Umsetzung eines stresswertbasierten Knotenlayout-Algorithmus für den Eclipse Layout Kernel

Bachelorarbeit

Jean-Pierre Runge

Bachelor Informatik
30. September 2024

Echtzeit und Eingebettete Systeme Gruppe
Institut für Informatik
Christian-Albrechts-Universität zu Kiel

Betreut durch
Prof. Dr. Reinhard von Hanxleden
Sören Domrös, M.Sc.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.
Weiterhin erkläre ich, dass die digitale Fassung dieser Arbeit, die dem Prüfungsamt per E-Mail zugegangen ist, der vorliegenden schriftlichen Fassung entspricht.

Kiel,

Abstract

Der Eclipse Layout Kernel (ELK) ist ein Werkzeug zum automatischen Erstellen von zweidimensionalen Graphen. Viele verschiedene Algorithmen zum Layouting der Komponenten stehen dabei zur Verfügung. Eine speziellere Art von Graph ist das Knotendiagramm zum Veranschaulichen mathematischer Knoten, die im Teilgebiet der Knotentheorie, aber auch in anderen wissenschaftlichen Bereichen Verwendung finden.

In dieser Arbeit wird das Konzept der Stressreduktion genutzt, um einen neuen Algorithmus in ELK zu implementieren, der einem vom Nutzer vorgegebenen Graphen in ein Knotendiagramm umwandelt. Die bekanntesten Knoten in der Mathematik haben bestimmte wiedererkennbare Formen, die es nachzustellen gilt. Hierfür hat sich die Funktion eines bereits existierendes Stresslayouts als praktische Initialisierung herausgestellt, bevor durch das Reduzieren eigens berechneter Stresswerte die bestimmten Kriterien an ein Knotendiagramm erfüllt werden.

Die Evaluation mit anderen Visualisierungen von Knotendiagrammen zeigt, dass der Algorithmus vergleichbare Knoten mittels Stressreduktion erfolgreich erstellen kann. Für ein möglichst übereinstimmendes Aussehen kann der Nutzer mehrere der implementierten Layoutoptionen für den Knoten individuell justieren.

Das Ergebnis ist ein funktionierender und anpassbarer neuer Layout-Algorithmus, der natürlich aussehende Knotendiagramme von kleineren Knoten erstellen kann und somit die Vielseitigkeit der Algorithmen-Bibliothek ELK weiter erhöht. Die einstellbaren Parameter erlauben es potenziell weitere Knoten in ästhetischer Form visualisieren zu lassen.

Vorwort zur genutzten Terminologie

Vorab müssen einige Überschneidungen von deutschen Begriffen geklärt werden, die es im Englischen nicht geben würde. Komponenten eines Graphen werden im Englischen meistens *nodes* und *edges* bezeichnet. Die läufige Übersetzung ins Deutsche wäre Knoten und Kanten. Doch gerade der Begriff Knoten, als Verbindungskomponente von Kanten, überschneidet sich mit dem herkömmlichen Knoten, als Verwicklung oder Verschlingung einer Schnur. Ein mathematischer Knoten, im Englischen *Knot*, bezieht sich auf gerade diese Metapher einer verknoteten Schnur, die an ihren Enden verbunden ist.

Daher werden im Folgenden nun die Verbindungskomponenten im Graphen als Knotenpunkt und der Knoten, den der gesamte Graph ergibt, einfach als Knoten bezeichnet.

Acknowledgements

Ich möchte mich hiermit bei meinem Professor Dr. Reinhard von Hanxleden bedanken, dass ich in seiner Arbeitsgruppe kurzfristig noch einen Platz bekommen habe, um meine Bachelorarbeit zu verfassen. Zusätzlich bedanke ich mich bei meinem Betreuer Sören Domrös, welcher mich mit voller Motivation in ELK eingearbeitet und auf die richtigen Ideen gebracht hat diesen Layout-Algorithmus zu entwickeln.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1. Mathematischer Knoten

Der mathematische Knoten ist ein eigenes Forschungsgebiet der Mathematik, bei dem Eigenschaften eines Knotens analytisch untersucht und mithilfe von Polynomen beschrieben werden.

Damals noch eher aus Neugier nachgegangen, wurden im späten neunzehnten Jahrhundert Theorien aufgestellt, in der die Erforschung des Knotens auch neue Erkenntnisse in anderen Forschungsbereichen bringen könnten [Liv93; AFL+22]. Dies führte dazu, dass mehr Mathematiker sich mit der Knotentheorie beschäftigten und versuchten verschiedene Knoten zu klassifizieren, bis es zum eigenen Teilgebiet der Topologie wurde. Mittlerweile hat die Knotentheorie Anwendungszwecke in der Chemie, bei der potenziellen Entwicklung neuer synthetischer Materialien, kann aber auch in der Biologie auf die Struktur von Proteinen, sowie auf DNA-Strängen angewandt werden [LJ15]. Die Analyse der entstandenen Knoten in einer verschlungenen menschlichen DNS haben zu Erkenntnissen in der Chemotherapie geführt.

Knotendiagramme haben dabei geholfen, die Listen an Knoten anhand ihrer Überkreuzungen zu klassifizieren und sind weiterhin ein wichtiges Werkzeug, um sich von der Struktur der Knoten eine Vorstellung machen zu können. Auch wenn es dreidimensionale Diagramme von Knoten gibt, handelt es sich bei Knotendiagrammen meistens um einen zweidimensional angeordneten Graphen in Form eines an den Enden zusammengeklebten Fadens, der vorher mehrmals ineinander verwickelt und verschlungen wurde. Bereits 1877 wurden von Tait erstmals verschiedene Knoten in einer Liste zusammengetragen [Tai77].

Dabei ist zu beachten, dass andere relevante Eigenschaften zum Unterscheiden von Knoten in solchen Darstellungen nicht berücksichtigt werden. In der Abbildung 1.1[1] sind die ersten 15 primären Knoten, nach Reihenfolge ihrer Überkreuzungen, abgebildet. Diese Aufzählung basiert auf eine von Roflsen zusammengestellten Liste von Knoten [Rol03].

[1] https://katlas.org/wiki

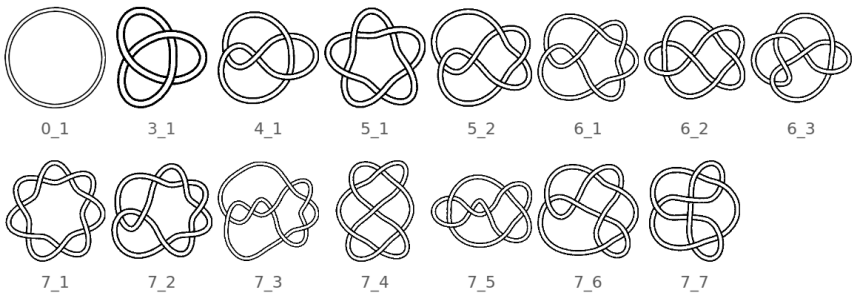


Abbildung 1.1. Eine Liste von Knotendiagrammen der ersten 15 primären Knoten.

1.2. Automatische Anordnung eines Graphens

Von Diagrammen und Graphen gibt es viele verschiedene Darstellungsarten, bei denen die Eigenschaften der Komponenten teilweise stark verändert werden müssen. Dafür gibt es automatische Verfahren, die eine Beschreibung oder Definition des Graphens bekommen und anschließend die Komponenten so anpasst, dass der gewünschte Graphentyp angezeigt wird.

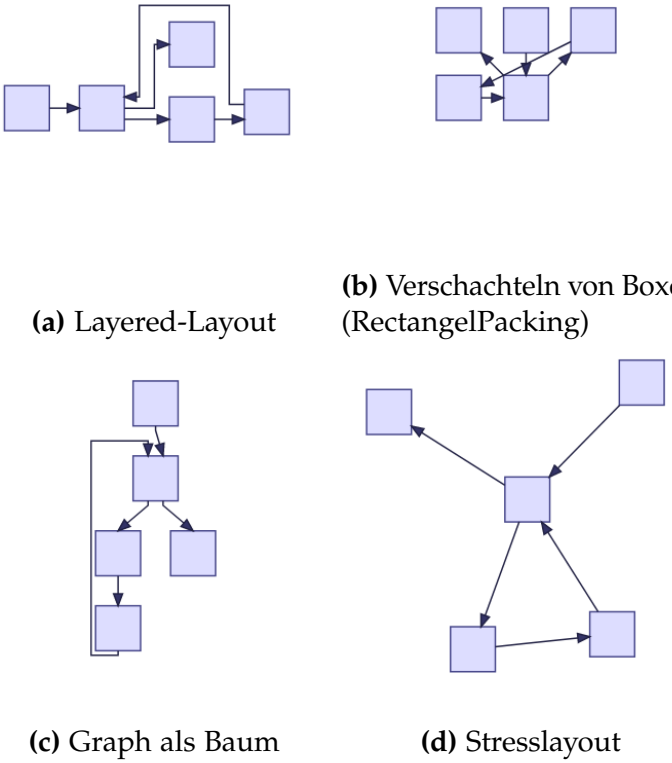
Der Eclipse Layout Kernel[2] des KIELER-Projektes[3] bietet Algorithmen mit denen die unterschiedlichen Layouts berechnet und anschließend ein planarer Graph gerendert werden können [DHS+23]. Zuvor muss in einer entsprechenden ELK-Textdatei die Knotenpunkte und verbundenen Kanten eines Graphens definiert werden, ehe die Einstellung des Algorithmus über die Form bestimmt.

Die Abbildung 1.2 zeigt mehrere Beispiele von Resultaten der in ELK auswählbaren Layout-Algorithmen. Ein Layout für zweidimensionale Knotendiagramme gibt es bisher noch nicht. Das Implementieren eines entsprechenden Algorithmus für die Konstruktion von Knotendiagrammen würde die Vielseitigkeit von ELK weiter erhöhen und war somit ein motivierender Faktor dieser Arbeit. Hierfür braucht es zuerst einen Ansatz die mathematischen Knoten als Graph definieren zu können, ehe eine Technik überlegt werden muss, um das organische Aussehen durch große Bögen und teilweise asymmetrisch verteilten Überkreuzungen nachzuahmen.

1.3. Entwicklung eines Knotenlayouts

Um mathematischen Knoten ähneln zu können, werden gewisse Kriterien an den Graphen nötig sein. Andere Arbeiten haben bereits Algorithmen gezeigt, mit denen

[2]https://github.com/eclipse/elk
[3]https://github.com/kieler



(a) Layered-Layout

(b) Verschachteln von Boxen (RectangelPacking)

(c) Graph als Baum

(d) Stresslayout

Abbildung 1.2. Definierte Knotenpunkte und Kanten werden mithilfe von ELKs Layout-Algorithmen unterschiedlich angeordnet.

selbst komplexe Knoten kompakt visualisiert werden konnten.

Ein anderer möglicher Ansatz basiert auf sogenannten Stresslayouts, welche eine gleichmäßige Anordnung von Knotenpunkten garantieren, indem ein aus Abständen berechneter Stresswert reduziert wird [GKN04]. Es ist vergleichbar mit einer Art Kraft zwischen den Knotenpunkten, wodurch diese sich anziehen oder abstoßen, bis ein natürliches Gleichgewicht den gesamten Graphen formt. Mit den richtigen Kriterien für die Stresserzeugung, hat dieses Prinzip das Potenzial die organische und teilweise willkürlich wirkende Optik von Knoten nachzustellen, um so ein authentisches Knotendiagramm zu generieren. Diese Kriterien sind jedoch komplexer und könnten zu Komplikationen mit dem von ELK gesetzten Framework führen.

In dieser Arbeit wird mithilfe von Stressreduktion ein neuer Algorithmus zur Erstellung annehmbarer Knotendiagramme im ELK implementiert. Dafür müssen Funktionen zur Berechnung akkurater Stresswerte und zum adäquaten Anpassen

des Graphens entwickelt werden. Das neue Feature soll letztlich dem Nutzer zur Verfügung stehen, wenn dieser es wünscht seinen definierten Eingabegraphen als Knotendiagramm visualisieren zu lassen. Auch werden zusätzliche Layoutoptionen bereitgestellt, welche die individuelle Anpassbarkeit eines Knotens erhöhen und bei Bedarf Justierungen für die Stressreduktion zulassen, sollte das Ergebnis unbrauchbar sein.

Die vom resultierenden Algorithmus erzeugten Knotendiagramme müssen mit den bekannten Varianten aus der Mathematik verglichen werden, um zu demonstrieren, ob sich ein Stresslayout auch wirklich eignet. Sollten die Darstellungen, selbst mit angepassten Optionen, zu abwegig von ihren Originalen sein oder bestimmte Knoten der Limitation der implementierten Stressreduktion unterliegen, kann in zukünftigen Arbeiten der Algorithmus weiter ausgebaut werden.

1.4. Aufbau der Arbeit

In Kapitel 2 wird die Umgebung von KIELER und ELK, in welche der Algorithmus implementiert wird, näher ausgeführt. Hier wird auch beschrieben wie ein mathematischer Knoten in einen Graphen umgewandelt werden kann und wie das Prinzip des Stresslayouts funktioniert.

Das Kapitel 3 zeigt einen anderen Ansatz zum Erzeugen von Knotendiagrammen.

Das gesamte Konzept wie der Algorithmus abläuft und die Form eines Knotens umgesetzt werden soll wird in Kapitel 4 aufgeführt.

Kapitel 5 beschreibt die Implementierung des Konzepts und die Probleme, die dabei entstanden sind. Dort werden auch die Lösungsansätze präsentiert, sowie die für den Nutzer anpassbaren Layoutoptionen aufgelistet.

In Kapitel 6 wird die Anwendung mit gängigen Knoten evaluiert und erzeugte Knotendiagramme mit anderen Darstellungen verglichen.

Zum Schluss wird in Kapitel 7 der Entwicklungsprozess und die Ergebnisse nochmal zusammengefasst, sowie weitere potenzielle Arbeiten für die Zukunft vorgeschlagen, die noch existierende Probleme aufgreifen.

Kapitel 2

Grundlagen und Technologien

In diesem Kapitel wird das Framework, in dem der neue Algorithmus implementiert werden soll, und die benutzen Mittel erklärt, die in den weiteren Kapiteln gebraucht werden.

2.1. Kiel Integrated Environment for Layout Eclipse Rich Client (KIELER)

Der Kiel Integrated Environment for Layout Eclipse Rich Client (KIELER)[1] ist ein Open-Source-Framework von mehreren Projekten und Werkzeugen und zielt darauf ab, graphische Modelle von komplexen Softwaressystemen zu verbessern. Auch wenn Eclipse im Namen steckt, sind diese Technologien unabhängig davon als Java und JavaScript Bibliotheken, sowie als Erweiterungen für VS Code oder anderen Integrated Development Environments (IDEs) verfügbar.

Die unterschiedlichen Konzepte beziehen sich auf verschiedene Anwendungsfällen von topologischen Problemen, wie die Verständlichkeit von automatisch erstellten Diagrammen, oder der effektiveren Analyse von dynamischen Systemen in der Softwareentwicklung.

2.1.1. Eclipse Layout Kernel (ELK)

Einer dieser Bibliotheken von KIELER ist der Eclipse Layout Kernel (ELK)[2], eine Sammlung von Algorithmen zur Berechnung der in Diagrammen platzierten Komponenten. Der Nutzer hat eine Bandbreite verschiedener Möglichkeiten eine sogenannte ELK-Textdatei (ELKT) in einen gewünschten Graphentypen übersetzen zu lassen [DHS+23]. Dabei werden Hierarchien innerhalb von Knotenpunkten, sowie Ports als explizite Ankerpunkte für Kanten unterstützt.

Jegliche Änderungen an der ELKT werden das Diagramm automatisch neu generieren lassen, wobei ELK lediglich für die Berechnung der Attribute aller Komponenten

[1] https://github.com/kieler
[2] https://github.com/eclipse/elk

des Graphens zuständig ist, wie beispielsweise die Positionen der Knotenpunkte und die Routen der Kanten. Andere Teile des KIELER Frameworks übernehmen anschließend die tatsächliche Visualisierung.

Die Modularität von ELK ermöglicht es neue Layout-Algorithmen zur bestehenden Bibliothek hinzuzufügen und soll somit die zentrale Umgebung für diese Arbeit sein. Es werden die Dateien `KnotLayoutProvider.java` und `Knot.melk` erzeugt die für die Einbindung in das gesamte Projekt verantwortlich sind. In `KnotLayoutProvider.java` findet die Implementierung des Algorithmus statt, wobei diese für eine bessere Übersicht weiter aufteilt werden kann, solange die `layout` Methode den gegebenen Graphen annimmt, verändert und anmerkt, sobald der Algorithmus abgeschlossen wurde. In `Knot.melk` können hingegen Layoutoptionen definiert werden, mit denen sich Werte aus der ELKT auslesen und in die Parameter des Algorithmus zuweisen lassen.

Für diese Arbeit müssen die Eigenschaften eines Knotendiagramms auf die von ELK vorgegebenen Klassen und Objekte angepasst werden. Die in den folgenden Kapiteln präsentierten Konzepte und Ansätze können allerdings auch unabhängig von ELK umgesetzt werden.

2.1.2. KIELER Lightweight Diagams (KLighD)

Wie bereits erwähnt ist ELK nur für die Berechnung der Komponenten zuständig. Das KIELER Lightweight Diagams (KLighD)[3] ist ein Projekt für leichtgewichtige Visualisierungen von Modellen, ganz ohne Erfordernis von Bearbeitungstools wie es in anderen Graphikeditoren der Fall ist. Die erzeugte Darstellung wird hierfür aus der Modellbasis aufgebaut, die zuvor von ELK überreicht wurde. Ab dem Moment, wo Komponenten nicht mehr gebraucht werden, entfernt KIELER Lightweight Diagams (KLighD) diese aus dem erstellten View-Modell wieder.

In dieser Arbeit wurde speziell die Erweiterung für die Eclipse IDE genutzt, bei der das finale Rendering in eine Eclipse-View durch das 2D-Grafikframework Piccolo geschieht.

