Subject

Unser Ziel ist es, die *Layer Assignment* Phase des *Layered* Layout Algorithmus zu visualisieren. Hierbei gibt die Graphische Schnittstelle dem Benutzer die Möglichkeit, den Algorithmus Schritt für Schritt betrachtet. So soll der Benutzer durch Ausprobieren ein intuitives Verständnis für den Algorithmus entwickeln. Wichtig ist uns dabei, die verständliche Visualisierung des Algorithmus, so wie eine intuitiv zu bedienende GUI. Weniger wichtig ist uns eine gute Performance bei sehr großen Graphen, da für das Verständnis der *Layer Assignment* Phase kleinere Graphen besser geeignet sind.

Technische Umsetzung

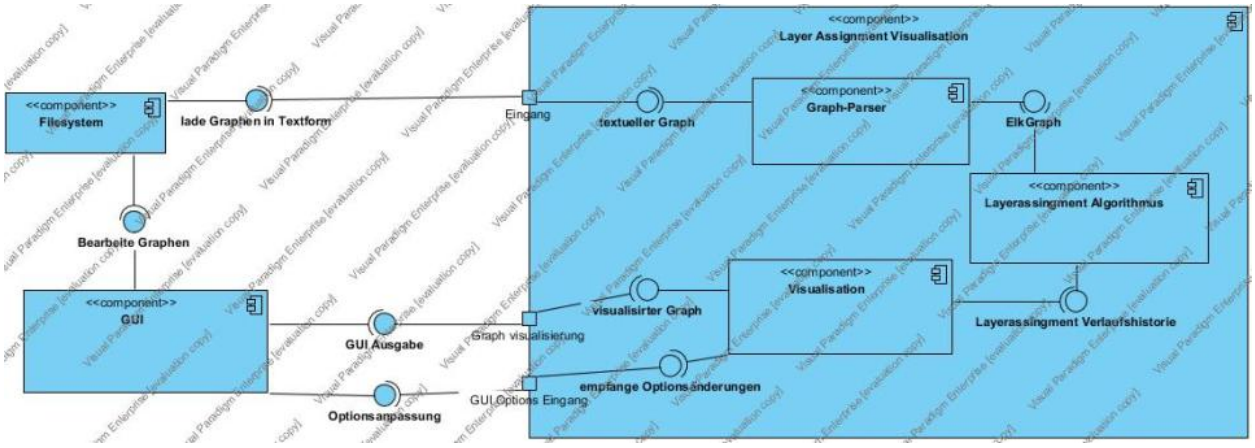
Die technische Umsetzung ist in drei Schritte unterteilt:

1. In diesem ersten Schritt wird ein vom Benutzer erstellter Graph eingelesen und auf Kreisfreiheit überorüft, falls Fehler auftreten wird eine Error-Text-Box angezeigt, falls nicht wird ein layerbarer ElkGraph zurückgegeben.

2. Als nächstes wenden wir den *Layer Assignment* Algorithmus auf den Elkgraphen an. Hierbei erstellen wir nach jedem Schritt einen SimpleGraph (vereinfachte Graphenstruktur, die nur die notwendigen Informeationen für die Visualisierung enthällt) aus dem Elkgraphen. Nachdem der *Layer Assignment* Algorithmus fertig ist, geben wir eine Liste dieser SimpleGraphs an die Visualisierung weiter.

3. In der Visualisierung wird in jedem Schritt ein neuer SimpleGraph aus der Ausgabe des Algorithmus aus Schritt 2 geladen. Die Unterschiede, die zwischen zwei SimpleGraphs bestehen, werden dabei hervorgehoben. So ist es dem Benutzer möglich die Auswirkungen des Algorithmus in jedem Schritt nachzuvollziehen.

Wie in folgendem Komponentendiagramm dargestellt, existieren Schnittstellen zwischen den einzelnen Berechnungsschritten, die die Datenweitergabe erlauben und an denen Überprüfungen vorgenommen werden, die sicher stellen, dass die weitere Berechnung möglich ist.



Parsing

Die Parsing-Klasse ist dafür zuständig das eigens erstelle Textformate einzulesen und in einen ElkGraph umzuwandeln, zusätlich wird der eingelesene Graph auf Kreisfreiheit überprüft da der Layer-Assignment-Algorithmus nur für kreisfreie Graphen definiert ist.

