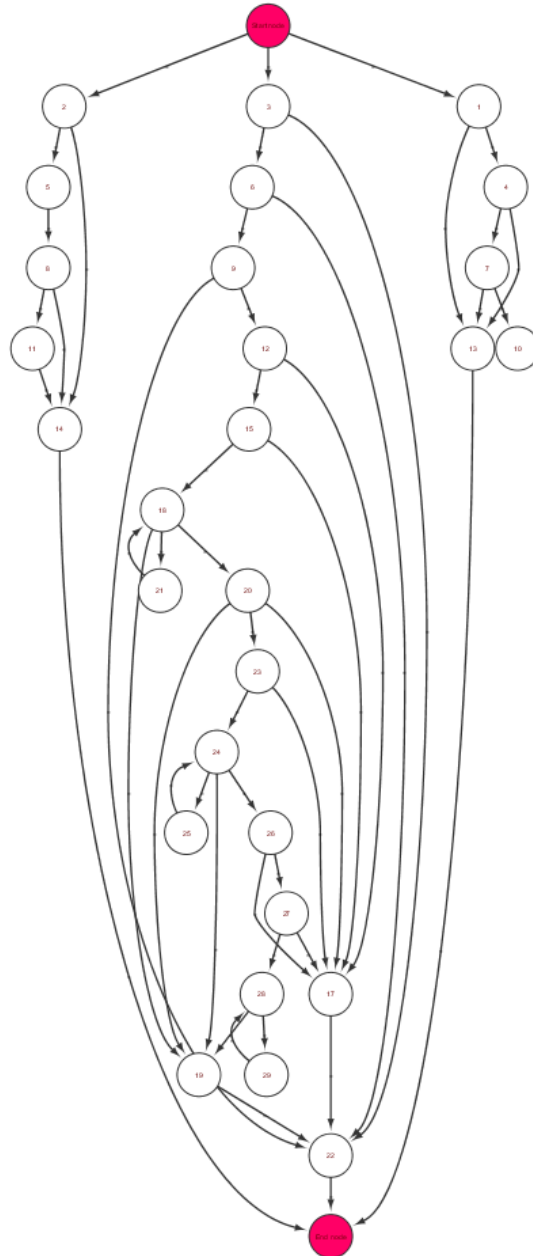
Graph 5



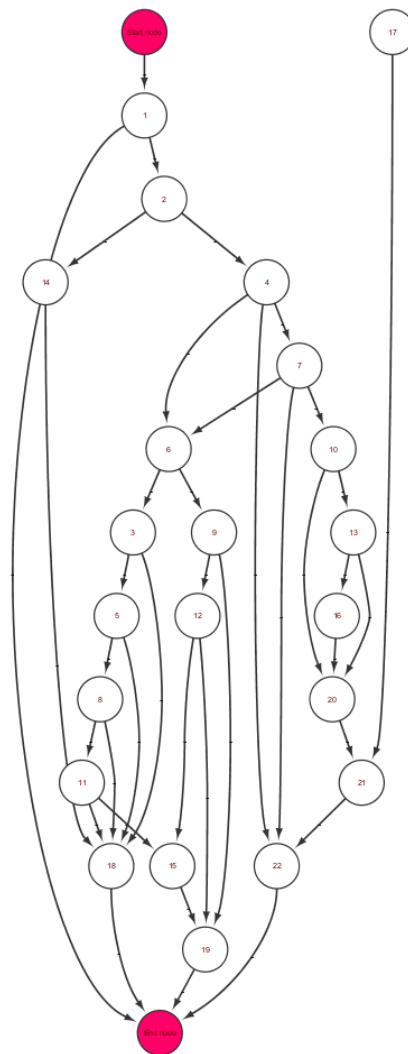


## APPENDIX D: CYTOGATE

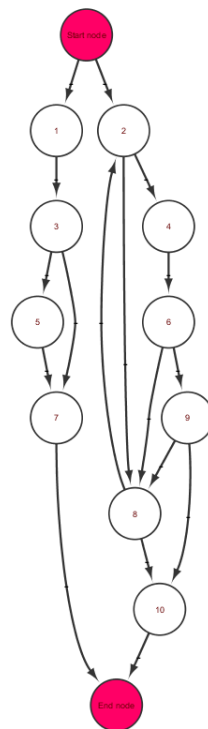
Graph 1



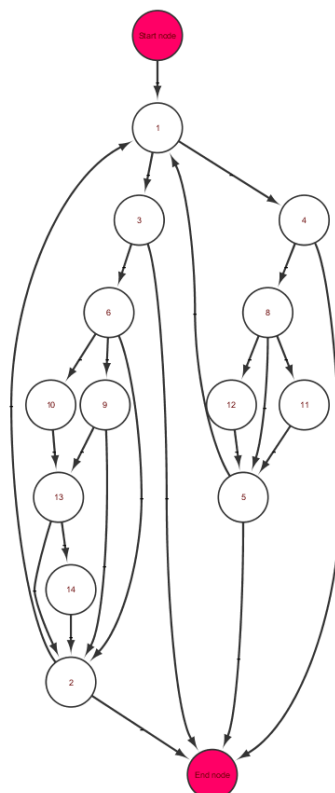
Graph 2