2.2. ELK-Textdatei (ELKT) und kgt-Datei (kgt)

Die Textquelle ELKT beinhaltet die Informationen, welche Knotenpunkte und Kanten existieren, wie sie verbunden sind und welcher Layout-Algorithmus verwendet werden soll. Hier muss der Nutzer vorher seinen Eingabegraphen definieren, bevor dieser berechnet und gerendert werden kann. Über die Layoutoptionen lassen sich

[3] https://github.com/kieler/KLighD

Eigenschaften wie der genutzte Algorithmus oder das Aussehen der Kanten einstellen. Weitere Optionen werden dann je nach gewähltem Layout-Algorithmus unterstützt.

Des Weiteren gibt es auch noch die kgt-Datei (kgt), welche ebenfalls für Definieren eines Graphens verwendet werden kann. Ähnlich wie in der ELKT, müssen dafür die Komponenten wie Knotenpunkte und Kanten aufgelistet werden. Allerdings ist die Syntax etwas anders. So werden Kanten direkt in dem Scope eines Knotenpunktes untergebracht.

Der Vorteil von kgt wird in späteren Kapiteln deutlich, denn bestimmte Teile eines Knotendiagramms lassen sich nicht über ELK verändern. Es braucht direkte Einstellungen am Rendering und eigene Designs von Kanten, welche nur im kgt-Format unterstützt werden.

2.2.1. Mathematische Knoten als Graph definieren

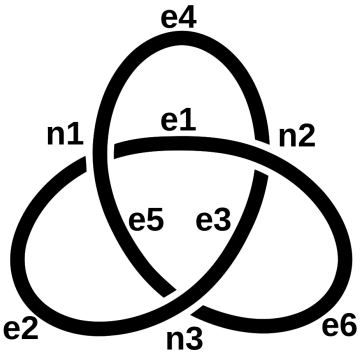
Ein wichtiges Merkmal von mathematischen Knoten sind die Anzahl an Überkreuzungen, also wie häufig der an den Enden zusammengeklebte Faden sich selbst kreuzt. Diese Überkreuzungen werden von den Knotenpunkten dargestellt und soweit es geht verkleinert. Um ein richtiges Kreuz abzubilden braucht jeder Knotenpunkt dann noch genau vier Kanten.

Mit der Hilfe von Kontrollpunkten wird in späteren Kapiteln der Verlauf der Kanten so geändert, dass diese orthogonal in den Knotenpunkt eintreffen, was einen knickfreien Übergang gewährleisten soll. Die Kontrollpunkte muss der Nutzer selbst aber nicht definieren, sondern werden nur von dem Algorithmus hinzugefügt, manipuliert und letztlich für das Rendering verwendet.

Als Beispiel kann der Trefoil, ein nichttrivialer Knoten mit der geringsten Anzahl Überkreuzungen, in Abbildung 2.1a betrachtet werden. Die drei Überkreuzungen werden als Knotenpunkte $n1, n2$ und $n3$ definiert. Insgesamt wird der durchgehende Faden auf sechs Kanten aufgeteilt – drei äußere und drei innere. Das ergibt die Kanten $e1$ bis $e6$, wo jede Kante einen Knotenpunkt mit den jeweils zwei anderen verbindet. Verbindungen werden mit einem Pfeil $(->)$ beschrieben.

Abbildung 2.1b zeigt dann den Trefoil als ELKT Variante. Für die kgt-Version in Abbildung 2.1c müssen die Kanten direkt im Scope ihrer Startknotenpunkten definiert werden. Ein Pfeil zeigt dabei wieder auf den Zielknotenpunkt.

Wichtig ist es hier, dass schon eine Voraussetzung beachten werden muss. Die zwei gegenüberliegenden Kanten an einem Knotenpunkt bilden zusammen entweder die unterführende oder überführende Kante einer Überkreuzung. Für den knickfreien Übergang muss dieses Paar an Kanten bei jeder Repositionierung von Knotenpunkten immer gegenüber bleiben. Damit der Algorithmus effektiv zwischen diesen Paaren unterscheiden kann, werden die eingehenden und ausgehenden Kanten eines Kno-



(a) Knotendiagramm eines Trefoil

```
// Trefoil
algorithm:Knot

// Nodes
node n1
node n2
node n3

//Edges
edge e1:n1 -> n2
edge e2:n1 -> n3

edge e3:n2 -> n1
edge e4:n2 -> n3

edge e5:n3 -> n1
edge e6:n3 -> n2
```

(b) ELKT-Variante des Trefoils

```
// Trefoil
properties:
algorithm=Knot

// Nodes + Edges
knode n1 {
  kedge (->n2)
  kedge (->n3)
}

knode n2 {
  kedge (->n1)
  kedge (->n3)
}

knode n3{
  kedge (->n1)
  kedge (->n2)
}
```

(c) kgt-Variante des Trefoils

Abbildung 2.1. Repräsentation eines Knotendiagramms als ein Graph in ELK

tenpunktes betrachtet. Die Voraussetzung ist also, dass jeder Knotenpunkt genau zwei ausgehende und zwei eingehende Kanten besitzen muss. Dies ist für jeden Knoten auch umsetzbar, da jede Überkreuzung eine Kante hat, welche über die andere verläuft. Es kann sich so vorgestellt werden, dass die zwei ausgehenden Kanten die unterführende Kante bildet und die zwei eingehenden Kanten dann insgesamt zu der überführenden Kante zusammengesetzt werden.

Ein Problem liefert das System bei etwas größeren Knoten mit mehr Überkreuzungen. Das richtige Angeben der Start- und Zielknotenpunkten bei Kanten wird weniger offensichtlich, wenn komplexere Knotendiagramme umgesetzt werden sollen. Besonders da es auch passieren kann, dass die Deklaration von Komponente in den Zeilen vertauscht werden müssen, ehe das gewünschte Knotendiagramm sich bilden kann.

Hierfür ist ein Übersetzer nützlich, der ein Polynom aus der Knotentheorie oder einen Zahlenstrang bekommt und entsprechend die benötigten Komponenten definiert. Das ist aber nicht der Fokus dieser Arbeit. Für Knoten die in Kapitel 6 zum Vergleichen verwendet werden, können die jeweiligen Modelle, sowohl als ELKT als auch kgt-Datei, im Anhang A gefunden werden.

2.3. Finden eines geeigneten Layouts für Knoten

So wie ein Knoten als Graph umgesetzt wird, ist vor allem die Position der Knotenpunkte essenziell, um den Graphen eine Form annehmen zu lassen, die sich mit bereits existierenden Knotendiagrammen vergleichen lässt.

Ein interessanter Layout-Algorithmus hierfür ist das von Eades entworfene kraftorientierten Layout. Es verfolgt den grundlegenden Ansatz ein ansprechenden und übersichtlichen Graphen zu erzeugen, indem eine gleiche Länge aller Kanten forciert wird. Nach einer zufälligen initialen Positionierung, verwendet er hier die Metapher von Sprungfedern, welche eine bestimmte Kraft zwischen den verbundenen aber dennoch beweglichen Knotenpunkten vermittelt und sie sich so lange zueinander hinziehen oder wegdrücken, bis ein Kräfteausgleich stattgefunden hat [Ead84]. Kamada und Kawai haben diese Art von kraftorientierten Layouts weiter angepasst, um auch gewünschte Distanzen außerhalb der Nachbarschaft von Knotenpunkten zu berücksichtigen [KK89].

2.3.1. Verwendung eines stresswertbasierten Layouts

Die Ideen für die optimale Positionierung der Knotenpunkte in einem kraftorientierten Layout werden bei der Stressreduktion in dem von Ganser et al. entworfenem Stresslayout übernommen, jedoch auf andere Weise als ein Anziehen und Abstoßen. Dabei wird eine Stressfunktion erstellt, welche die aktuelle Position aller Knotenpunkte in Betracht zu einer gewünschten Distanz zwischen diesen zieht. Sollten die eigentlichen Distanzen weiter oder kürzer als die Entfernung über Kanten und Nachbarknotenpunkte sein, dann erzeugt dies einen Stresswert. Der Algorithmus versucht den Stresswert immer weiter zu verringern, indem die Koordinaten der Knotenpunkte angepasst und ein neuer Stresswert berechnet wird, um somit ein lokales Minimum auf der Stressfunktion zu erreichen [GKN04].

Ein Vorteil dieser Methode ist das Zusammenstellen und Gewichten des Stresswertes. Für den Ansatz von Ganser et al. ist die optimale Entfernung der Knotenpunkte zueinander das Hauptkriterium, aus dem der Stresswert hervorgeht. Zusätzliche Kriterien an den Graphen können hinzugefügt werden, indem hierfür ebenfalls ein Stresswert berechnet und zu dem gesamten Stress addiert wird. In Stressreduktionsverfahren wird dann versucht alle Kriterien weitestgehend zu erfüllen, denn dadurch würde am wenigsten Stress existieren. Und auch der Prozess selbst kann optimiert werden, um dieses lokale Minimum auf einer Stressfunktion möglichst schnell zu erreichen.

Das Erfüllen mehrerer Kriterien und Anpassen des Graphens, bis ein stabiler Zustand erreicht wurde, macht die Technik eines stresswertbasierten Layouts zu

einem guten Kandidaten, um natürlich aussehende Knotendiagramme zu erzeugen.

ELK selbst verfügt bereits über das Stresslayout. Bei der Anwendung auf den in Abschnitt 2.2.1 konzipierten Trefoil, wie in Abbildung 2.2 zu sehen, wird ersichtlich, dass die stresserzeugenden Kriterien erweitert werden müssen. Allerdings ist die Position der Knotenpunkte für den Fall des Trefoils bereits gut. Alle drei Knotenpunkte sind haben einen gleichmäßigen Abstand zueinander und sind als Dreieck angeordnet, so wie auch die Überkreuzungen des Trefoils in Abbildung 2.1a. Das existierende Stresslayout ist somit eine praktische Initialisierung für Knoten dieser Art, und auch andere Knoten können sich daraus formen lassen.

Auch braucht es weitere Funktionen, damit die der fertig angeordnete Graph in ein vollständiges Knotendiagramm verwandelt werden kann, mitsamt ausschweifenden äußeren Bögen und korrekten Lücken für Überkreuzungen. Diese Aspekte werden in Kapitel 4 weiter vertieft und letztlich in Kapitel 5 als Algorithmus im ELK umgesetzt.

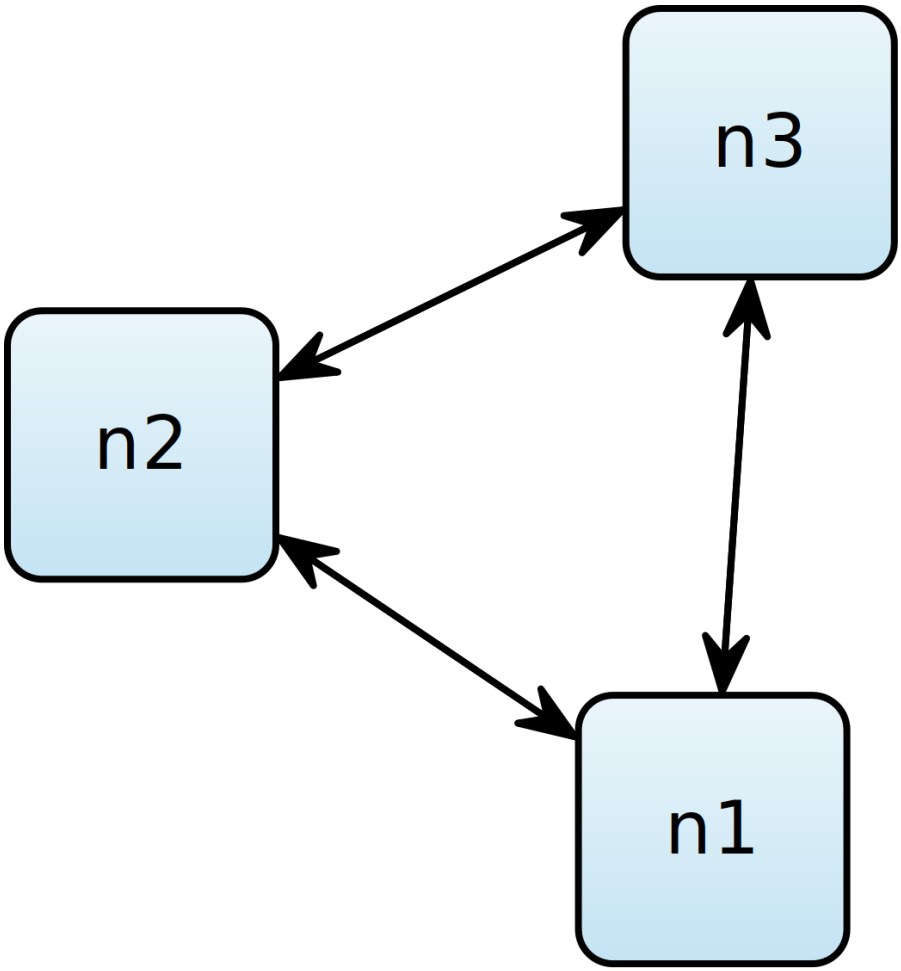


Abbildung 2.2. Der als Graph definierte Trefoil-Knoten mit dem existierenden Stresslayout aus ELK.

Kapitel 3

Verwandte Arbeiten

Digital erzeugte Diagramme von mathematischen Knoten gibt es in verschiedenen Ausführungen. In diesem Kapitel wird ein anderer Ansatz, vorgestellt der bereits erfolgreich auch große Knoten erzeugen konnte.

3.1. Knotendiagramm durch Tutte-Einbettung

Der von Browne entwickelte Algorithmus stützt sich auf eine sehr unterschiedliche Technik, im Vergleich zu der Stressreduktion [Bro23]. En Gausskode beschreibt die Überkreuzungen eines Knotens und in welcher Reihenfolge diese besucht werden. Von diesem Gausskode aus werden dann zunächst die nötigen Knotenpunkte, Kanten und auch Facetten definiert, bevor daraus ein Gittergraph des Knotens entsteht. Mithilfe einer Tutte-Einbettung kann dieser Gittergraph in einen vollwertiges und anschauliches Knotendiagramm umgewandelt werden.

Die Ergebnisse von Browne zeigen, dass dieser Algorithmus in der Lage ist selbst große Knoten mit vielen Überkreuzungen zu visualisieren [Bro23].

Kapitel 4

Automatische Erzeugung eines authentischen Knotendiagramms

In diesem Kapitel wird das Hauptkonzept zur automatischen Erzeugung eines Knotendiagramms präsentiert. Die bekanntesten Knoten in der Mathematik haben bestimmte wiedererkennbare Formen, die es nachzustellen gilt. Dabei werden die genauen Kriterien ermittelt, welche die Optik eines Knotens, mithilfe des Verfahrens zur Stressreduktion, forcieren sollen. Durch rotierende Bewegungen und Verschiebungen von Knotenpunkten und Kontrollpunkten, werden Kanten möglichst direkt von ihren Startknotenpunkten zu den Zielknotenpunkten geführt, um ungewollte Schnittpunkte zu vermeiden. Letztlich werden weitere Funktionen verwendet, um den Graphen vergleichbar mit Knotendiagrammen zu machen.

4.1. Kriterien an den Graphen

Im optimalen Fall sollen die resultierenden Knotendiagramme ihren entsprechenden Knoten eindeutig abbilden und dabei eine ästhetisch ansprechendes Aussehen besitzen. So wie der Knoten als Graph umgesetzt ist, sollen die Knotenpunkte die Stellen sein für sich der Faden selbst überkreuzt. Es darf also keine weiteren Schnittpunkte von Kanten geben. Das kann erreicht werden, wenn eine Kante möglichst ohne Umwege von ihrem Startknotenpunkt zum Zielknotenpunkt führt.

Bei mathematische Knoten ist die Richtung der Kanten irrelevant, denn sämtliche Kanten in dem Knotendiagramm sollen nahtlos verbunden werden um den Anschein einer einzelnen durchgezogenen Schlaufe, die an ihren Enden verbunden ist, zu erwecken. Für den Algorithmus ist die Unterscheidung zwischen eingehenden und ausgehenden Kanten aber wichtig, weshalb die Richtung von Kanten dennoch beim Erstellen des Eingabegraphen, wie in Abschnitt 2.2.1, berücksichtigt werden muss.

Um den nahtlosen und knickfreien Übergang zwischen den Kanten zu gewährleisten, müssen extra Kontrollpunkte um einen Knotenpunkt platziert werden. Dabei sollen immer jeweils zwei Kontrollpunkte direkt gegenüber voneinander liegen und die Enden der Kanten orthogonal zum Knotenpunkt führen. Bei jeder Justierung des

Graphens müssen diese Kontrollpunkte weiter berücksichtigt werden muss.

Sobald die Position von Knotenpunkten weiter angepasst werden soll, braucht es ein Kriterium für den Abstand zu anderen Knotenpunkten. Sie sollen nicht ineinander geraten, damit auch ihre Kanten sich nicht ungewollt berühren, aber sich auch nicht zu sehr distanzieren, da sonst der Graph zu sehr verzerrt wird.

Knotendiagramme haben häufig große, ausschweifende Bögen an den äußeren Seiten. Wenn größere Bögen zwischen den Knotenpunkten entstehen sollen, braucht es ebenfalls zusätzliche Kontrollpunkte, die entsprechend platziert werden müssen. Aus den Knotenpunkten und Kontrollpunkten werden die Bézierkurven berechnet, um die Bögen in abgerundeter Form darzustellen.

Zu der Authentizität eines Knotendiagramms gehören auch die teilweise unsymmetrischen Abstände und Bogengrößen dazu, die dem Knoten eine organische oder natürliche entstandene Optik verleiht. Wenn also nicht alle Komponenten gleichmäßig verteilt oder parallel zueinander sind, ist das je nach Knoten nicht weiter schlimm, sondern trägt, bis zu einem gewissen Grad, mit zu den wiedererkennbaren Formen bei.