Syntax

Eine Node hinzufügen:
    node _<node name>

Eine Edge hinzufügen:
    edge_<start node> _<end node>
Wobei _ für beliebig viele Whitespaces steht, weiter muss jeder Knoten der von einer edge benutzt wird vorher mit dem *node* Keyword eingeführt werden

Methoden

Parser.parse(<filepath>)
Return: null falls ein fehler auftrat, sonst ein ElkGraph

Benutzung des Parsers

```
try {
ElkNode testGraph = Parser.parse("testGraphs/testfile.txt");
}
catch (Exception e)
{
e.printStackTrace();
}
```

Beispiel

```
node n1
node n2
node n3


edge n1 n3
edge n2 n3
```

Layer Assignment Algorithm

Die Klasse LayerAssignment enthällt eine Reihe hilfreicher Funktionen, die Wichtigste ist allerdings die Methode assignLayers. assignLayers nimmt einen ElkGraphen, führt das Layerassignment an ihm durch und gibt eine Bearbeitungshistorie in Form einer ArrayList vom typen SimpleGraph zurück. Wobei SimpleGraph eine von uns entwickelte vereinfachte Graphenstruktur ist. Jeder SimpleGraph in der Liste stellt also einen Momentaufnahme aus dem Layerassingment des Graphen dar. Durch Seiteneffekte beinflusst die Methode auch den eingegebenen ElkGraphen, so dass er am Ende der Methode als vollständig gelayerter Graph vorliegt. Dazu haben wir jedem Knoten 3 Propertys gegeben: LAYER: Ein Integer wert größer oder gleich 1, der das Layer angibt, in dem sich der Knoten befindet
IS_DUMMY: Ein boolscher wert, der True ist, wenn es sich bei dem Knoten um einen von uns eingefügten Dummyknoten handelt. Zu beachten ist, dass auch Edges die Property IS_DUMMY haben.
POSITION_IN_LAYER: Gibt an, an welcher Stelle im Layer sich der Knoten befindet, diese Property muss in der Crossingminimisation optimiert werden.
Beim einfügen der Dummyknoten geht der Algorithmus wie folgt vor: Wenn eine Kante k ein Layer übersprngt, fügen wir für jedes Layer zwischen den Layern des Startknotens von k und des Endknotens von k einen neuen Dummyknoten in dem jeweiligen Layer ein. Nun fügen wir Dummykanten so ein, dass von jedem Dummyknoten eine Dummykante zum Dummyknoten im nächsthöheren Layer führt. Desweiteren soll eine Dummykante vom höchsten Dummyknoten zum Zielknotne von k führen. Zuletzt wird noch der Zielknoten von k auf den Dummyknoten im Layer direkt über dem Startknoten von k gesetzt.
SimpleGraph ist eine vereinfachte Graphenstruktur. Sie enthällt nur eine Liste von Knoten und eine Liste von Kanten. Wobei diese auch die 3 Propertys beinhalten.

Visualisierung

Die GUI wurde mittels *Java Swing* implementiert und nutzt ein *JSplitPane*, um die textuelle Darstellung eines Graphen gleichzeitig mit der Visualisierung des auf diesen Graphen angewandten Algorithmus anzuzeigen. Der Graph ist hierbei in textueller Form editierbar, so dass der Benutzer komfortabel Änderungen im Graphen vornehmen kann.
Da wir im Algorithmus keinen *longest path* berechnen, um die Anzahl der nötigen Layer zu bestimmen, können wir in jedem Schritt alle Quellknoten einem Layer zuordnen. Quellknoten sind hierbei Knoten ohne eingehende Kanten, deren Startknoten noch keinem Layer zugewiesen wurden). Die tatsächliche Anzahl der benötigten Layer wird über die Länge der SimpleGraph Liste berechnet, in der noch keine Dummyknoten eingesetzt wurden. Diese Liste erhält die Visualisierung bevor sie gestartet wird.
In welchem Schritt sich der Algorithmus gerade befindet, wird oberhalb des visualisierten Graphen angezeigt. Weiterhin wird jeder Knoten dessen Position sich ändert rötlich eingefärbt. Jede Kante, die durch einen Dummyknoten und zwei neue Kanten ersetzt wird, wird kurz bevor sie ausgeblendet wird rötlich hervorgehoben, um jeden Schritt der Berechnung nachvollziehbar zu machen. Um DummyKnoten erkennbar zu machen, werden sie rund und blasser als normale Knoten dargestellt.

Öffnen eines Graphen

Momentan gibt es einen Unterordner *testGraphs* im Projekt, der einige Testgraphen enthält. Der Inhalt dieses Ordners wird dem Benutzer im File Dialog angezeigt, wenn er auf *Load File* in der Applikation klickt. Dies war zu Testzwecken sinnvoll und wurde bisher nicht angepasst.