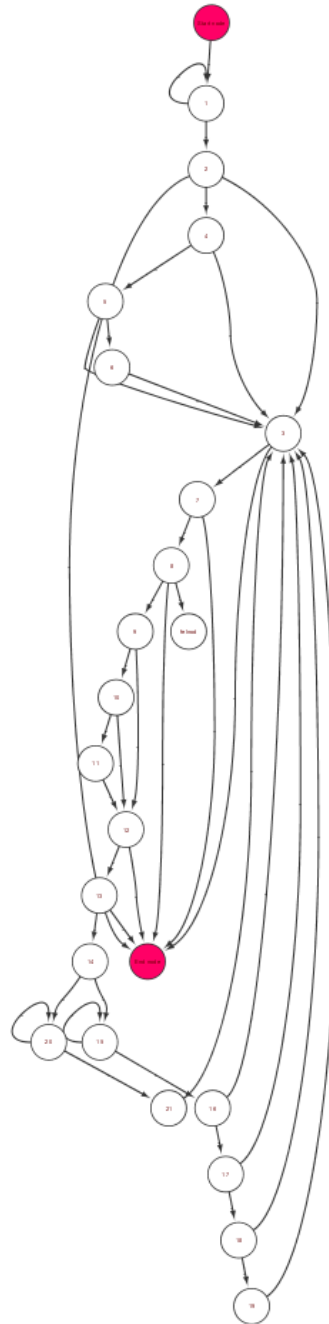
Graph 3



Graph 4

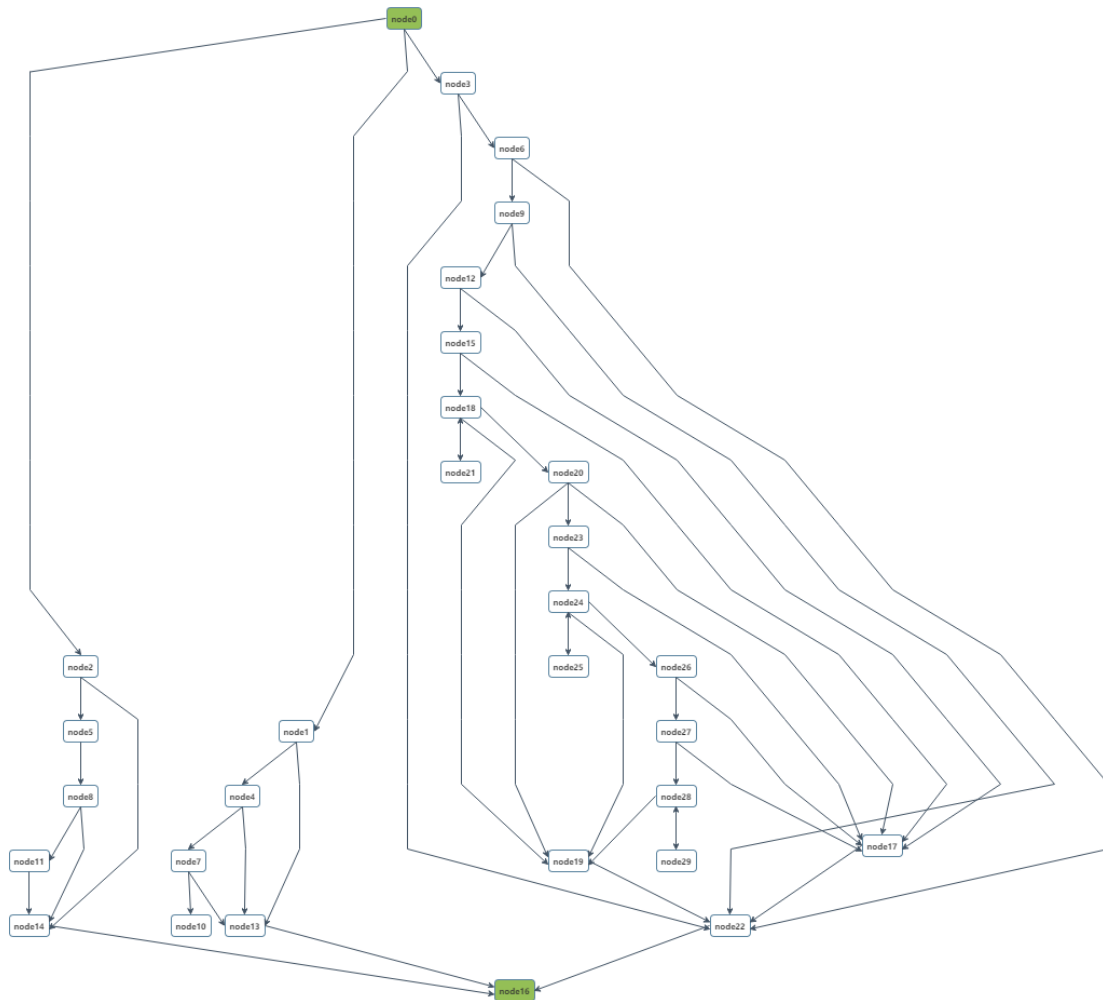


Graph 5