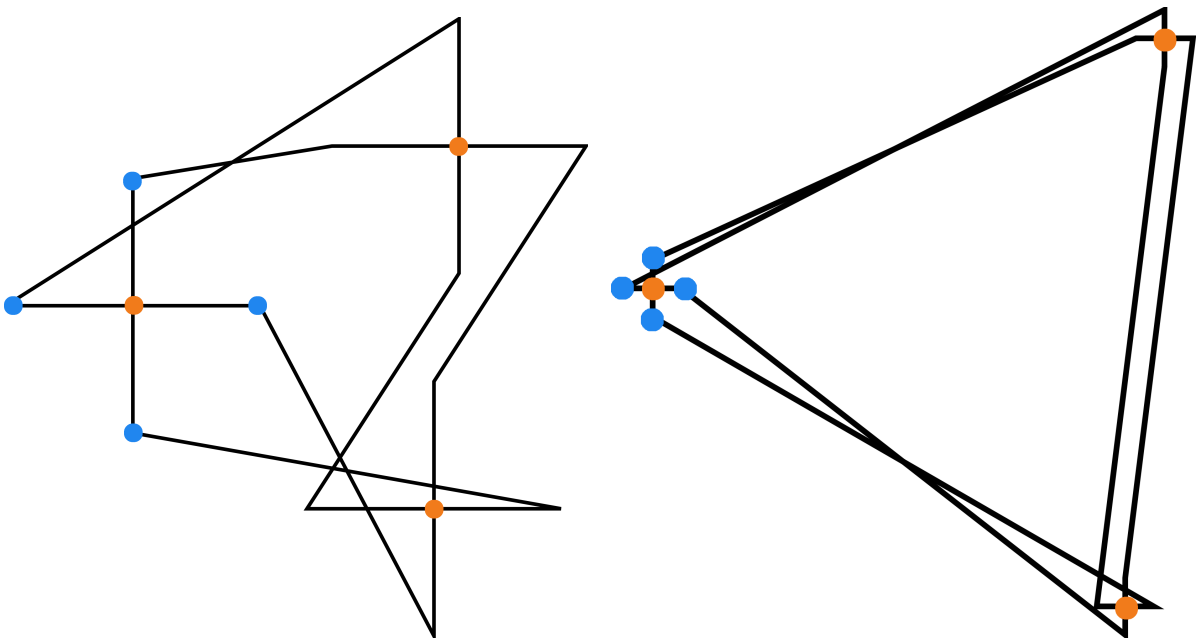
Für eine eindeutige Unterscheidung, welche Teil des Fadens über den anderen verläuft, sollen die Überkreuzungen im gerenderten Knotendiagramm ebenfalls mit Lücken gekennzeichnet sein.

4.2. Initiale Platzierung

Nachdem der Nutzer einen Graphen als ELKT auf Grundlage eines gewünschten Knotens definiert hat, wird dieser ELK und dem Algorithmus übergeben. Über den eingebenden Graphen können auf sämtliche Knotenpunkte, von Knotenpunkten auf alle verbundenen Kanten und über die Kanten auf dessen Kontrollpunkte, zugegriffen werden.

Um die Komponenten erst einmal in der Darstellung unterzubringen wird das bereits existierende Stresslayout verwendet, so wie es in Abschnitt 2.3.1 erklärt wurde. Diese Anordnung hat sich als gute Basis herausgestellt, besonders bei kreisförmigen Knoten wie dem Trefoil oder Pentafoil, da die Position der Knotenpunkte und deren Abstände bereits nahezu identisch mit anderen Knotendiagrammen ist. Hier braucht es dann keine weitere Anpassung bezüglich der Koordinaten der Knotenpunkte. Bei anderen Knoten, wie dem Achterknoten, kann eine stressreduzierende Verschiebung es in eine Form bringen, die dem Achterknoten aus bekannten Darstellungen nahekommt.

Sobald die initiale Ausrichtung durch das Stresslayout steht, werden neue Komponenten in den Graphen eingebunden. Vier Kontrollpunkte pro Knotenpunkt sind



(a) 35 Pixel Abstand zwischen Knotenpunkt und Kontrollpunkt

(b) 5 Pixel Abstand zwischen Knotenpunkt und Kontrollpunkt

Abbildung 4.1. Initialisierung eines Trefoils durch Stresslayout mit extra Kontrollpunkten und ohne weiterer Anpassung. Knotenpunkte wurden in Orange und Kontrollpunkte in Blau hervorgehoben.

nötig, um die vier Kanten orthogonal in den Knotenpunkt zu führen. Auch dienen sie dazu, aus den Kanten runde Bézierkurven machen zu können, wodurch die Bögen entstehen, die bei Knotendiagrammen üblich sind. Die Distanz zwischen dem Knotenpunkt und seinen zugehörigen Kontrollpunkten ist ausschlaggebend, wie gut erkennbar die Überkreuzungen werden und welches Ausmaß die Bögen annehmen.

Von dieser ersten initialen Platzierung aus, werden Stresswerte anhand der Kriterien aus Abschnitt 4.1 dafür sorgen, dass die Komponenten wie Knotenpunkte und Kontrollpunkte einen akzeptablen Zustand im Graph erreichen. Im optimalen Falle wäre dies ein Knotendiagramm, welches den vom Nutzer gegebenen Knoten in einer Form wiedergibt, die ähnlich zu seinen Gegenstücken aus der Literatur ist. Es ist jedoch nicht ausgeschlossen, dass das Prinzip der Stressreduktion für einige Knoten nicht ausreicht und es möglicherweise bessere initiale Layouts als das Stresslayout von ELK gibt.

Die Abbildung 4.1 zeigt das Knotendiagramm für einen Trefoil unmittelbar nach der Platzierung durch das bestehende Stresslayout. Lediglich die zusätzlichen Kontrollpunkte wurden den Kanten hinzugefügt und deren Koordinaten relativ zu den

Knotenpunkten angepasst. In dem Graphen wurde noch keine Technik zum Konstruieren der Bézierkurven angewandt. Dies ermöglicht es, die aktuelle Positionen der Knotenpunkte, hier in Orange markiert, und die der Kontrollpunkte, in Blau, leichter abzulesen. Die Kontrollpunkte befinden sich sowohl über und unter dem jeweiligen Knotenpunkt, sowie an den Seiten.

In Abbildung 4.1a wurde ein großer Abstand zwischen diesen Punkten verwendet. Die Kurven würden einen größeren Bogen machen, ehe sie in den Knotenpunkt führen, wodurch die Überkreuzungen besser zu erkennen sind. Beim Verschieben der Knotenpunkte muss darauf geachtet werden, dass dieser gebildete Radius sich nicht mit den von anderen Knotenpunkten überschneidet, da sonst die Wahrscheinlichkeit hoch ist, dass sich die Kanten ebenfalls schneiden.

Hingegen wurde in Abbildung 4.1b ein sehr viel kleinerer Abstand gewählt und die entstehenden Bögen sind viel flacher. Die gesamte Visualisierung wird dadurch kompakter gemacht, da die Distanz zwischen den Knotenpunkten ebenfalls kleiner sein darf. Dieser größere Freiraum wird die Stressberechnungen des Algorithmus ändern und ein teilweise stark unterschiedliches Ergebnis verursachen. Je nachdem, welchen Knoten der Nutzer erstellen möchte und wie er sich diesen vorstellt, wäre ein individuelles Anpassen des Abstandes also wünschenswert.

4.3. Ausrichtung der Kanten

Nachdem alle Knotenpunkte initial positioniert wurden, haben die neuen hinzugefügten Kontrollpunkte eine vorerst feste Position. Eine Kante verläuft nun von einem Startknotenpunkt aus zum ersten Kontrollpunkt, dann zum zweiten Kontrollpunkt und von dort aus in seinen Zielknotenpunkt. In Abbildung 4.2 ist der Pfad, den eine Kante von $n1$ nach $n2$ nimmt, mit lila markiert. Hier sind die Kontrollpunkte orange und die Kontrollpunkte blau.

Die Startpositionen der Kontrollpunkte sorgen dafür, dass einige verbundene Kanten von $n1$ spitz in die Richtung des nächsten Kontrollpunktes verlaufen, wie es in Rot markiert ist. Dieser spitze Winkel deutet an, ob die Kante zurück in die Richtung verläuft, aus der sie von ihrem Startknotenpunkt herkam, oder schon passend in Richtung ihres Ziels ausgerichtet ist. Bei einem Winkel von 0° würde die Kante also direkt zurück durch den Knotenpunkt führen, während sie bei einem Winkel von 180° gerade aus weiter verlaufen würde. Ein angestrebtes Ziel ist es diesen Winkel möglichst an 180° zu halten und einen Winkel von nahe 0° zu verhindern.

Um den Winkel zu ändern, muss der jeweilige Kontrollpunkt angepasst werden, wo der Winkel anliegt. Im Folgenden wird daher auch von Kontrollpunktwinkeln gesprochen. Eine einfache Rotation um den jeweiligen Knotenpunkt reicht aus, um

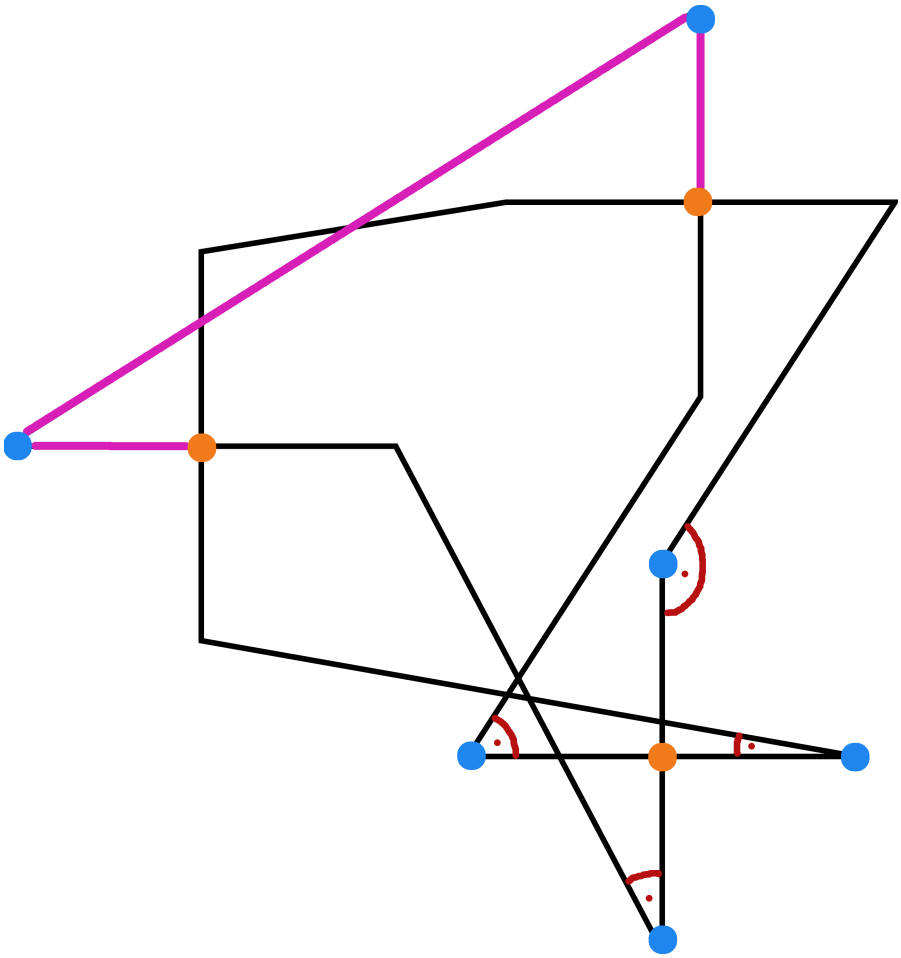
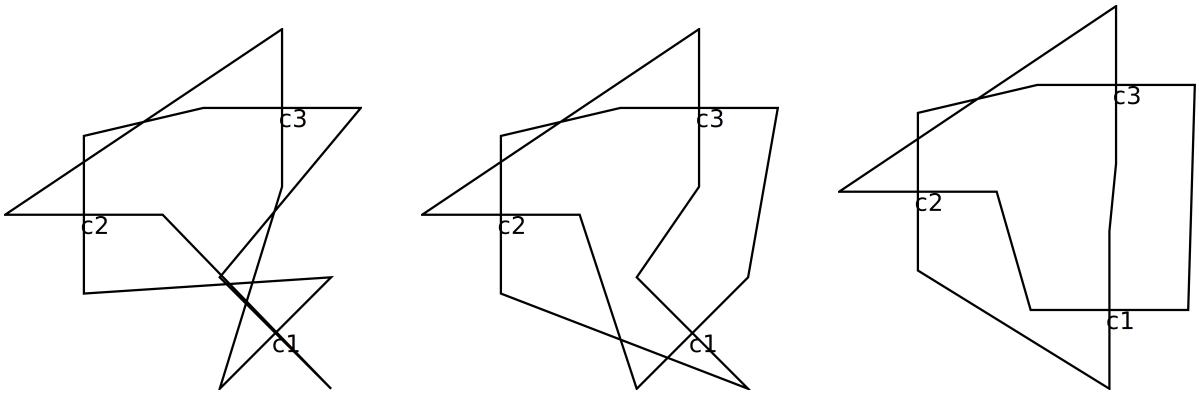


Abbildung 4.2. Trefoil nach Initialisierung. Die in Rot markierten Winkel an den blauen Kontrollpunkten bestimmen den weiteren Verlauf einer Kante.

den Kontrollpunkt in eine Position zu bringen, von der aus die Kante direkt zu ihrem Ziel verlaufen kann. Dabei ist wichtig, die Form der Überkreuzung, die der Knotenpunkt mit den vier Kanten darstellen soll, beizubehalten. Die anderen Kontrollpunkte müssen also ebenfalls um den gleichen Wert rotiert werden, was den Anschein erweckt, dass sich der Knotenpunkt selbst dreht, obwohl dieser unverändert bleibt. Es gilt alle vier Kontrollpunktwinkel in Betracht zu ziehen, um einen Stresswert zu erzeugen. Dieser kann auch Winkelstress genannt werden. Auf dessen Basis erfolgt dann eine Rotation, um alle Winkel möglichst groß zu halten. Dabei sollen die kleinen Winkel mehr Stress erzeugen, um zu garantieren, dass 0° nie entstehen können, während Winkel im 90° Bereich komplett akzeptabel und häufig unvermeidlich sind.

Sobald der Stresswert eines Knotenpunktes bestimmt wurde, beginnt die Rotation



(a) Rotation von $c1$ um -45° **(b)** Rotation von $c1$ um $+45^\circ$ **(c)** Rotation von $c1$ um $+90^\circ$

Abbildung 4.3. Rotation eines einzelnen Knotenpunktes eines Trefoils nach Initialisierung ohne weiterer Anpassung.

zur Winkeloptimierung. Sollte eine Rotation in eine Richtung die Kontrollpunktwinkel an den Kanten insgesamt verschlechtern und einen höheren Stresswert verursachen, wird stattdessen mit einer größeren Schrittweite in die entgegengesetzte Richtung rotiert. Dies ermöglicht es den Knotenpunkt über mehrere Iterationen hinweg in die Richtung zu drehen, in der auch eine wirkliche Verbesserung entsteht. Wenn die aktuelle Position bereits optimal ist, wird in jeder weiteren Iteration nur noch vor und zurückgedreht und verbleibt somit in dieser Lage, da ansonsten der Stresswert wieder steigen würde. Der Knotenpunkt befindet sich somit im lokalen Minimum der Stressfunktion.

Auch wenn eine Rotation vorher eine Verbesserung hervorgerufen hat, kann keine Aussage getroffen werden, in welche Richtung die Drehung den besten Effekt erzielt. Sobald das lokale Minimum erreicht ist, würde jegliche Drehung nur schlechtere Werte ergeben. Während der Implementierung hat dies allerdings zu Problemen geführt die in Abschnitt 5.5.1 weiter besprochen werden.

Die Rotation eines Knotenpunktes allein sorgt schon dafür, dass sich die Stresswerte der anderen Knotenpunkte verändern. Dies aber zu berücksichtigen würde dazu führen, dass der rotierende Knotenpunkt sich nicht optimal zu seinen Gunsten ausrichtet. Daher wird der Reihe nach jeder Knotenpunkt gedreht, bis seine Kanten eine gute Ausrichtung gefunden haben, ehe der nächste Knotenpunkt das Gleiche mit dieser neuen Ausgangslage versucht. Diese Sequenz kann beliebig häufig nochmal wiederholt werden, denn nachdem der letzte Knotenpunkt abgeschlossen wurde, könnte der erste möglicherweise erneut optimiert werden.

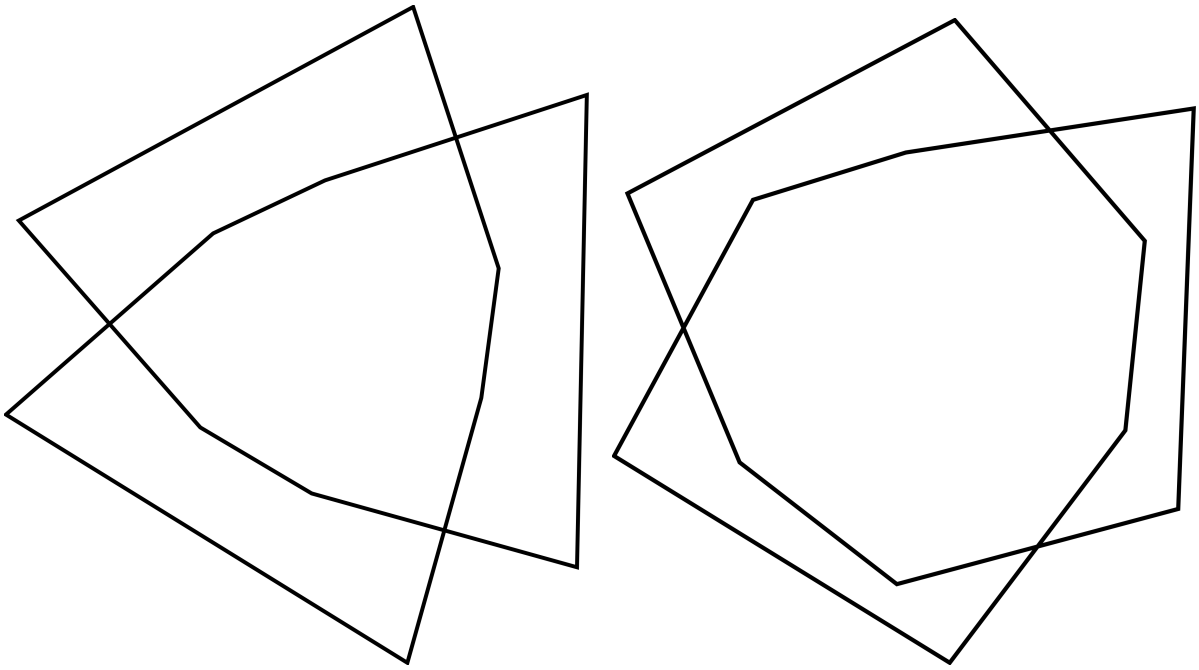
Die Abbildung 4.3 zeigt die Rotation eines einzelnen Knotenpunktes des Trefoils nach seiner Initialisierung, wie es schon in Abbildung 4.1 zu sehen war. Hierbei

werden die vier Kontrollpunkte vorerst um einen vorgegebenen festen Wert rotiert, statt abhängig vom Stresswert, um verschiedene Situationen darzustellen.