Speichern eines Graphen

Momentan gibt es einen Unterordner *savedImages* im Projekt. Der Inhalt dieses Ordners wird dem Benutzer im File Dialog angezeigt, wenn er über den Rechtsklick in der Applikation und dann auf *Save Image* klickt. Um nun die aktuelle Ansicht als Bild zu speichert, muss ein Name angegeben werden. **Achtung! Keinen Dateityp angeben!** Das Bildt wird als *.png* im entsprechenden Ordner gespeichert.

User Interface

Slider und Button: Der *Step Size* Slider startet aus Ästhetikgründen für die Beschriftungen bei 0. Wird hier die 0 ausgewählt, wird kein Update an der Visualisierung vorgenommen. Die Obergrenze ist die Anzahl der möglichen Berechnungsschritte. Befindet sich die Berechnung bereits in einem fortgeschrittenen Berechnungsschritt, wird beim Modifizieren des Sliders überprüft ob die Maximale Anzahl der Berechnungsschritte nicht überschritten wird. Dieser Slider bietet also eine zusätzliche Möglichkeit zum *Speed* Slider die Animation des Algorithmus zu beschleunigen. Um die Anzahl der Schritte genau angeben und ablesen zu können wurde außerdem ein Textfeld hinzugefügt, welches mit dem Slider Synchronisiert ist. Der Slider kann nicht verwendet werden während die Animation des Algorithmus, die durch den *Play* Button gestartet wurde, läuft.

Der im *Speed* Slider angegebene Wert definiert, wie viele Schritte die Knoten zwischen ihrer Start und Zielposition zurücklegen sollen. Ist der Wert des Sliders hoch, so erreichen die Knoten ihre Zielposition in weniger Schritten und umgekehrt. Die Geschwindigkeit ist über die Anzahl der Berechnungszyklen definiert, da wir erreichen wollen, dass die Knoten alle zum (fast) gleichen Zeitpunkt ihre Zielposition erreichen, unabhängig von der Strecke, die sie dafür zurücklegen müssen. Obere und Untere Grenzen wurden so gewählt, dass die Animation flüssig läuft und nachvollziehbar bleibt.

Button und Slider verfügen über Tooltips, um die Bedienung zu vereinfachen.

Toolbar: Die Elemente für die Justierung und Steuerung der Visualisierung sind in einer Toolbar zusammengefasst. Das hat den Vorteil, dass man diese per *drag and drop* aus dem Fester heraus- und auch wieder hineinziehen kann.

Animation

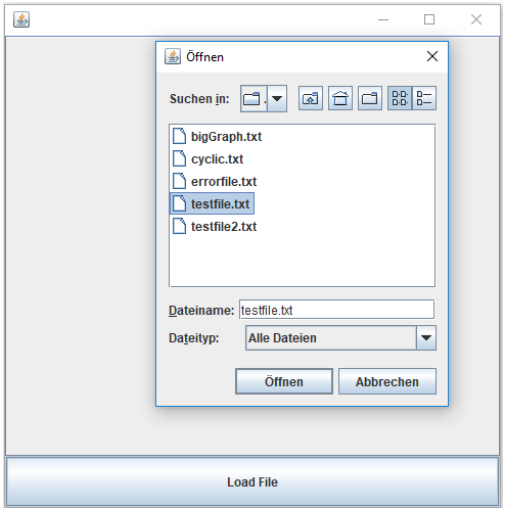
Die Animation wird über Updates und einen Timer ausgeführt. Bevor der Timer gestartet wird erhält jeder Knoten eine neue Zielposition. Wird der Timer gestartet, bewegt sich jeder Knoten auf seine Zielposition zu bis er diese erreicht hat. Hierbei wird eine Updatefunktion

für die Positionierung der Knoten entsprechend häufig durch den Timerevents ausgelöst. In jedem Timerevent wird die View neu gezeichnet. Das heißt die Kanten, deren Positionen von ihren Start- und Zielknoten abhängen, werden ebenfalls mit den Knoten Bewegt, was zu einer flüssig ablaufenden Animation führt.
Wurde der *Play* Button gedrückt, so wird ein Thread gestartet, der für jeden Schritt die Knotenpositionen erneuert und den Timer für die Updatefunktion startet. Dieser Thread wartet darauf, dass der Timer nicht mehr aktiv ist bevor er den Schritt der Berechnung ausführt.
Geht man jedoch Schrittweise durch den Algorithmus, indem mal den *Forward* oder *Backward* Button drückt, so kann man den nächsten Schritt bereits ausführen, während die Animation noch läuft. An dieser Stelle erschien es uns aus rechentechnischen Gründen nicht sinnvoll per buisy waiting o.Ä. das Ende des Animation abzuwarten, bevor die Button wieder ansprechbar sind.