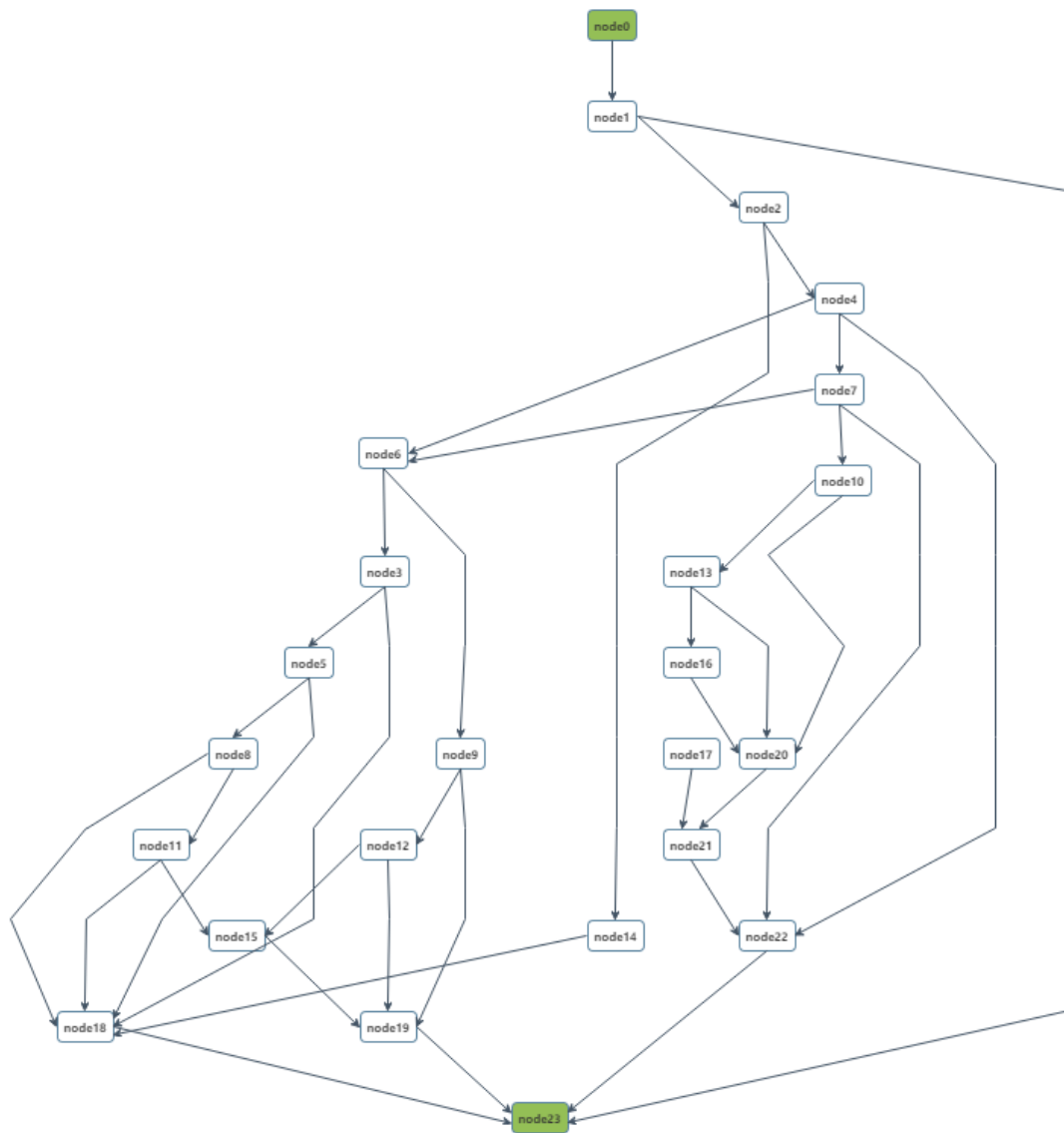


## APPENDIX E: CLUSTIC

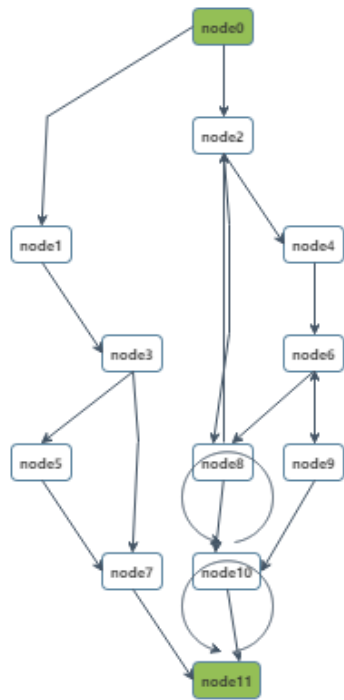
Graph 1