In Abbildung 4.3a wurde der Knotenpunkt $c1$ zunächst um -45° gedreht. Dies hat die Folge, dass mehrere sehr spitze Winkel an den Kontrollpunkten entstehen, wobei einer sogar nahezu 0° besitzt. Die dazugehörige Kante geht dementsprechend direkt wieder zurück in die Ursprungsrichtung und durch den Knotenpunkt selbst. Eine solche Ausrichtung wird einen enormen Stresswert erzeugen der definitiv höher als der vorherige ist und dafür sorgen, dass es mit der anderen Rotationsrichtung versucht wird, wie es in Abbildung 4.3b zu sehen ist. Hier wurde um 45° im Uhrzeigersinn gedreht, was einige Winkel entlastet hat und somit einen geringeren Stresswert für $c1$ erzeugt.

In Abbildung 4.3c wurde der Knotenpunkt um insgesamt 90° rotiert und es ist eine deutliche Verbesserung zu den vorherigen Ausrichtungen zu erkennen. Alle Kanten von $c1$ haben relativ direkte Wege zu den anderen Knotenpunkten, wobei keine Kante durch $c1$ selbst verläuft. Auch ein Schnittpunkt mit einer anderen Kante, der vorher noch in Abbildung 4.3b existierte, wurde automatisch behoben. Ein weiterer positiver Nebeneffekt kann in $c3$ erkannt werden. Dort sind die Winkel ebenfalls verbessert worden, obwohl sich dieser Knotenpunkt noch gar nicht ausgerichtet hat. Dies muss allerdings nicht immer der Fall sein und eine eigenständige Rotation von $c2$ und $c3$ würde den Gesamtgraphen weiter in das gewünschte Aussehen bringen. Abbildung 5.6a zeigt das Trefoil, wenn alle Knotenpunkte einmal eine stressreduzierende Rotation ausführen. Dabei wurden die Knotenpunkte der Reihe nach, innerhalb von jeweils 200 Iterationen mit etwa 1° bis 2° pro Iteration, gedreht. Im Ergebnis können die Grundzüge des erwarteten Knotens bereits erkannt werden.

Ein weiterer Ansatz zur Optimierung der Kantenausrichtung wäre eine Aufteilung in zwei Achsen pro Knotenpunkt. Statt alle Kontrollpunkte gleichzeitig zu bewegen und potenziell gute Winkel wieder zu verschlechtern, können die Kontrollpunkte paarweise als eine Achse gedreht werden. Dabei müssen die jeweiligen zwei Kontrollpunkte sich nach wie vor gegenüber vom Knotenpunkt befinden, damit der nahtlose Übergang der Kante weiterhin gewährleistet wird. Dieses Konzept benötigt ein neues Kriterium, nämlich wie nahe beide Achsen zueinander liegen dürfen. Wenn sie direkt aufeinander liegen würden, wäre die Überkreuzung im Knoten nicht mehr erkennbar, während ein orthogonales Verhältnis zwischen diesen ein ideales Kreuz bildet, so wie es bislang der Fall ist. Es wird also der Winkel zwischen den Achsen berechnet und dem Winkelstress durch die Kontrollpunkte hinzuaddiert. Ein Winkel von 45° ist standardmäßig bei den Überkreuzungen und somit erstrebenswert, darf aber auch kleiner sein, solange es nicht zu nahe an 0° herankommt. Dafür muss der Stresswert nicht mehr insgesamt über den Knotenpunkt bestimmt werden, sondern nur über die jeweilige Achse.



(a) Rotation abhängig vom gesamten Stress eines Knotenpunktes

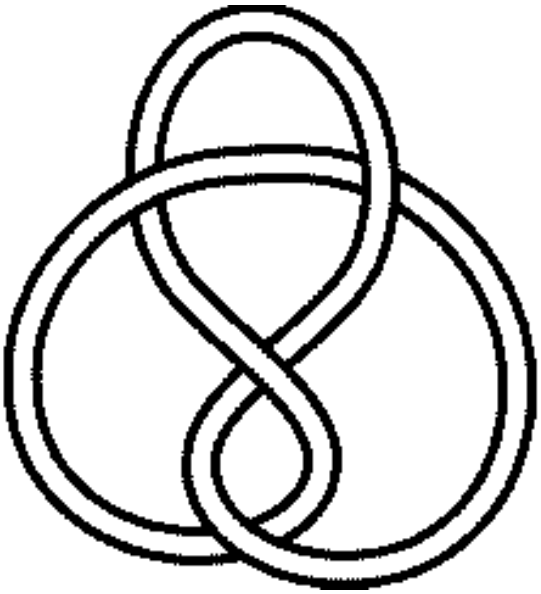
(b) Rotation einzelner Achsen abhängig vom Stress und Winkel zwischen Achsen

Abbildung 4.4. Trefoil nach 200 kleineren Rotationen pro Knotenpunkt, von jeweils 1° bis 2° per Iteration.

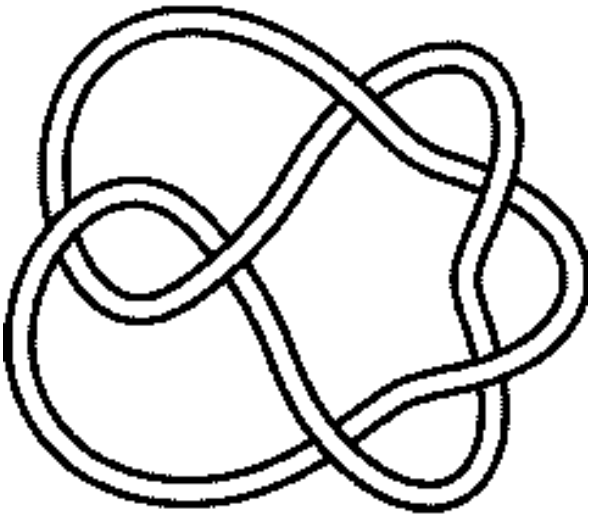
Die Abbildung 5.6b verwendet die Drehung der individuellen Achsen, wodurch die Überkreuzungen leicht gestaucht wirken. Ein Vorteil dieser Technik ist die erhöhte Flexibilität eine geeignete Ausrichtung für die Kanten zu finden. Und auch die Optik ist ähnlich zu Knotendiagrammen, die in verschiedener Literatur zu finden sind, wo ebenfalls nicht alle Überkreuzungen ein perfektes Kreuz bilden. Auf der Kehrseite werden die äußeren Bögen, welche vielleicht eher ausschweifend gewünscht sind, womöglich zu flach dargestellt, da die Kontrollpunkte auch die Größe der Bézierkurve bestimmen und nun näher zusammen liegen.

4.4. Verschiebung der Knotenpunkte

Die Positionierung der Überkreuzungen ist entscheidend für die Form des Knotens. Beim Trefoil oder Pentafoil, sind die Überkreuzungen vom Stresslayout im Kreis angeordnet worden und der Abstand zwischen den verbundenen Nachbarknotenpunkten ist gleichmäßig. Die Initialisierung bestimmt also bereits die optimale Position für Knoten dieser Art.



(a) Gängige Darstellung eines Achterknotens



(b) Gängige Darstellung eines Stevedore-Knotens

Abbildung 4.5. Zwei Knoten aus der von Roflsen zusammengetragenen Liste.

In einem Achterknoten stehen die Überkreuzungen hingegen etwas kompakter zueinander, wie in Abbildung 4.5a gesehen werden kann. Dort sind die unteren beiden Knotenpunkte sich näher, als die oberen zwei, wobei der Unterste weit weg von den oberen ist. Knoten wie der Stevedore-Knoten in Abbildung 4.5b haben sowohl eine kreisförmige Anordnung, als auch eine Sektion mit zwei eng stehenden Überkreuzungen.

Bei einem Trefoil, Pentafoil und anderen artverwandten Knoten sollte der Algorithmus keine Verschiebung vornehmen, weil sie in diesem Falle nicht nötig ist. Für sämtliche andere Knoten müssen Anpassungen vorgenommen werden, wobei in Fällen, wie dem Stevedore-Knoten, es auch lediglich eine Verschiebung von bestimmten Knotenpunkten braucht, um aus der gleichmäßigen Verteilung einen ähnlichen Knoten wie den Stevedore zu machen. Und auch andere Knoten wie der Achterknoten können mit dieser Methode erzeugt werden.

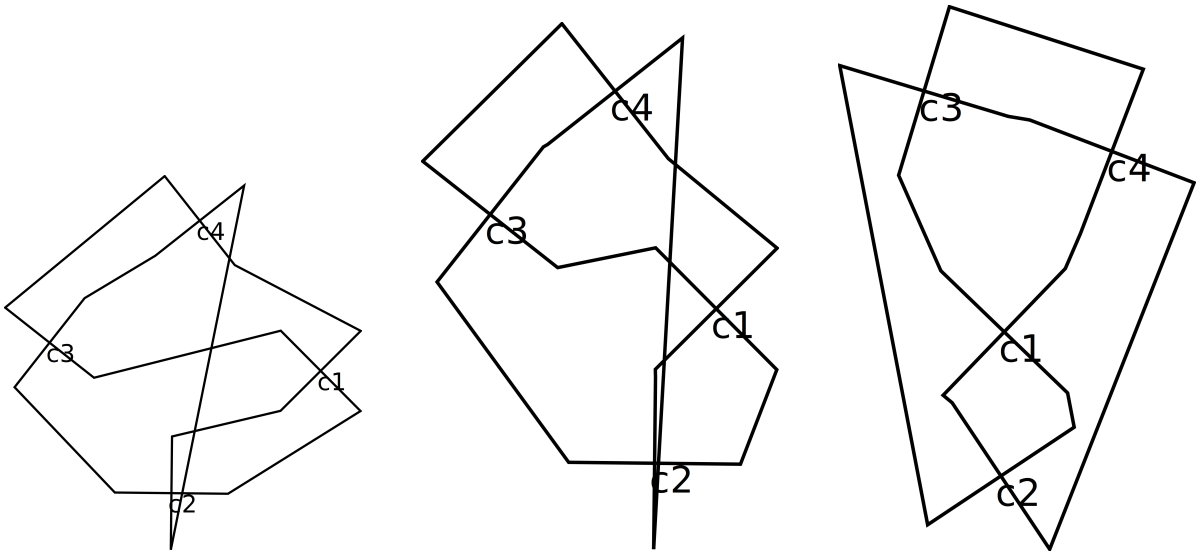
Manche Überkreuzungen gehören in die Mitte von Knoten, am Rand oder an anderen bestimmten Positionen. Sollten sie dort nicht vorzufinden sein, macht sich das bemerkbar. Ihre zugehörigen Kanten überdehnen sich, um weiterhin orthogonal in den Knoten eintreten zu können. Die Kontrollpunktwinkel sind weiterhin zu spitz und selbst eine Rotation könnte sie nicht genug verbessern, ohne einen anderen Kontrollpunktwinkel des Knotenpunktes zu verschlechtern. Eine stetige Verschiebung

des Knotenpunktes, um diese Winkel zu entspannen, könnte im besten Fall dazu führen, dass die Knotenpunkte eine für sie günstigere Position einnehmen, wo alle Kontrollpunktwinkel möglichst groß sind. Aber nicht jeder Knotenpunkt sollte diese Methode anwenden dürfen. Das würde dazu führen, dass bereits anschauliche Knotendiagramme mit einer guten Positionierung, wie zum Beispiel der Trefoil, eine für den Knoten untypische Verformung annehmen würde. Daher soll eine Verschiebung nur bei Knotenpunkten stattfinden, die einen gewissen Schwellenwert für die entsprechenden Winkel unterschreiten, nach dem schon eine Rotation nicht ausgereicht hat.

Sobald Knotenpunkte verschoben werden, braucht es Restriktionen, die den Bewegungsfreiraum festlegen. Hierfür werden die Distanzen zwischen den Knotenpunkten betrachtet. Eine Mindestdistanz muss eingehalten werden, damit die Knotenpunkte und ihre verbundenen Komponenten sich nicht mit anderen überschneiden. Theoretisch wäre es bei bestimmten Knoten möglich einen Knotenpunkt immer weiter von seinen Nachbarn zu distanzieren, um seine Winkel immer mehr zu entspannen. Dies würde zwar den Stress, den die Winkel verursachen reduzieren, aber nicht der Optik eines Knotendiagramms entsprechen. Daher muss es auch eine Höchstdistanz geben, die etwas Freiraum für die Optimierung bereitstellt, jedoch unübersichtlich langgezogene Graphen verhindert. Beim über- oder unterschreiten dieser Distanzen wird zusätzlich Stress für diesen Knotenpunkt erzeugt, um den Algorithmus dazu zu bringen, eine Verschiebung in diese Richtungen zu vermeiden, während aber weiterhin das Ziel verfolgt wird, die Winkel zu vergrößern.

Nachdem das Umplatzieren abgeschlossen ist, wäre auch eine weitere Rotationsphase aller Knotenpunkte sinnvoll, um die Winkel nochmal an die neue Ausgangslage anzupassen. Das kann natürlich dafür sorgen, dass sich Knotenpunkte weiter in Richtungen verschieben können, die vorher noch ungünstig waren. Und auch wie bei der Rotation sind die Knotenpunkte in der Reihe nacheinander dran. Sobald der letzte Punkt verschoben wurde, wäre beim ersten eventuell erneuter Optimierungsbedarf. Die Abfolge von Rotation und Verschiebung kann beliebig häufig wiederholt werden, um einen möglichst optimalen Zustand zu erreichen, würde aber die Laufzeit entsprechend erhöhen. Bei simpleren Knoten wie dem Trefoil wäre das nicht nötig, könnte aber bei komplexeren Knoten durchaus helfen. Hierfür braucht es eine Abbruchbedingung, bei der die Veränderung im Stresswert gemessen wird und die Schleife beendet, sobald sich kaum noch etwas ändert. Zusammen mit einer maximalen Anzahl Iterationen sollte diese Einstellung für jeden Knoten individuell getroffen werden.

Welchen Einfluss die Verschiebung auf einen Knoten haben kann, wird in Abbildung 4.6 aufgezeigt. Da bei dem Trefoil keine Verschiebung notwendig ist, wird hierfür der Achterknoten stattdessen betrachtet. Abbildung 4.6a zeigt den Achter-



(a) Achterknoten nach der Initialisierung gefolgt von einer Rotationssequenz

(b) Durchführung einer Verschiebungssequenz auf jeden Knotenpunkt

(c) 60 Abfolgen von Verschiebung mit anschließender Rotation

Abbildung 4.6. Erzeugung eines Achterknotens mithilfe der Verschiebungsmethode. Ein Knotenpunkt wird dabei 200 mal um 1 bis 2 Pixel angepasst. Der Knotenpunkt wurde in Orange und die Kontrollpunkte in Blau gekennzeichnet.

knoten nach der Initialisierung durch den Stress-Algorithmus, gefolgt von einer Rotation jedes Knotenpunktes. Die Rotation allein reicht nicht aus, um den Knoten in seine erkennbare Form zu bringen. Besonders bei $c1$ kann erkannt werden, wie eine Umpositionierung in die Mitte des Knotens nicht nur seine, sondern auch die Winkel der anderen Knotenpunkte verbessern würde. Die Knotenpunkte sind für die Übersicht wieder in Orange und die Kontrollpunkte in Blau eingefärbt.

In Abbildung 4.6b wurde der Schwellwert für Winkel so angepasst, dass jeder Knotenpunkt vom Algorithmus einmal verschoben wird. Die Verschiebungssequenz erfolgt innerhalb von 200 Iterationen, bei dem pro Iteration die x- und y-Koordinaten um 1 vor bzw. -2 Pixel zurückverschoben werden. Diese Werte wurden vorläufig gewählt, um das Prinzip vorzustellen, werden aber in Kapitel 5 noch genauer zugeschnitten. Insgesamt wurde $c1$ tatsächlich mehr zum Zentrum gebracht, während $c2$ etwas weiter nach rechts gewandert ist. Die Knotenpunkte $c3$ und $c4$ sind weiter zusammengerückt. Nun gibt es jedoch einen sehr spitzen Winkel in $c2$, wodurch die Kante direkt durch den gesamten Graphen verläuft. Also reicht eine Reihe von Verschiebungen auch nicht aus, um dieses Problem zu lösen.

Für den in Abbildung 4.6c dargestellten Graphen wurden nun weitere Rotationen

einbezogen, gefolgt von noch weiteren Verschiebungen. Die Abfolge ist dabei immer eine Verschiebung und anschließender Rotation aller Knotenpunkte. In der Summe haben 60 dieser Abfolgen das Knotendiagramm in die gezeigte Form gebracht. Hier konnte nun $c1$ vollständig in die Mitte gebracht werden mit $c2$ direkt darunter. $c2$ hat sich weiter gedreht, sodass $c1$ sehr nahe herangebracht werden konnte, was mit den Verflechtung aus bekannten Darstellungen vom Achterknoten übereinstimmt. Da $c3$ und $c4$ sich ebenfalls rotiert haben, konnten sie durch Verschiebungen auf ein Level gebracht werden. Insgesamt wurde das Problem durch die durchgezogene Kante gelöst und es hat sich ein Knoten ergeben, der durchaus die Merkmale eines Achterknotens besitzt.