User Guide

Neues File Öffnen

Beim Starten der Applikation wird ein leeres Fenster mit einem *Load File* Button angezeigt. Betätigt man diesen Button, öffnet sich ein Dialogfenster, in dem der Benutzer ein *.txt* File auswählen und öffnen kann.



Der *Load File* Button ist auch noch dann sichtbar, wenn bereits Graphen geladen wurden. Öffnet man weitere Text Files, werden diese entsprechend in neuen Tabs geladen.
Ist das File kein Graph im Sinne unseres Eingabeformats oder ist der Graph nicht azyklisch, so wird eine Fehlermeldung ausgeben.

User Interface

Hat der Benutzer einen gültigen Eingabegraphen ausgewählt und geöffnet, so stehen im mehrere Möglichkeiten zur Auswahl:

Anpassen der Fenstergrößen: Alle Fenster können in ihrer Größe angepasst werden. Der Balken, der textuelle und graphische Ansicht der Graphen trennt, ist hierbei nur verschiebbar, wenn die Mindestgrößen der beiden Seiten nicht unterschritten werden. Das Panel, das die Steuerungselemente enthält kann per *drag and drop* aus dem Fenster heraus- oder wieder hineingezogen werden.

Editieren des Graphen: Auf der linken Seite des Fensters wird der Inhalt der geöffneten Textdatei angezeigt. Der Text kann modifiziert werden und die Änderungen durch das Klicken des *Safe*/*Reload* Buttons gespeichert werden. Hierbei wird außerdem die Anzeige des Graphen neu geladen, so dass die vorgenommenen Änderungen sichtbar werden.

Achtung! Erzeugt man hierbei einen ungültigen oder zyklischen Graphen, kann dies nur geändert werden, indem man außerhalb der Applikation die Datei öffnet und korrigiert!

Graphen neu laden: Den Knoten im Graphen werden zunächst Zufallskoordinaten zugewiesen, Dies kann dazu führen, dass sich Knoten beispielsweise überschneiden oder an einem Punkt häufen. Ist dergleichen der Fall, bietet es sich an den *Safe*/*Reload* Button zu betätigen. Hierbei werden den Knoten neue Zufallskoordinaten zugewiesen.

Weitere Graphen öffnen: Um weitere Graphen in neuen Tabs zu öffnen, kann der Benutzer den *Load File* Button betätigen.

Justierungen vornehmen: Die GUI stellt drei Slider bereit, die es dem Benutzer erlauben die Darstellung des Graphen und die Animation des Algorithmus anzupassen. Der *Step Size* Slider legt fest, wie viele Berechnungsschritte auf ein mal ausgeführt werden sollen. Über Textfeld neben dem *Step Size* Slider kann die Selbe Angabe gemacht werden.

Der *Speed* Slider definiert wie schnell sich die Knoten in der Animation bewegen.