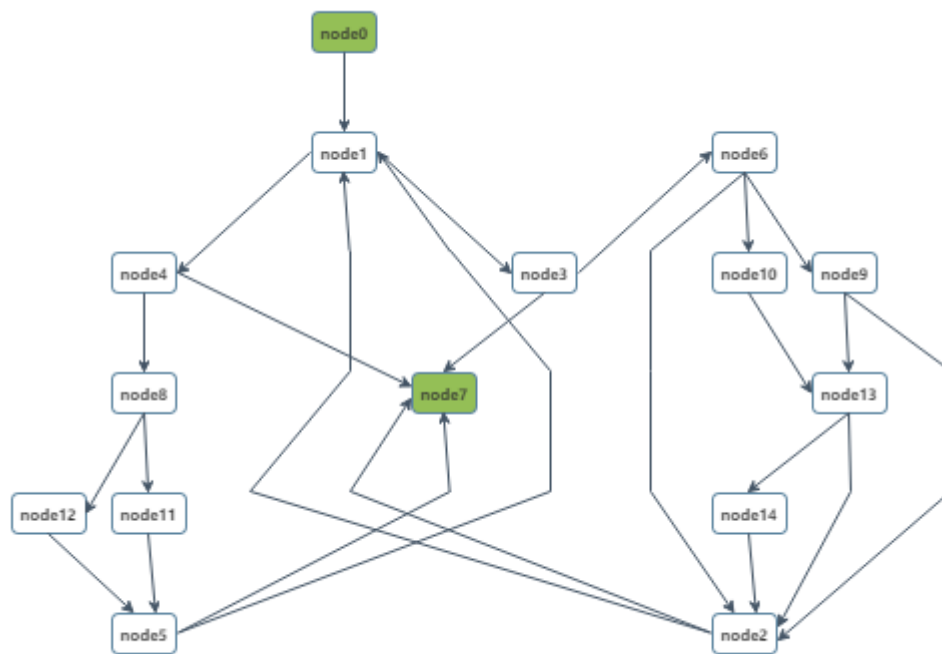
Graph 2



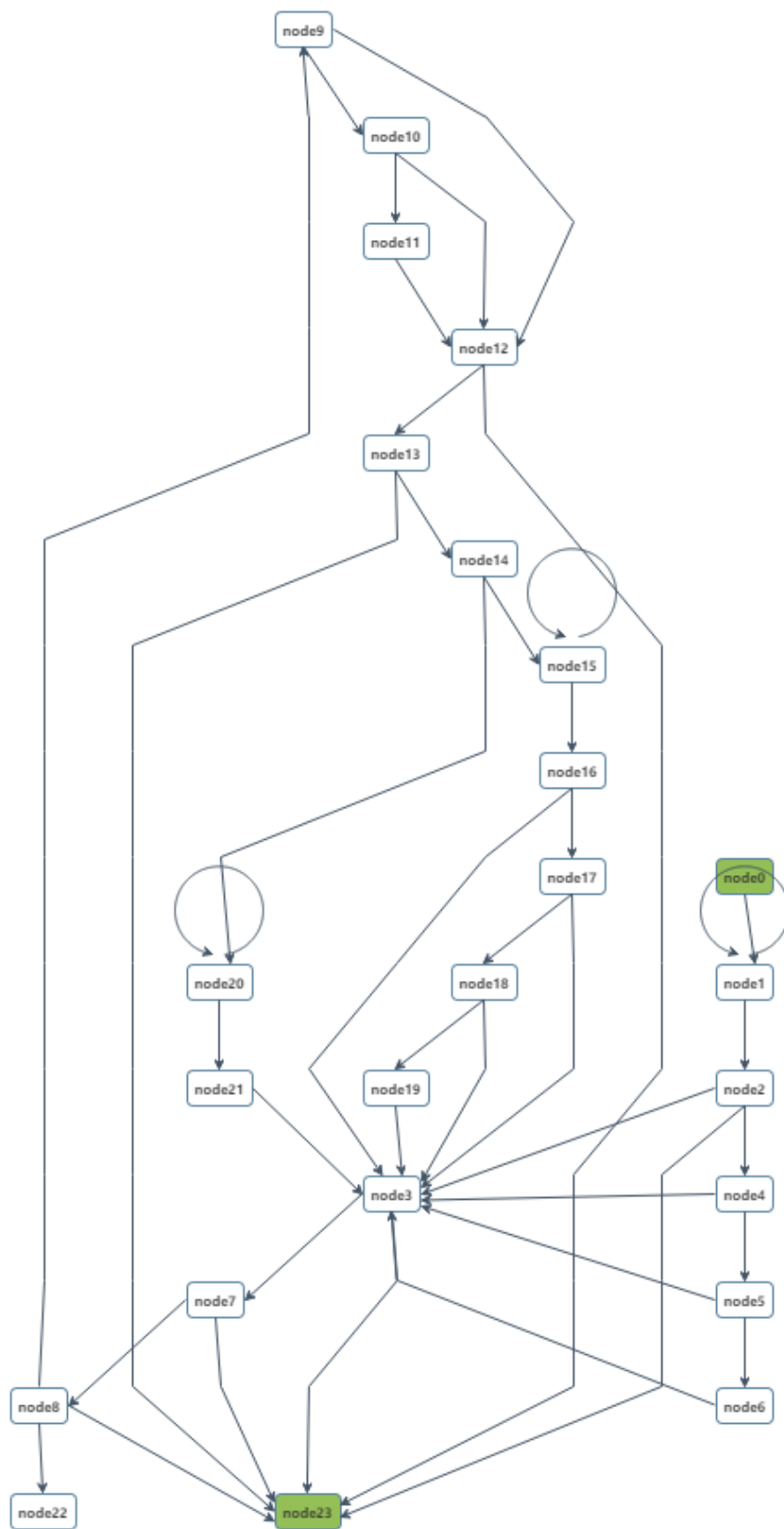
Graph 3



Graph 4