4.5. Hinzufügen weiterer Kontrollpunkte

Wie in der Abschnitt 4.4 zu erkennen war, kann eine Abfolge von Rotation und Verschiebung, gute Ergebnisse für Knotendiagramme liefern. Nachdem diese Sequenzen abgeschlossen wurden, sind aber noch weitere Verbesserungen möglich und teilweise nötig. So können extra Kontrollpunkte auf den längeren, äußeren Kanten hinzugefügt werden, um Winkel weiter zu entlasten und einen größeren Bogen erzeugen zu können.

Hierfür muss als erstes ermittelt werden, ob ein zusätzlicher Kontrollpunkt von Vorteil ist. Ein Schwellwert für Winkel an der betrachteten Kante ist dafür ausreichend. Sobald dieser unterschritten ist, werden die Koordinaten der beiden existierenden Kontrollpunkte der Kante benutzt, um die Startposition des neuen Kontrollpunktes zu berechnen. Dieser soll sich zwischen den beiden anderen befinden und wird dementsprechend auch genau mittig von denen platziert. Von dort aus muss er jetzt weiter verschoben werden, und zwar in die Richtung, wo die Winkel an den anderen beiden Kontrollpunkten vergrößert werden. Dafür kann die gleiche Herangehensweise, wie zur Verschiebung der Knotenpunkte, verwendet werden. Also pixelweise in kleinen Schritten auf der x- und y-Achse, über mehrere Iterationen hinweg.

Auch hier müssen die Distanzen zu den anderen zwei Kontrollpunkten berücksichtigt werden. Am besten wäre es, wenn die Abstände zu den zwei äußeren Kontrollpunkten gleich bleiben, sodass der neue mittlere Kontrollpunkt auch wirklich in der Mitte bleibt, um einen möglichst großen Verbesserungseffekt zu erzielen. Zu weit entfernen darf der mittlere Kontrollpunkt sich aber auch nicht, da sonst das gleiche Problem entsteht, wie es auch schon in Abschnitt 4.4 erläutert wurde. Denn theoretisch würden die Winkel immer mehr entlastet werden, umso weiter der neue Kontrollpunkt sich distanziert, wäre aber nicht zielführend für das Gesamtbild des Knotens.

Die Technik der zusätzlichen Kontrollpunkte wurde in Abbildung 4.7 einmal auf das Trefoil angewandt, bei dem zuvor ein kompletter Stressminimierungsablauf stattgefunden hat. Durch den benötigten Schwellwert, sind nur die drei äußeren der insgesamt sechs Kanten erweitert worden. Die inneren drei Kanten brauchen keinen weiteren Kontrollpunkt, da sie nahezu über direktem Wege Nachbarknotenpunkte verbinden. Im Vergleich zu dem in Abbildung 5.6a gezeigten Trefoil, haben jetzt die drei äußeren Kanten jeweils einen neuen Kontrollpunkt in die Mitte gesetzt bekommen. Dieser wurde daraufhin über 50 Iterationen hinweg verschoben, um die Winkel an den anderen beiden äußeren Kontrollpunkten zu vergrößern, wobei abgebrochen wird, sobald eine gesetzte Maximaldistanz erreicht wurde. Das Ergebnis sind ausschweifendere Bögen, welche die Form eines Trefoils wesentlich besser definiert.

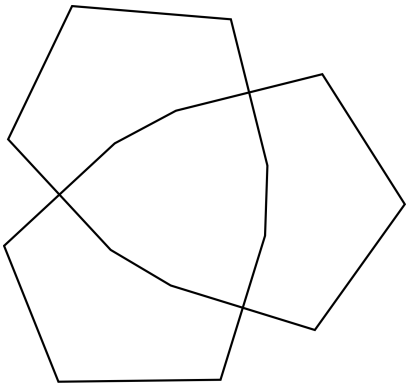


Abbildung 4.7. Trefoil mit neu hinzugefügten und bewegten Kontrollpunkten nach Stressminimierungsprozess.

4.6. Vermeidung von Kollision

Neue Kontrollpunkte entscheiden über den Verlauf, den die Kurve annimmt. Es kann nach der Stressreduktion immer noch passieren, dass Kanten sich ungünstig mit anderen Komponenten überlappen, wodurch sich Kanten potenziell überschneiden. Dies darf nicht passieren, da es den Kontext des Knotendiagramms verfälschen würde.

Zum Erkennen dieses Problems müssen alle Kanten nach Schnittpunkten mit anderen Kanten überprüft werden. Sobald eine Überschneidung gefunden wurde, stellt sich die Frage, welche der Kanten nun bewegt werden sollte. Aus der Perspektive des Algorithmus ist das nicht so leicht zu beantworten, wie für einen Menschen. Es lassen sich aber zwei Aussagen über die Anzahl der Schnittpunkte treffen.

Wenn zwei Schnittpunkte mit einer anderen Kante existieren, handelt es um eine Art Schlaufe, die über die andere Kante hinausragt. Hier könnte eine Rotation oder eine kleine Verschiebung der Kontrollpunkte potenziell schon ausreichen, um die Schlaufe auf die richtige Seite der betroffenen Kante zu ziehen.

Wenn nur ein einziger Schnittpunkt mit einer anderen Kante existiert, ist die Überschneidung derzeit notwendig, um die Zielknotenpunkte der Kanten zu erreichen. In diesem Fall wird es schwieriger einen konkreten Lösungsansatz zu finden. In einigen

Situationen kann auch eine Rotation, gegebenenfalls mit einer Verschiebung des Knotenpunktes den Konflikt bereits lösen. Eventuell muss aber eine Kante komplett um den Rest des Graphens herumgelegt werden, besonders wenn sich die verbundenen Knotenpunkte am äußeren Rand des Knotens befinden, wodurch die Kante sonst den Weg durch die Mitte nimmt und somit mehrere andere Kanten durchkreuzt. Der extra Bogen um die anderen Komponenten wird erreicht, indem der zusätzliche Kontrollpunkt entsprechend verlegt und noch weitere Kontrollpunkte der Kante angefügt werden.

In jedem der Fälle müssen die Komponenten korrigiert werden, wodurch auch das erneute Anwenden der Stressminimierung eine Option wäre. Solange Schnittpunkte von Kanten ermittelt werden, haben die dazugehörigen Komponenten einen hohen Stresswert. Die eingeführten Verfahren versuchen nun diese Komponenten anzupassen, bis die Schnittpunkte behoben wurden. Allerdings hat diese Veränderung möglicherweise einen Einfluss auf den restlichen Teil des Knotendiagramms. Neue unerwünschte Schnittpunkte könnten entstehen, während die bisherigen gelöst werden. Oder eine Ausrichtung, welche bislang akzeptabel war, könnte danach wieder ungültig sein und bedarf selbst einer erneuten Stressminimierung. Das würde sich wie eine Kaskade durch den gesamten Graphen ziehen, bis sich eine neue stabile Lage entwickelt. Nicht nur wäre es eine erheblich längere Laufzeit, je nach Komplexität des Knotens, es wäre auch nicht gewährleistet, dass die Visualisierung weiterhin ähnlich zum vorgesehenen Knoten ist. Die ganzen weiteren Veränderungen haben wahrscheinlich negative Auswirkungen auf das Ergebnis.

In solchen Situationen könnte ein manuelles Anpassen von Knotenpunktpositionen unabdingbar für den gewünschten Knoten sein und sollte den Nutzer zur Verfügung stehen, falls der Algorithmus scheitert.

4.7. Akkurate Krümmungen und Überkreuzungen

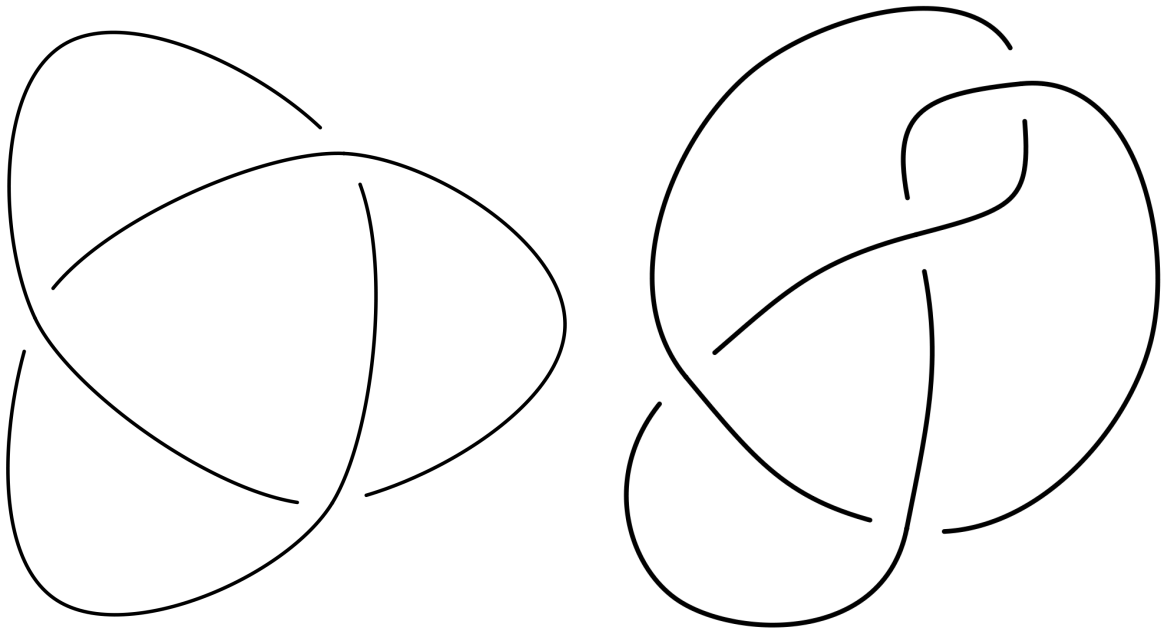
Der aus den verschiedenen stressminimierenden Verfahren resultierende Graph hat nun optimalerweise eine ähnliche Form des beabsichtigten Knotens und keine Überschneidungen von Kanten. Als letzten Schritt fehlt nun das Hinzufügen von wichtiger Merkmale, die den Graphen in ein vollständiges Knotendiagramm verwandelt.

Wie bereits in Abschnitt 4.1 erwähnt, sollen aus den Kanten mithilfe der Kontrollpunkte Bézierkurven berechnet werden, um letztlich wie realistische Bögen eines Knotens auszusehen. Des Weiteren sollen nun die Überkreuzungen richtig dargestellt werden, bei dem eine der Kanten über die jeweils andere verläuft. Hierfür wird gängigerweise eine Lücke gelassen, die für die unterführende Kante steht. Das soll auch in dieser Visualisierung genau so umgesetzt werden.

Nun ermöglicht ELK diese Funktionen allerdings nicht. ELK ist für die Berechnung der Attribute aller Komponenten des Graphens verantwortlich, weshalb bereits im Algorithmus die Knotenpunkte auf eine Größe von einem Pixel reduziert werden konnten. Damit berühren sich die Kanten direkt und bilden so eine durchgängige Linie. Das Erzeugen der Bézierkurven erfolgt jedoch erst beim eigentlichen Rendering des Diagramms durch KLighD. Daher muss das genaue Design von Kanten außerhalb von ELK definiert werden.

Die ELK-Textdatei (ELKT) bietet ein paar weitere Optionen, darunter auch das Aussehen der Kanten, als orthogonale Linien, abgerundete Kanten, oder Bézierkurven. Für die Lücken ist dieses Dateiformat aber zu unflexibel, weshalb stattdessen auf eine kgt-Datei gewechselt wird. Es gilt den Inhalt der ELKT in die kgt-Variante zu konvertieren, ehe der Renderingstil für Knotendiagramme eingestellt wird. Der Knoten-Algorithmus wird als Layout vorgegeben, Bézierkurven für das Kantenrouting festgelegt und ein eigener Stil für die Kanten erstellt. Die Funktion eigene Muster, wie gestrichelte Linien, zu designen wird hierbei für die Lücken genutzt. Der Anfang jeder Kante soll bis zu einer festen Länge ausgeblendet werden, und von dort aus normal durchgezogen weiterlaufen. Zusätzlich erhalten die Kanten noch abgerundete Enden, damit der Übergang von einer Kante zur nächsten möglichst nahtlos ist.

Das Endergebnis dieser genutzten Technik wird in Abbildung 4.8 gezeigt. Hier wurden die Knotendiagramme eines Trefoils und eines Achterknotens mit der in diesem Kapitel erläuterten Methoden automatisch erstellt. Dabei wurden die aus Abschnitt 4.5 vorgestellten Kontrollpunkte nicht verwendet. Die erweiterten Rendering-Optionen sorgen für runde Bögen und Lücken an den ausgehenden Kanten jedes Knotenpunktes. Dabei ist noch zu erwähnen, dass der Nutzer beim Erstellen des Knotens auf das korrekte Verbinden der Knotenpunkte mit jeweils zwei eingehenden und zwei ausgehenden Kanten achten muss.



(a) Automatisch erstelltes Knotendiagramm eines Trefoils.

(b) Automatisch erstelltes Knotendiagramm eines Achterknotens.

Abbildung 4.8. Graphen eines Trefoil und Achterknoten, welche nach der Stressminimierung mit erweiterten Rendering-Optionen zu richtigen Knotendiagrammen wurden.

Kapitel 5

Erstellen eines neuen Layout-Algorithmus

All die in Kapitel 4 besprochenen Konzepte und Ansätze müssen bei der Entwicklung eines neuen Layout-Algorithmus berücksichtigt werden. In diesem Kapitel geht es primär um die Implementierung in ELK, weshalb es bereits eine bestehende Struktur für Graphen und zugehörigen Komponenten, sowie eine Einbindung anderer Algorithmen oder Methoden gibt. Ein weiterer Vorteil von ELK ist das einfache Übertragen von vom Nutzer vorgegebenen Werten auf die genutzten Parameter.

Insgesamt soll die Implementierung die Vielfalt der von ELK angebotenen Layout-Algorithmen zu erweitern. Es werden auf Probleme bei der Umsetzung einiger Funktion eingegangen, sowie dessen Lösungen beschrieben. Dabei ist ein Problem nur im spezifischen Kontext der Visualisierung durch KLighD entstanden. Ungeachtet dessen sind diese Techniken aber auch in einem reinen, alleinstehenden Algorithmus denkbar, bei dem mit objektorientierter Programmierung die Komponenten eines Graphen umgesetzt und verwendbar gemacht wurden.

5.1. Eingliederung im ELK

Als ein neues Feature des Eclipse Layout Kernel, wird die Implementierung aus hauptsächlich zwei Dateien bestehen. Die Javadatei `KnotLayoutProvider.java` beinhaltet den eigentlichen Algorithmus, der von dort aus in das gesamte ELK-Projekt integriert wird. Des Weiteren lassen sich somit Abhängigkeiten von anderen Algorithmen übertragen und ermöglicht es so, bereits bestehende Methoden aus anderen existierenden Algorithmen wiederzuverwenden. Durch diese Einbindung erhält der neue Algorithmus Zugriff auf den in Abschnitt 4.2 erwähnten Layout-Algorithmus vom Stresslayout. Dieser wird in der Initialisierung verwendet und ist entsprechend kein eigens entwickelter Teil dieser Arbeit.

ELK enthält bereits Interfaces und Klassen, die für das Erstellen und Manipulieren von Graphen ausgelegt sind. Instanzen von `ElkNode`, `ElkEdge` oder `ElkBendpoint`, sowie der gesamte Graph als hierarchisch übergeordneter `ElkNode` sind dementspre-

chend Objekte, welche häufig in der Implementierung verwendet werden. Über die Instanzen können die Attribute der Komponenten eines Graphens auf direktem Wege geändert werden, ermöglichen es aber auch, dass zukünftige Features bei Bedarf ebenfalls von den Methoden des Knotenlayouts Gebrauch machen können.

In `Knot.melk` werden ein paar Standardparameter für das Layout sowie eine Beschreibung festgelegt. Hier werden auch Layoutoptionen und deren Standardwerte definiert, die vom Knotenlayout dann zur Verfügung gestellt werden. Einige ausgewählte Optionen vom Layered-Layout werden ebenfalls unterstützt. Die Dateien `KnotOptions.java` und `KnotMetadataProvider.java` werden beim Bauen des Projektes aus der `Knot.melk` automatisch erzeugt und dienen zur Einbettung der unterstützten Optionen, die beim Ausführen des Programms vom Nutzer angesprochen werden können.

Abbildung 5.1 zeigt, die hierarchische Struktur des Projektes und wo das Knotenlayout eingegliedert ist. Viele weitere Dateien werden beim Erstellen eines neuen Algorithmus automatisch generiert.

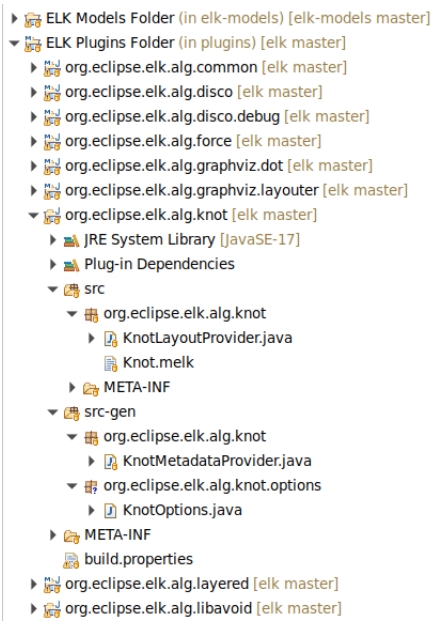


Abbildung 5.1. Eingliederung in die Ordnerstruktur des ELK-Projektes

5.1.1. Debugging mit Snapshots

Für das Debuggen des Algorithmus kann der von ELK zur Verfügung gestellte `progressMonitor` genutzt werden. Während des Layoutings werden zu bestimmten Zeitpunkten Snapshots des Graphens gemacht, wie dieser zu der Zeit aussah. Auch kann hierüber die Laufzeit zu Erstellung des Graphens eingesehen werden.