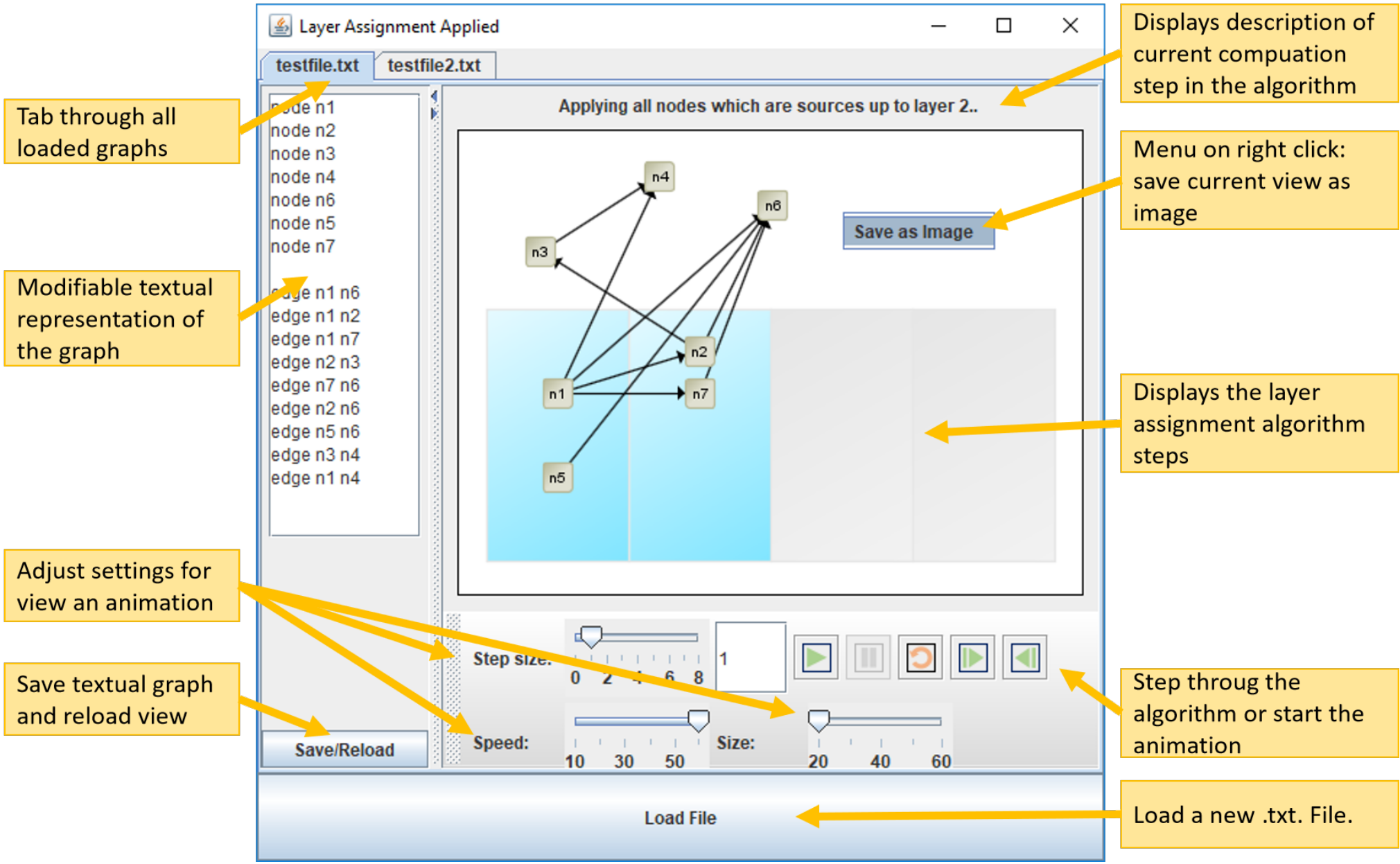
Der *Size* Slider setzt die Größe der Knoten fest und damit auch die Größe des Labels im Knoten.

Anzeige Speichern: Per Rechtsklick auf die rechte Seite des Fensters (die den Graphen anzeigt) kann der Benutzer einen Snapshot der aktuellen Anzeige speichern.

Animation Starten/Stoppen: Durch Drücken des *play* Buttons kann der Benutzer die Animation starten, die entsprechend seiner Justierungen den Algorithmus auf den Graphen Ausführt. Durch Drücken auf den *Pause* Button wird die Animation im aktuellen Schritt unterbrochen.

Zurücksetzen: Wird der *Reset* Button gedrückt, so begeben sich die Knoten wieder zu ihren Ursprungspositionen und die Animation kann erneut gestartet oder der Algorithmus schrittweise verfolgt werden.

Schrittweises Ausführen: Die Button *Jump Forward* und *Jump Backward* springen jeweils um so viele Berechnungsschritte vor oder zurück, wie über den *Step Size* Slider angegeben wurde.


Layer Assignment Applied
testfile.txt
testfile2.txt
node n1
node n2
node n3
node n4
node n6
node n5
node n7
edge n1 n6
edge n1 n2
edge n1 n7
edge n2 n3
edge n7 n6
edge n2 n6
edge n5 n6
edge n3 n4
edge n1 n4
Applying all nodes which are sources up to layer 2..
Save as Image
n1
n2
n3
n4
n5
n6
n7
Step size:
0 2 4 6 8
1
Speed:
10 30 50
Size:
20 40 60
Save/Reload
Load File
Tab through all loaded graphs
Modifiable textual representation of the graph
Adjust settings for view an animation
Save textual graph and reload view
Displays description of current compuation step in the algorithm
Menu on right click: save current view as image
Displays the layer assignment algorithm steps
Step throug the algorithm or start the animation
Load a new .txt. File.