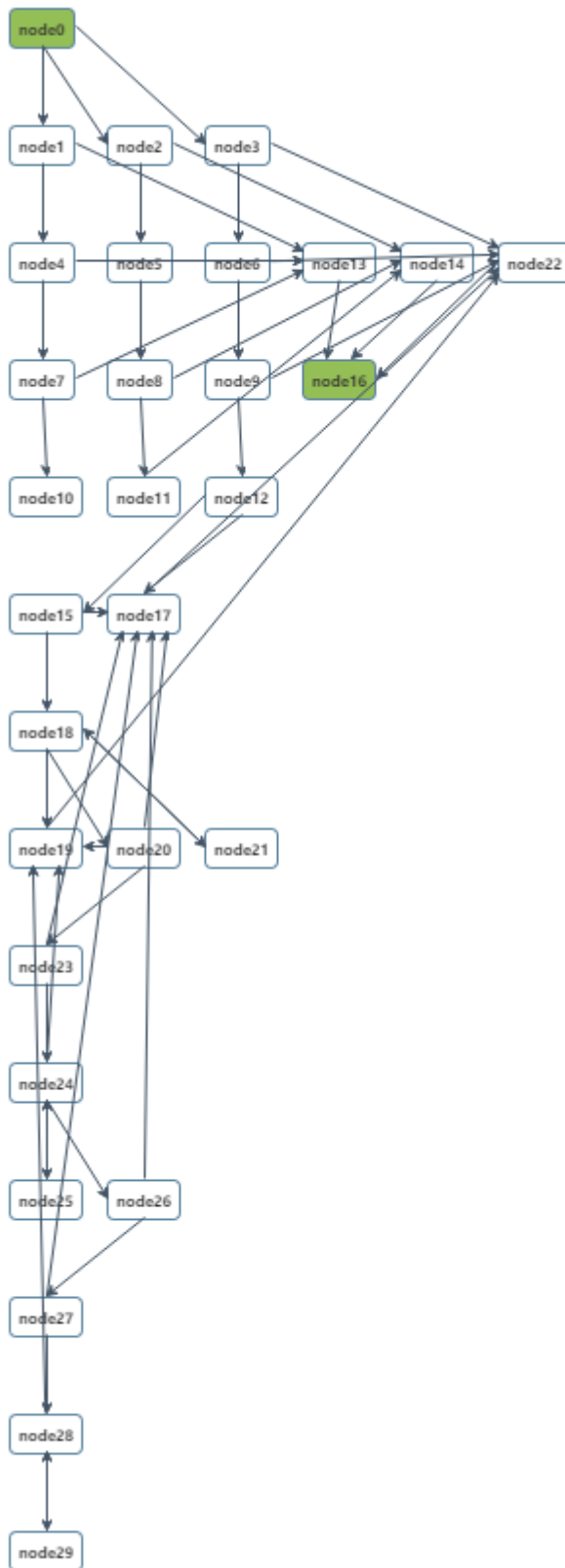


Graph 5

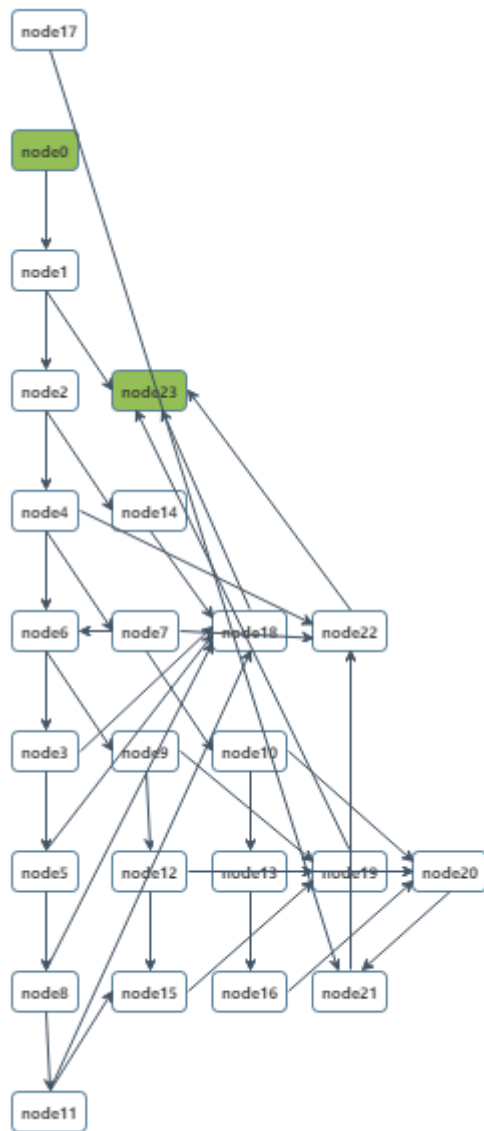




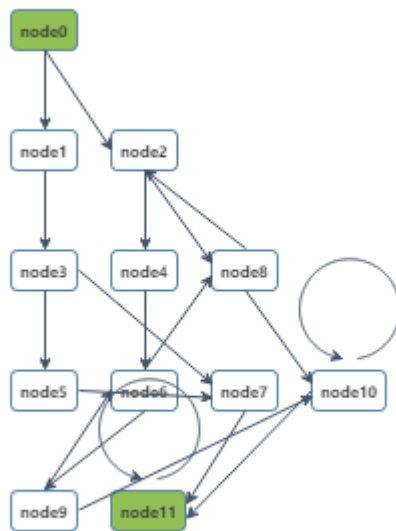