In der Implementierung erfolgt der erste Snapshot direkt nach der Initialisierung durch das Stresslayout inklusive des Hinzufügens der Kontrollpunkte an jeden Knotenpunkt. Ein weiterer wird nach der ersten stressreduzierenden Rotation gemacht. Während der Hauptschleife wird in jeder Iteration eine Verschiebung und eine Rotation durchgeführt, wo anschließend der `progressMonitor` auch den Graphen loggt. Dafür wird Nummer der aktuellen Iteration mit gespeichert. Anschließend erfolgt noch eine Abschlussrotation nach der letzten Verschiebung in der Schleife, die ebenfalls noch geloggt wird. Der letzte Snapshot wird dann nach dem Hinzufügen

der mittleren Kontrollpunkte gemacht, welche dann auch den Abschluss und somit den fertigen Graphen darstellt.

Für die Verwendung dieser Snapshots oder gemessenen Laufzeiten müssen in der laufenden Anwendung die Entwickleroptionen „`Generate debug information`“und „`Measure execution times`“aktiviert werden. Nun gibt es zurzeit aber ein Problem, beim Anzeigen der Kanten in dem Debug-Fenster. Neue Positionen von Knotenpunkten werden nicht berücksichtigt und die Kanten verlaufen zu den alten Positionen. Das macht die Fehlersuche etwas umständlicher, der fertige Graph ist allerdings nicht betroffen.

5.2. Voraussetzungen

Aufgrund der Funktionsweise des Algorithmus, sind bestimmte Voraussetzungen nötig, die beim Erstellen eines Knotens mittels ELK-Textdatei oder kgt-Datei beachtet werden müssen. Jeder Knotenpunkt besitzt genau vier Kanten, von denen zwei ausgehende Kanten und zwei eingehende Kanten sein müssen. Es kann sich so vorgestellt werden, dass die ausgehenden Kanten die unterführende Achse darstellt, welche dann in der kgt-Variante, als Lücke visualisiert wird. Die eingehenden Kanten stellen dementsprechend die überliegende Achse dar.

Für die in diesem Kapitel vorgestellten Methoden ist die Unterscheidung von ausgehenden und eingehenden Kanten wichtig, um die zwei überkreuzenden Achsen zu differenzieren, damit die Orthogonalität der Kanten gegenüber der Knotenpunkte gewährleistet wird. Vor Ausführung des eigentlichen Algorithmus wird eine `IllegalArgumentException` geworfen, sollte die Voraussetzung nicht erfüllt sein. Genauer, muss für jeden Knotenpunkt gelten:

```
node.getOutgoingEdges().size() == 2 \wedge node.getIncomingEdges().size() == 2
```

5.3. Initialisierung des Graphen

Der vom Nutzer übergebene Graph wird zunächst auf die in Abschnitt 5.2 besprochene Voraussetzung überprüft. Dabei werden sich alle Knotenpunkte angeschaut und die Anzahl der ausgehenden und eingehenden Kanten gezählt. Danach erfolgt eine erste Positionierung mit dem Stresslayout. Dafür wird der Graph aufgeteilt und dem Stress-Algorithmus übergeben, um eine gleichmäßige Ausrichtung der Knotenpunkte zu erhalten.

Noch existieren keine Kontrollpunkte auf den Kanten, also müssen der Reihe nach neue hinzugefügt und um die entsprechenden Knotenpunkte richtig platziert

werden. Es ist wichtig, dass pro Knotenpunkt nur die ausgehenden Kanten behandelt werden, sodass die Reihenfolge der Kontrollpunkte entlang der Kante korrekt ist und keine Kontrollpunkte mehrfach erzeugt werden. Für die relative Position des ersten Kontrollpunktes einer Kante werden die Koordinaten des Startknotenpunkts verwendet, während die Position des zweiten bzw. letzten Kontrollpunktes abhängig vom Zielknotenpunkt ist. Für das einfachere Differenzieren werden diese Kontrollpunkte im Weiteren auch als Startkontrollpunkt und Zielkontrollpunkt bezeichnet. Die Startkontrollpunkte werden entlang der y-Achse als Erstes und die Zielkontrollpunkte entlang der x-Achse nach allen Startkontrollpunkten platziert.

Der Parameter `nodeRadius` bestimmt hierfür den Abstand zum Knotenpunkt, der bei allen zukünftigen Transformationen beizubehalten werden muss. Wenn der Radius von Kontrollpunkt und Knotenpunkt erhöht wird, vergrößert sich das gesamte Knotendiagramm.

5.4. Methoden zur Stressberechnung

Nachdem die Knotenpunkte durch das Stresslayout positioniert und von den Kontrollpunkten umkreist sind, können jetzt die Methoden zum Verschieben oder Drehen angewandt werden. Zuvor braucht es aber die Kriterien, anhand denen die Manöver ausgeführt werden. Der Stresswert wird erzeugt, sobald ein Kriterium nicht erfüllt ist und wird je nach Abweichung vom gewünschten Wert oder Zustand immer stärker.

5.4.1. Stress durch spitze Winkel

Zu den wichtigsten Kriterien eines Knotendiagramms, so wie er in ELK dargestellt wird, sind die Winkel an den vier Kontrollpunkten eines Knotenpunktes. Hierfür können Vektorfunktionen genutzt werden, welche einen Winkel zwischen zwei Vektoren ermitteln. Die Vektoren werden von den Kontrollpunkten und den Knotenpunkten aufgespannt, wodurch eine Überprüfung notwendig ist, um welche Art Kante es sich gerade handelt.

Bei ausgehenden Kanten wird der Winkel am Startkontrollpunkt verlangt. Jede Kante hat eine Liste ihrer Kontrollpunkte, also wäre es in diesem Fall der Kontrollpunkt am Index 0. Ein Vektor führt von der Position dieses Kontrollpunktes zur Position des nächsten Kontrollpunkts, also dem mit Index 1. Da es sich um die ausgehende Kante handelt, ist der vorherige Punkt der Startknotenpunkt selbst. Der zweite Vektor führt vom Startkontrollpunkt zum Startknotenpunkt der Kante. In Abbildung 5.2 sind die zwei Vektoren in Grün markiert und der zu ermittelnde Kontrollpunktwinkel wird in Rot dargestellt.

Für eingehende Kanten funktioniert das Prinzip ähnlich. Hier ist der Winkel am Zielkontrollpunkt gefragt, welcher sich auch an der letzten Position in der Liste befindet. Der sicherste Zugriff auf den Index ist `bps.size()-1`, sollte diese Abfrage mit einer größeren Anzahl Kontrollpunkten genutzt werden. Somit ist der vorherige Kontrollpunkt an der Stelle `bps.size()-2` und der nächste Punkt ist der Zielknotenpunkt selbst. Wieder werden die zwei Vektoren aufgespannt und der Winkel ermittelt.

Die Liste von Kontrollpunkten heißt `bps` und `bendPoint` ist der Kontrollpunkt, von dem der Winkel gefragt ist. Der Algorithmus arbeitet mit Winkeln in Grad.

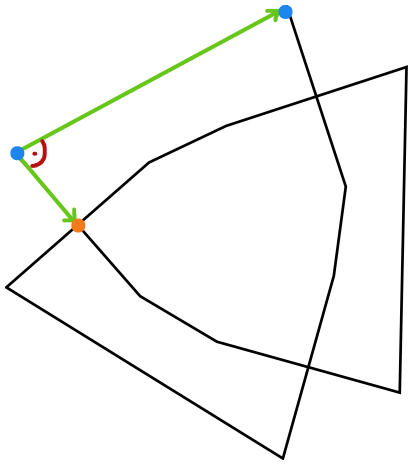


Abbildung 5.2. Die grünen Vektoren werden genutzt, um an einem Kontrollpunkt im Trefoil den Winkel zu berechnen.

```
double x = bendPoint.getX();
double y = bendPoint.getY();
int i = bps.indexOf(bendPoint);

if(i == 0) {
    ElkNode sNode = (ElkNode) edge.getSources().get(0);
    prevPoint = new KVector(sNode.x - x, sNode.y - y);
    nextPoint = new KVector(bps.get(i+1).x - x, bps.get(i+1).y - y);
} else if (i == bps.size()-1) {
    ElkNode tNode = (ElkNode) edge.getTargets().get(0);
    prevPoint = new KVector(bps.get(i-1).x - x, bps.get(i-1).y - y);
    nextPoint = new KVector(tNode.x - x, tNode.y - y);
}
angle = Math.toDegrees(nextPoint.angle(prevPoint));
```

Wie in **??**chap:konzept) erklärt wurde, sollen spitze Winkel vermieden werden, da davon ausgegangen werden kann, dass diese Kanten zurück zum verbundenen Knotenpunkt führen und dabei sehr wahrscheinlich andere Komponenten schneidet. Wenn überall große Winkel gewährleistet sind, stehen die Richtung, aus der die Kante aus dem Startknotenpunkt austritt, und die Richtung, in der sie in den Zielknotenpunkt eintrifft, nahezu gegenüber. Dadurch hat die Kante einen möglichst einfachen Weg. Der Zielwinkel ist somit 180° und der Stresswert die Differenz zu dem aktuell ermittelten Winkel. Da ein Winkel von nahezu 0° nicht existieren darf, wird noch eine

Potenzfunktion angewandt, wodurch der Stresswert bei dieser Ausrichtung enorm ansteigt und den Algorithmus dazu bringt einen besseren Winkel anzustreben.

Der Stresswert eines Knotenpunktes akkumuliert sich aus den Berechnungen mit dem Winkel der vier Kontrollpunkte. Da es keine Liste der gesamten Kanten gibt, werden die eingehenden und ausgehenden Kanten separat durchlaufen. Diese Unterscheidung hilft aber auch dabei, die richtigen Kontrollpunkte aus der Liste der Kontrollpunkte `bps` der jeweiligen Kante abzurufen.

```
for (ElkEdge outEdge : node.getOutgoingEdges()) {
    ElkBendPoint bp = outEdge.getBendPoints().get(0);
    angle = 180 - calculateBendPointAngle(bp);
    stress = stress + Math.pow(angle,2);
}
for (ElkEdge inEdge : node.getIncomingEdges()) {
    bps = inEdge.getBendPoints()
    ElkBendPoint bp = bps.get(bps.size()-1);
    angle = 180 - calculateBendPointAngle(bp);
    stress = stress + Math.pow(angle,2);
}
```

Sollte es gewünscht sein, können die Achsen der ausgehenden und eingehenden Kanten separat voneinander rotiert werden. Die Option dazu wird mit dem Parameter `separateAxisRotation` aktiviert. Die Stresswertberechnung durch spitze Winkel funktioniert vom Prinzip her gleich und wird lediglich auf die entsprechenden Achsen aufgeteilt. Allerdings ist noch ein weiteres Kriterium erfordert. Wenn die zwei Achsen sich unabhängig voneinander drehen dürfen, muss garantiert werden, dass sie nicht direkt übereinander liegen. Es wird ein weiterer Winkel berechnet, und zwar am Knotenpunkt direkt, wo sie sich die Achsen schneiden. Auch hier werden Vektoren erstellt, diesmal vom Knotenpunkt zu dem Startkontrollpunkt auf eine der ausgehenden Kanten und zu dem Zielkontrollpunkt auf einer eingehenden Kante. Der optimale Winkel für einen gleich großen Abstand aller vier Kanten ist 90°, so wie es bei der Rotation, ohne diese Option, standardmäßig eingestellt ist. Je kleiner beziehungsweise auch je größer der Winkel über 90° ist, desto größer wird wieder der Stresswert sein. Das Ergebnis wird dann dem Stresswert einer Achse hinzugefügt.

Der letztlich ermittelte Winkelstress wird als Vergleichswert für die Rotation und für die Verschiebung von Knotenpunkten genutzt.

5.4.2. Stress durch Knotenpunktabstand

Der zweite wichtige Stresswert entsteht durch den Abstand zwischen verbundenen Knotenpunkten. Dieser wird erst relevant, wenn Knotenpunkte auch verschoben werden sollen, da bei der Rotation der Abstand weiterhin gleich bleibt. Hierfür gibt es eine gewünschte Distanz, die eingehalten werden soll. Direkt ineinander dürfen die Knotenpunkte nicht sein, aber je nach Drehausrichtung kann derselbe Abstand wie von einem Knotenpunkt zu einem seiner Kontrollpunkte ausreichen. Also sollte er mindestens `nodeRadius` betragen, was allerdings bei bestimmten Knotendiagrammen zu wenig ist. Dementsprechend ist es sinnvoll diese Knotenpunktdistanz als Layoutoption individuell anpassbar zu machen oder eine optimale Distanz zu berechnen. Für diese Implementierung hat der Nutzer die Möglichkeit den Parameter `desiredNodeDistance` für seinen Knoten selbst anzupassen, wobei der Standardwert sonst gleich zum Parameter `nodeRadius` ist. Ein größerer forcierter Abstand wird auch zur Vergrößerung des gesamten Knotendiagramms führen.

Über die Kante kann auf die verbundenen Start- oder Zielknotenpunkte und somit ihren Koordinaten zugegriffen werden. Mit einem Vektor zwischen diesen zwei Knotenpunkten kann dann schnell der Abstand berechnet und die Differenz zum gewünschten Abstand bestimmt werden. Umso größer diese Differenz ist, umso mehr Stress soll sie verursachen, damit kleine Diskrepanzen erlaubt sind, um woanders dafür einen besseren Stresswert zu erzielen, aber zu dichte oder entfernte Knotenpunkte nicht erstrebt werden. Auch hier eignet sich eine Potenzfunktion dafür.

Aus mehreren Tests hat sich ergeben, dass bei der Potenzfunktion der Abstandsstress etwas weniger stark anziehen sollte, als der Winkelstress. Beim Verschieben wird somit eine etwas größere Priorität auf die Ausrichtung zu den anderen Knotenpunkten gelegt.

```
for (ElkEdge outEdge : node.getOutgoingEdges()) {
    ElkNode target = outEdge.getTarget();
    double dx = target.x - node.x;
    double dy = target.x - node.x;

    KVector distanceVector = new KVector(dx, dy);
    distance = distanceVector.length();
    stress =+ Math.pow(Math.abs(distance - desiredNodeDistance), 1.5);
}
```

Zusammen mit dem Winkelstress ist der Abstandsstress notwendig, damit der Graph beim Verschieben von Knotenpunkten nicht in sich kollabiert oder viel zu weit gefächert wird.

5.5. Reduzieren des Stresswerts

Nachdem die Stresswerte für alle Knotenpunkte ermittelt worden sind, brauch es jetzt Methoden, um ihn effektiv reduzieren zu können. Dies wird mit den in Kapitel 4 vorgestellten Techniken des Rotierens und Verschiebens umgesetzt.

5.5.1. Stressminimierende Rotation

Die stressminimierende Rotation versucht lediglich mit dem Drehen der Kontrollpunkte um den Knotenpunkt herum die Winkel an diesen Kontrollpunkten zu vergrößern. Das Zentrum der Rotation ist somit um den Knotenpunkt selbst und wird mittels einer Verschiebung in den Koordinatenursprung erreicht.

Zunächst wird der Stresswert vor der Rotation, wie in Abschnitt 5.4.1 gezeigt, berechnet. Dann werden die Kontrollpunkte um einen kleinen Wert im Uhrzeigersinn gedreht. Anschließend wird erneut der Stresswert mit dieser neuen Ausrichtung angeschaut und mit dem alten Wert verglichen. Wenn der neue Stresswert kleiner ist, war die Aktion erfolgreich und die Ausrichtung wird behalten. Sollte es zu einem Zuwachs an Stress gekommen sein, wird wieder entgegen dem Uhrzeigersinn zurückgedreht.

Eine der Rotationen muss etwas größer sein als die andere, damit die Drehung auch in beide Richtungen möglich ist. Hier ist die Rotation entgegen dem Uhrzeigersinn doppelt so groß, wie die erst Rotation im Uhrzeigersinn. So wird bei erfolgreicher Reduktion immer weiter in die optimale Richtung rotiert und es bei Misserfolg mit der anderen Richtung probiert. Sollte die entgegengesetzte Richtung sich ebenfalls negativ auf den Stresswert auswirken, so wird in der nächsten Iteration wieder in Richtung des Uhrzeigersinns rotiert und die vorherige Drehung wieder stückweise rückgängig macht. Über mehrere Iterationen hinweg wird der Knotenpunkt immer weiter in seine optimale Ausrichtung gebracht. Sobald diese erreicht wurde, ist jede weitere Rotation schlecht für den Stresswert, wodurch sich wiederum beide Rotationen immer wieder ausgleichen und die Ausrichtung insgesamt beibehalten wird.

Die Funktion `rotateNode` bekommt als Argumente den Knotenpunkt, dessen Kontrollpunkte rotiert werden sollen, und einen Winkel in Grad. Hier wird also um 1° vor bzw. 2° zurückgedreht. Die Schleife wird beendet, sobald die maximale Anzahl Iterationen erreicht wurde oder die Größe der Stresswertverbesserung unter einem gesetztem `epsilon` liegt.

```
count = 0;
prevStress = Double.MAX_VALUE;
```

```
do {
    prevStress = computeAngleStress(node);
    rotateNode(node, 1);
    if (prevStress < computeAngleStress(node)) {
        rotateNode(node, -2);
    }
    count++;
} while(count < iterationLimit &&
    (prevStres-computeAngleStress(node)) > epsilon);
```

Mit einer festgelegten Drehweite für die Rotation wird es bei einigen Knoten zu Problemen kommen. In Abbildung 5.3 wird ein Pentafoil gezeigt, so wie er nach einer stressreduzierenden Rotation aussieht. Vier Knotenpunkte konnten eine zufriedenstellende Ausrichtung annehmen, während bei dem Knotenpunkt $n3$ übermäßig Spitze Winkel an dessen Kontrollpunkten zu finden sind, und als Resultat die Kanten durch den Knotenpunkt selbst stoßen. Durch genaues hinschauen, kann erkannt werden, dass eine Rotation um etwa 180° das Durcheinander eigentlich beheben sollte, doch der Algorithmus hat es nicht geschafft. Das kann passieren, wenn die Startausrichtung bereits so ungünstig ist, dass jeder Versuch einer Rotation den Stresswert verschlimmert. Die Drehbewegungen haben einfach einen zu geringen Effekt.

Daher ist es sinnvoll die Höhe des Grades, in der Rotiert werden soll, abhängig vom Stresswert zu machen. Umso größer der Stress, umso weiter dreht sich der Knotenpunkt pro Iteration. Wenn der Knotenpunkt nahe einer optimalen Ausrichtung ist, nimmt der Stress ab und macht somit auch kleinere Drehbewegungen. Dies ermöglicht es die Ausrichtung feiner zu justieren während dem Problem in Abbildung 5.3 entgegengewirkt wird.

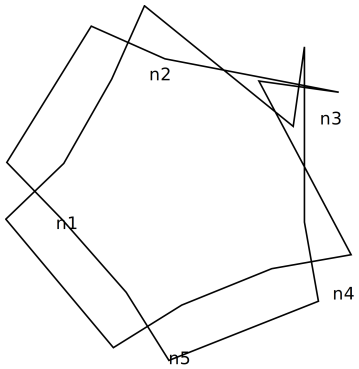


Abbildung 5.3. Pentafoil-Knoten nach der Rotation. Ein Knotenpunkt konnte sich nicht genug Rotieren.

Zusätzlich ist es praktisch dem Nutzer einen anpassbaren Parameter zur Verfügung zu stellen. Große Rotationen können das Problem feststeckender Knotenpunkte beheben, allerdings auch das Endergebnis beeinträchtigen, indem kleinere Bewegungen von etwa 1° nicht mehr möglich sind und eine optimale Ausrichtung schwerer zu erreichen ist. Mit dem Parameter `rotationValue` kann die Gewichtung

des stressabhängigen Rotationsgrades skaliert und für jedes Knotendiagramm individuell entschieden werden. Standardmäßig würde dieser sonst 1 betragen und keinen Einfluss haben.

```
prevStress = computeAngleStress(node);
angle = rotationValue*(prevStress / 2500);
rotateNode(node, angle);
if (prevStress < computeAngleStress(node)) {
        rotateNode(node, - 2*angle);
}
```

Bei der Einstellung, womit sich die Achsen der eingehenden und ausgehenden Kanten separat drehen lassen, bleibt der Prozess zur Stressreduktion grundlegend gleich und wird lediglich auf die beiden Achsen aufgeteilt. Zuerst werden die Startkontrollpunkte der ausgehenden Kanten vor beziehungsweise zurückgedreht und anschließend die Zielkontrollpunkte der eingehenden Kanten. Der zusätzliche Stress durch den Winkel der beiden Achsen zueinander muss auch im Vorher-Nacher-Vergleich beachtet werden.

5.5.2. Stressminimierende Verschiebung

Wenn die gesamten Winkel eines Knotenpunkts sich unter dem gesetzten Schwellwert `shiftThreshold` befinden, soll eine Verschiebung dieses Knotenpunktes erfolgen. Nicht nur werden die Koordinaten des Knotenpunkts angepasst, sondern auch die der vier Kontrollpunkte, denn diese müssen weiterhin ihre Position relativ zum Knotenpunkt behalten.

Wie auch schon bei der Rotation werden hier Stresswerte vor der Verschiebung mit Stresswerten nach der Verschiebung verglichen. Hierfür braucht es jetzt die Winkel an den Kontrollpunkten und den Abstand zu verbundenen Knotenpunkten, wie in Abschnitt 5.4.2 gezeigt. Zuerst erfolgt ein Schritt vorwärts auf der x-Achse. Wenn die Berechnung einen höheren Stresswert ermittelt, wird ein größerer Schritt rückwärts auf der x-Achse ausgeführt. So können Bewegungen in beiden Richtungen vorgenommen werden, je nachdem, wo sich der geringere Stresswert erzeugen lässt. Egal welcher Schritt auf der x-Achse unternommen wurde, muss diese neue Position in einer weiteren Stressberechnung berücksichtigt werden, bevor der Prozess auf der y-Achse wiederholt wird. Pro Iteration erfolgt eine Verschiebung auf der x- und y-Achse.

Für das Repositionieren des Knotenpunktes wird die Funktion `moveNode` verwendet, welche sich auch um die Kontrollpunkte kümmert. Die Argumente sind dabei der zu bewegende Knotenpunkt und die x- und y-Koordinaten der gewünschten

Position. Auch hier bricht die Schleife ab, sobald die maximale Anzahl Iterationen erreicht wurde oder die Größe der Stresswertverbesserung unter einem gesetztem `epsilon` liegt. Da pro Iteration mehrmals der Abstandsstress berechnet werden muss, erhöht sich demnach auch die Laufzeit bei steigender Anzahl von Knotenpunkten die berücksichtigt werden müssen.

```
count = 0;
prevStress = Double.MAX_VALUE;
afterStress = 0;

do {
    // Verschiebung auf x-Achse:
    prevStress = computeAngleStress(node) + computeDistanceStress(node);
    x = node.x + shiftValue*(prevStress / 5000);
    moveNode(node, x, node.y);
    afterStress = computeAngleStress(node) + computeDistanceStress(node);

    if (prevStress < afterStress) {
        moveNode(node, - 2*x, node.y);
    }

    // Verschiebung auf y-Achse
    prevStress = computeAngleStress(node) + computeDistanceStress(nod);
    y = node.y + shiftValue*(prevStress / 5000);
    moveNode(node, node.x, y);
    afterStress = computeAngleStress(node) + computeDistanceStress(node);

    if (prevStress < afterStress) {
        moveNode(node, node.x, - 2*y);
    }
    count++;
    afterStress = computeAngleStress(node) + computeDistanceStress(node);
} while(count < iterationLimit &&
    (prevStres-afterStress) > epsilon);
```

Die Verwendung einer stresswertbasierten Schrittweite, skaliert mit dem vom Nutzer anpassbaren Wert `shiftValue`, bietet ähnliche Vorteile wie bei der Rotation. Besonders, wenn der gewünschte Abstand `desiredNodeDistance` eingehalten werden soll, werden größere Bewegungen ausgeführt, um einen zu dichten Knotenpunkt auf Distanz zu bringen, oder einen weit entfernten Knotenpunkt effektiv wieder an den

Rest des Graphens heranzuholen.

Die Einhaltung des Knotenpunktabstandes steht auch teilweise in direkter Konkurrenz mit der Optimierung der Kontrollpunktwinkel. Es können Situationen entstehen in der sich ein Knotenpunkt immer weiter vom Graphen abwendet, da so die vier Kontrollpunktwinkel tatsächlich insgesamt größer werden. Um dieses Problem zu lösen ist das richtige Verhältnis zwischen den beiden Kriterien notwendig.

5.6. Weitere Kontrollpunkte

Nachdem die Stressreduktion abgeschlossen ist, kommt die Phase in der weitere Verbesserungen am Knotendiagramm vorgenommen werden können. Das jetzige Ergebnis kann unter Umständen schon zufriedenstellend genug sein, hat aber meistens das Problem, dass die äußeren Bögen des Graphens etwas zu flach sind, nachdem die Kanten so stark optimiert wurden. Zu spitze Winkel an den jeweiligen Kontrollpunkten lassen die Kante so wirken, als wäre sie überspannt. Dies ist an sich nicht falsch, allerdings sind in verschieden Darstellungen von Knotendiagrammen die Bögen, welche im Grunde die Schlaufen des Knotens darstellen sollen, eher ausschweifend. Sollte der Nutzer diese Veranschaulichung bevorzugen, kann er die Layoutoption `enableAdditionalBendPoints` aktivieren.

In diesem Verfahren wird erneut durch den Graphen iteriert und versucht auf jeder Kante einen weiteren Kontrollpunkt in der Mitte der Kante hinzuzufügen. Dieser mittlere Kontrollpunkt ermöglicht es die Kante flexibler verformen zu können. Für Kanten, die im mehr im Zentrum des Knotendiagramms liegen, ist es häufig nicht nötig, da sie einen bereits kurzen und geraden Weg von ihrem Startknotenpunkt zum Zielknotenpunkt haben. Hierfür ist der Parameter `additionalBendPointsThreshold` gedacht. Dieser bestimmt wie spitz die Winkel an den zwei zurzeit existierenden Kontrollpunkten einer Kante sein müssen, ehe ein weiterer Kontrollpunkt eingefügt wird.

Wenn die Bedingung erfüllt von einer Kante sind, werden die Koordinaten der zwei bestehenden Kontrollpunkte abgerufen. Einer befindet sich neben dem Startknotenpunkt und der andere neben dem Zielknotenpunkt und werden weitergehend auch als äußere Kontrollpunkte bezeichnet. Der Neue ist dann der mittlere Kontrollpunkt und befindet sich initial genau auf der Mitte der Kante. Damit dieser auch der Kante angehört, muss er noch an der richtigen Stelle in der Liste von Kontrollpunkten hinzugefügt werden. Andernfalls würde die Kante zwischen den vor dem Erreichen des Zielknotenpunkts noch einen schlagartigen Sprung in die Mitte machen.

Da beim regulären Hinzufügen der neue Kontrollpunkt an das Ende der Liste von Kontrollpunkten `bps` angehangen wird, muss dieser vorher unabhängig von

```
x = (outerBp1.x+ outerBp2.x)/2;
y = (outerBp1.y + outerBp2.y)/2;

middleBp = ElkGraphUtil.createBendPoint(x, y);
        bps.add(1, middleBp);
```

Komponenten über `createBendPoint` erstellt werden.

Als Nächstes gilt es den mittleren Kontrollpunkt zu verschieben, damit dieser auch einen sinnvollen Zweck erfüllt. Das Hauptziel dabei ist es, die Winkel an den äußeren Kontrollpunkten zu vergrößern, um diese potenziell überspannten Kanten einen größeren Bogen machen zu lassen. Zusätzlich muss ein gleichmäßiger Abstand zu den äußeren Kontrollpunkten bewahrt werden, damit die Kante sich am Ende nicht in eine spezifische Richtung neigt.

Diese Konditionen sind sehr ähnlich zu der Knotenpunktverschiebung und können daher auch auf ähnlichem Wege erfüllt werden. Die Winkel und Distanzen zu den äußeren Kontrollpunkten gehen in eine Stresswertberechnung ein. Dann erfolgt die Repositionierung auf der x-Achse nach dem gleichen Prinzip des Vor- und Zurückschiebens wie in Abschnitt 5.5.2 erklärt. Wenn sich der Stresswert vergrößert hat, wird der mittlere Kontrollpunkt um eine größere Schrittweite in die entgegengesetzte Richtung zurückgeschoben. Analog passiert das auf der y-Achse.

Solange der neue Kontrollpunkt in der Mitte der anderen beiden Kontrollpunkte bleibt, werden die äußeren Winkel immer geringer je weiter sich dieser distanziert. Auch wenn der Stresswert damit erfolgreich verringert wird, entstehen dadurch viel zu große Bögen und das Knotendiagramm wird in der Gesamthöhe verzerrt, etwa so wie es in Abbildung 5.4 dargestellt ist. Eine Möglichkeit dieses Problem zu beheben wäre die Distanz von der aktuellen Position zur anfänglichen Initialposition mit in den Stresswert des Kontrollpunkts einzubauen. In dieser Implementierung wurde stattdessen nun die Schleife für den Verschiebungsprozess gestoppt, sobald die gewünschte maximal Distanz gegenüber der Initialposition erreicht wurde. Der mittlere Kontrollpunkt stellt die Spitze des Bogens dar, weshalb diese Distanz auch Bogenhöhe genannt werden kann. Der Nutzer hat die Option die Bogenhöhe eines Knotens mit dem Parameter `curveHeight` nach Belieben zu ändern.

Anhand des neuen mittleren Kontrollpunktes, den zwei äußeren Kontrollpunkten und dem Start- und Zielknotenpunkt, kann theoretisch nun eine Bézierkurve über fünf Punkte konstruiert werden. Einfacher ist es allerdings diese Punkte aufzuteilen und zwei Bézierkurven zu berechnen, die dann

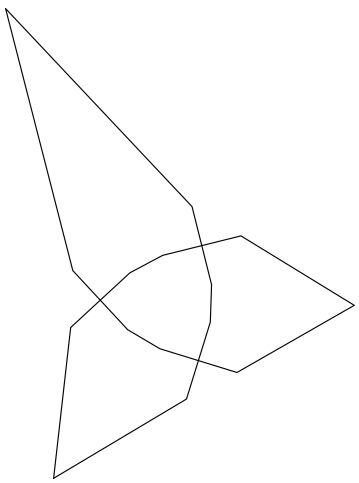


Abbildung 5.4. Trefoil mit zusätzlichen mittleren Kontrollpunkten. Um

zusammengefügt werden. Beim Rendering des Graphens durch KLighD passiert genau das, doch das führt wiederum zu einem Fehler in der Kurve. Es wird eine Bézierkurve mit vier Punkten, also vom Startknotenpunkt, über den ersten äußeren und mittleren Kontrollpunkt, bis zum zweiten äußeren Kontrollpunkt, erstellt. Diese wird dann an eine Bézierkurve mit nur zwei Punkten, also vom zweiten äußeren Kontrollpunkt zum Zielknotenpunkt, angehängt. Zwei Punkte reichen für die Berechnung einer Bézierkurve nicht aus und die Kante bleibt an dieser Stelle einfach gerade, wie in Abbildung 5.5a gesehen werden kann.

Um dieses Renderingproblem zu umgehen, müssen noch weitere Kontrollpunkte erstellt werden, damit zwei Bézierkurven über jeweils vier Punkten bestimmt werden kann. Da hierbei im Grunde zwei Kanten an einem Punkt aufeinander treffen, muss auch wieder ein sauberer und knickfreier Übergang geschaffen werden, so wie es in den Knotenpunkten selbst der Fall ist. Hierfür existieren die Kontrollpunkte um den Knotenpunkt herum, bei dem die zwei Kontrollpunkte der ausgehenden Kanten und die zwei der eingehenden Kanten direkt gegenüber voneinander liegen müssen. Dieses Konzept kann nun auch auf den mittleren Kontrollpunkt angewandt werden, bei dem zwei neu eingeführte Hilfskontrollpunkte seitlich von der Mitte platziert werden. Ein Hilfskontrollpunkt liegt somit zwischen dem ersten äußeren Kontrollpunkt und dem mittleren Kontrollpunkt, während der andere Hilfskontrollpunkt zwischen dem Mittleren und dem anderen Äußeren liegt.

5.6 zeigt wie die Hilfskontrollpunkte eingebaut werden. Um diese Positionen zu bestimmen wird ein Richtungsvektor verwendet, der zwischen den beiden äußeren Kontrollpunkten verläuft. Für den ersten Hilfspunkt wird der Vektor wird etwas kleiner skaliert und den Koordinaten des mittleren Kontrollpunkts angehangen. Beim zweiten Hilfspunkt passiert das gleiche, nur dass der Vektor einmal invertiert wird. Nun sind die Hilfskontrollpunkte direkt gegenüber und gleichzeitig parallel zum ursprünglichen Verlauf der Kante. Sie müssen nur noch an der richtigen Stelle in der Liste eingesetzt werden.

```
directionVector =
    new KVector(outerBp2.x - outerBp1.x, outerBp2.y - outerBp1.y);
directionVector.scale(curveWidthFactor);

helperBp1.set(middleBp.x+ directionVector.x, middleBp.y + directionVector.y);
```

```
directionVector.scale(-1);
helperBp2.set(middleBp.x + directionVector.x, middleBp.y + directionVector.y);

bps.add(2, helperBp1);
bps.add(1, helperBp2);
```

In Abbildung 5.5b ist der neue Verlauf der Kante zu sehen, bei dem der mittlere Kontrollpunkt lila und die Hilfskontrollpunkte grün markiert sind. Wie die zwei sauber Zusammengesetzen Bézierkurven dann eine ganze Kurve Bilder ist in Abbildung 5.5c zu sehen. Die Notwenigkeit dieser zusätzlichen Hilfskontrollpunkte ermöglicht aber auch das Einstellen der Breite einer Kurve, wie es in Abbildung 5.5d deutlich wird. Hierfür steht der Parameter `curveWidthFactor` und entspricht dem Skalierungsfaktor des genutzten Richtungsvektors. Insgesamt sind also Höhe und Breite von Bögen eines Knotendiagramms individuell anpassbar.

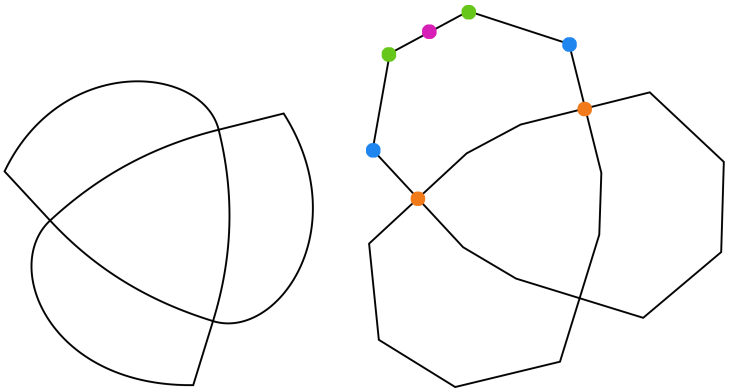
Nachdem die Kanten weiter optimiert wurden, ist der Algorithmus am Ende und das Knotendiagramm kann von ELK an KLighD zum visualisieren übergeben werden.

5.7. Hauptschleife

Die zwei stressreduzierenden Methoden reichen aus, um die grobe Form verschiedener Knotendiagramme automatisch anzuordnen. Wichtig ist dabei, sie mehrmals oder im Falle einer Verschiebung abwechselnd auszuführen. Die Hauptschleife iteriert durch jeden Knotenpunkt durch und führt diese Methoden am aktuellen Knotenpunkt aus. Sobald der letzte Knotenpunkt angepasst worden ist, könnte der erste Knotenpunkt womöglich erneut verbesser werden.