Graph 1



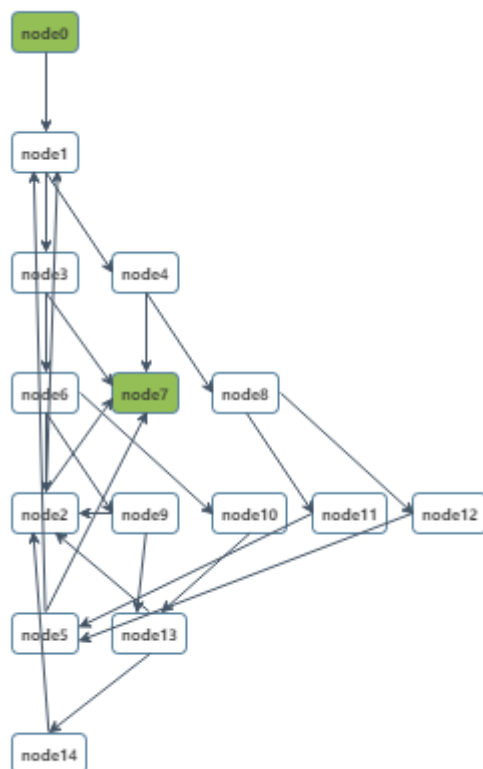
Graph 2



Graph 3



Graph 4



Graph 5