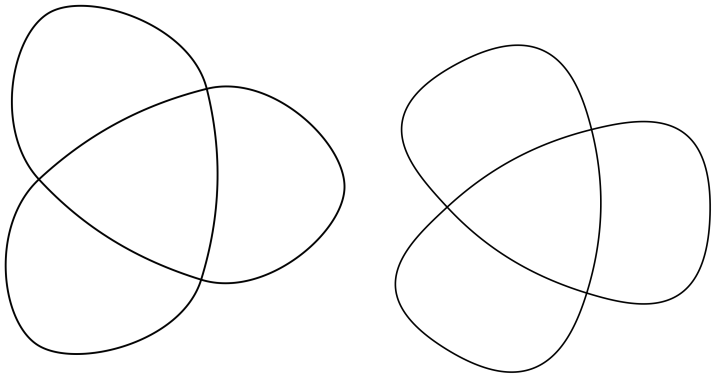
```
int count = 0;
do {
    for(node : graph.getNodes()) {
        stressMinimizingRotation(node);

        if (computeNodeAngles(node) < shiftThreshold) {
            stressMinimizingShift(node);
        }
    }
    count++;
} while(count < iterationLimit);
```

Wie genau die Schleifen verschachtelt sind, kann in 5.7 eingesehen werden. In der Hauptschleife wird der Reihe nach durch die Knotenpunkte iteriert und dabei die



(a) Renderingproblem durch Aufteilung von Kontrollpunkten

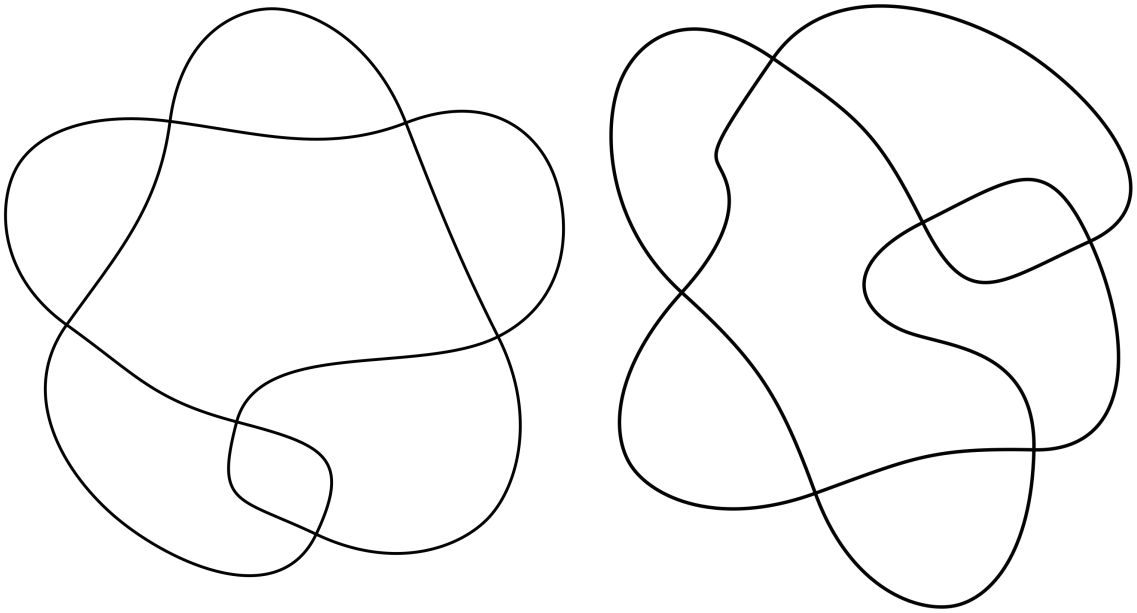
(b) Zusätzliche Hilfskontrollpunkte neben den mittleren Kontrollpunkten



(c) Zwei Bézierkurven aneinander bilden eine ganze Kurve dank Hilfskontrollpunkte

(d) Hilfskontrollpunkte ermöglichen das Einstellen breiter Bögen

Abbildung 5.5. Ein Treoil der mit zusätzlichen Kontrollpunkten umfangreichere Bögen hat.



(a) Nach einer Verschiebung erfolgte immer eine Rotation

(b) Die Rotation erfolgte immer vor der Verschiebung

Abbildung 5.6. Das finale Layout des Stevedore-Knotens mit unterschiedlichen Reihenfolgen der Stressreduktion.

stressreduzierenden Funktionen `stressMinimizingRotation` und bei Notwendigkeit `stressMinimizingShift` ausgeführt. Der Prozess geht zu Ende, sobald die maximale Anzahl an Iterationen erreicht wurde. Ein `Epsilon`-Argument ist in dem Fall nicht möglich, da es manchmal Iterationen gibt, bei dem der Stresswert sich ein wenig verschlechtert. Das kommt von der Reihenfolge der Funktionen, welche ebenfalls ein wichtiger Faktor für das abschließende Aussehen des Graphens ist. Wenn eine Verschiebung stattgefunden hat, ist eine Rotation notwendig, um sich an der neuen Position richtig zu den anderen Knotenpunkten auszurichten.

Abbildung 5.6 zeigt zwei Knotendiagramme vom Stevedore-Knoten, die nach mehreren Iterationen der zwei Stressreduktionen entstanden ist. Im linken Graph wurde die Rotation immer nach einer Verschiebung ausgeführt, während beim rechten Graphen die Rotation davor passiert. Durch die Hauptschleife werden somit aber trotzdem Rotationen bei allen außer der letzten Verschiebung ausgeführt, weshalb am Ende noch einmal eine Abschlussrotation ansteht. Insgesamt wurden bessere Ergebnisse mit dieser Variante erzielt.

Die Verbesserung der Bögen durch zusätzlichen mittleren Kontrollpunkten geschieht nach dem Stressminimierungsprozess Bei Bedarf, sind an dieser Stelle durch-

aus noch weitere Optimierungsfunktionen möglich. So gibt es bislang keine Überprüfung auf ungewollte Schnittpunkten der Kanten, wobei dieses Problem durch das Justieren von Parametern vorübergehend gelöst werden kann. Zusammen mit der Initialisierung ergibt sich eine Abfolge, die grob dem Kontrollflussdiagramm Abbildung 5.7 entspricht.

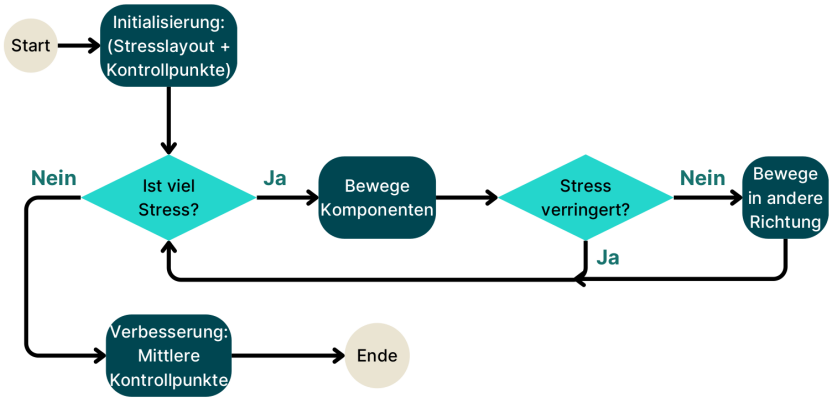


Abbildung 5.7. Abstrahiertes Kontrollflussdiagramm der Hauptschleife des Algorithmus.

5.8. Parameter des Algorithmus

Wie in den vorhergegangenen Abschnitten zu sehen war, haben sich im Verlauf der Implementierung viele Parameter ergeben, welche das Erscheinungsbild eines Knotendiagramms maßgeblich beeinflussen. Über mehrere Versuche hinweg Knotendiagramme zu gängigen Knoten generieren zu lassen, konnten keine Standardwerte gefunden werden, bei dem jedes der Diagramme ein zufriedenstellendes Ergebnis erzielte. Daher ist es sinnvoll dem Nutzer all diese Parameter als Layoutoptionen bereitzustellen, sodass potenziell jeder gewünschten Knoten mit den richtigen Einstellungen umsetzbar ist.

In der Datei `Knot.melk` werden die entsprechenden Layoutoptionen deklariert und mit einem Standardwerte versehen. Somit können die Felder der Knotenlayout-Methode mit Werten, die der Nutzer potenziell über `layoutGraph.getProperty(KnotOptions)` eingibt, definiert werden. Schnelle Änderungen an den Parameter helfen dem Nutzer die gewünschte Darstellung ihres Knotens zu erreichen, falls die Standardwerte zu unbrauchbaren Resultaten führen.

Beispiel einer Zuweisung eines Feldes vom Algorithmus:

```
this.nodeRadius = layoutGraph.getProperty(KnotOptions.NODE_RADIUS);
```

5.8.1. Liste aller für den Nutzer einstellbarer Parameter

Im Folgenden findet sich eine Liste aller anpassbarer Layoutoptionen, deren Datentyp und Standardwerte, sowie eine kleine Beschreibung ihrer Funktion.

Tabelle 5.1. Liste aller Parameter, die über die Layoutoptionen verändert werden können.

Liste der Parameter			
Parameter	**Datentyp**	**Standardwert**	**Beschreibung**
nodeRadius	double	25 Pixel	Die Abstände von Kontrollpunkten um einen Knotenpunkt herum.
shiftThreshold	double	420 Grad (aller Winkel zsm.)	Knotenpunkte mit einem gesamten Kontrollpunktwinkel unter diesem Schwellwert werden während der Stressreduktion verschoben.
desiredNodeDistance	double	nodeRadius (25 Pixel)	Die Distanz, welche zwischen verbundenen Knotenpunkten eingehalten werden soll.
separateAxisRotation	boolean	false	Ob die Achse der eingehenden und ausgehenden Kanten sich separat voneinander drehen dürfen.
rotationValue	double	1	Gewichtete Schrittweite für Rotationen.
shiftValue	double	1	Gewichtete Schrittweite für Verschiebungen.
enableAdditional- BendPoints	boolean	true	Ob der Algorithmus zusätzliche Kontrollpunkte in die Mitte von Kanten setzen darf, um größere Bögen zu schaffen.
additionalBendPoints-Threshold	double	100 Grad	Wenn der Winkel an äußeren Kontrollpunkten einer Kante unter diesem Schwellwert liegt, wird ein mittlerer Kontrollpunkt hinzugefügt.

Tabelle 5.2. Fortführung der Liste der Parameter Tabelle 5.1

Fortführend: Liste der Parameter			
Parameter	**Datentyp**	**Standardwert**	**Beschreibung**
curveHeight	double	30 Pixel	Maximale Distanz, die sich ein mittlerer Kontrollpunkt entfernen darf. Entscheidet über die Höhe eines Bogens.
curveWidthFactor	double	0.2	Skalierungsfaktor für den Abstand zwischen mittleren und Hilfskontrollpunkten. Entscheidet über die Breite eines Bogens.
iterationLimit	integer	200 Iterationen	Maximale Anzahl Iterationen für jeden Stressminimierungsprozess.

Kapitel 6

Evaluation an gängigen Knoten

Der Algorithmus ist dazu konzipiert verschiedene mathematische Knoten so darzustellen, wie es aus bekannter Literatur üblich ist. Es gibt viele Knoten in diesem Gebiet und nicht alle können getestet werden. Jedoch die wichtigsten und bekanntesten Knoten können mit der üblichen Darstellung verglichen werden.

6.1. Vergleich der automatischen Darstellung

6.1.1. Trefoil

Der Trefoil, oder auch 3_1-Knoten, besteht aus drei Überkreuzungen, die über sechs Kanten alle untereinander verbunden sind.

Für das Visualisieren wurde eine Laufzeit von 869.871ms gemessen. Die höhere Laufzeit lässt sich durch den extra Aufwand der separaten Achsen begründen.

(a) Aus der von Roflsen zusammengestellten Liste von Knoten

(b) Durch ELKs Knotenlayout-Algorithmus erzeugt

Abbildung 6.1. Vergleich des durch den implementierten Algorithmus erzeugten Trefoils.

Die folgenden Layoutoptionen wurden angepasst, um den Knoten authentischer zu gestalten:

```
algorithm=Knot
edgeRouting=splines
org.eclipse.elk.alg.knot.separateAxisRotation=true
org.eclipse.elk.alg.knot.enableAdditionalBendPoints=true
org.eclipse.elk.alg.knot.additionalBendPointsThreshold=120
org.eclipse.elk.alg.knot.curveHeight=40
org.eclipse.elk.alg.knot.curveWidthFactor=0.25
```

(a) Aus der von Roflsen zusammengestellten Liste von Knoten

(b) Durch ELKs Knotenlayout-Algorithmus erzeugt

Abbildung 6.2. Vergleich des durch den implementierten Algorithmus erzeugten Achterknotens.

6.1.2. Achterknoten

Der Achterknoten, oder auch 4_1-Knoten, ist der einzige Primknoten mit vier Überkreuzungen.

Für das Visualisieren wurde eine Laufzeit von 607.640ms gemessen.

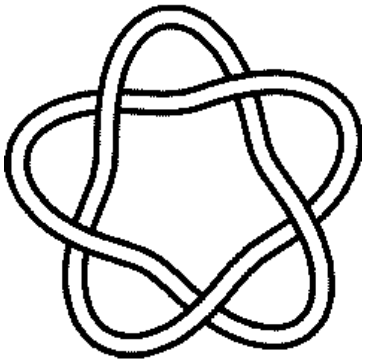
6.1.3. Pentafoil

Der Pentafoil, oder auch 5_1-Knoten, besteht aus fünf Überkreuzungen, bei dem die Knotenpunkte immer zwei verbundene Nachbarknotenpunkte haben.

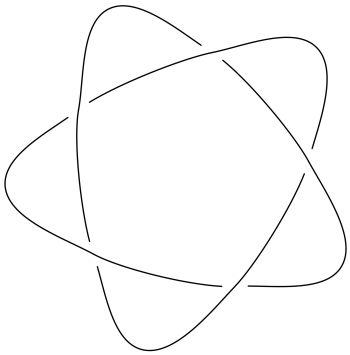
Für das Visualisieren wurde eine Laufzeit von 1861.842ms gemessen. Die höhere Laufzeit lässt sich durch den extra Aufwand der separaten Achsen begründen.

Die folgenden Layoutoptionen wurden angepasst, um den Knoten authentischer zu gestalten:

```
algorithm=Knot
edgeRouting=SPLINES
org.eclipse.elk.alg.knot.rotationValue=1.8
org.eclipse.elk.alg.knot.enableAdditionalBendPoints=true
org.eclipse.elk.alg.knot.shiftValue=2
org.eclipse.elk.alg.knot.curveHeight=15
```



(a) Aus der von Roflsen zusammengestellten Liste von Knoten

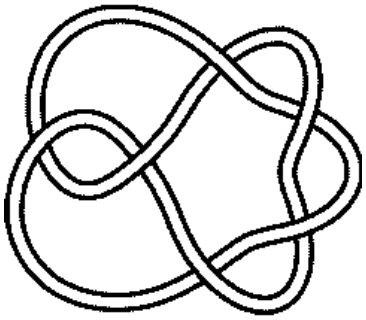


(b) Durch ELKs Knotenlayout-Algorithmus erzeugt

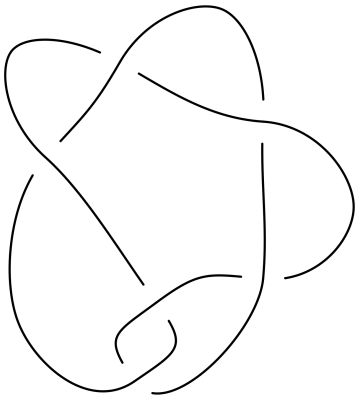
Abbildung 6.3. Vergleich des durch den implementierten Algorithmus erzeugten Pentafoils.

Die folgenden Layoutoptionen wurden angepasst, um den Knoten authentischer zu gestalten:

```
algorithm=Knot
edgeRouting=SPLINES
org.eclipse.elk.alg.knot.enableAdditionalBendPoints=true
org.eclipse.elk.alg.knot.separateAxisRotation=true
org.eclipse.elk.alg.knot.additionalBendPointsThreshold=150
org.eclipse.elk.alg.knot.curveWidthFactor=0.3
org.eclipse.elk.alg.knot.curveHeight=40
```



(a) Aus der von Roflsen zusammengestellten Liste von Knoten



(b) Durch ELKs Knotenlayout-Algorithmus erzeugt

Abbildung 6.4. Vergleich des durch den implementierten Algorithmus erzeugten Stevedore-Knotens.

Die folgenden Layoutoptionen wurden angepasst, um den Knoten authentischer zu gestalten:

```
algorithm=Knot
edgeRouting=SPLINES
org.eclipse.elk.alg.knot.rotationValue=1.5
org.eclipse.elk.alg.knot.enableAdditionalBendPoints=true
org.eclipse.elk.alg.knot.additionalBendPointsThreshold=140
org.eclipse.elk.alg.knot.shiftValue=1.2
org.eclipse.elk.alg.knot.shiftThreshold=480
org.eclipse.elk.alg.knot.curveWidthFactor=0.2
org.eclipse.elk.alg.knot.curveHeight=15
```

6.1.4. Stevedore-Knoten

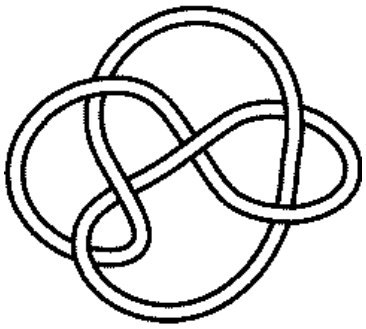
Der Stevedore-Knoten oder auch 6_1-Knoten ist der erste Primknoten mit sechs Überkreuzungen.

Für das Visualisieren wurde eine Laufzeit von 915.295ms gemessen.

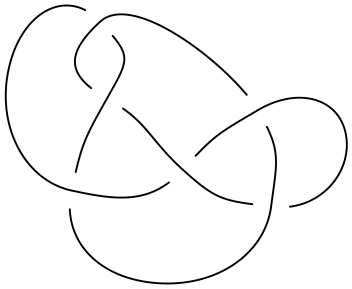
6.1.5. 6_3-Knoten

Der 6_3-Knoten ist der dritte Primknoten mit sechs Überkreuzungen.

Für das Visualisieren wurde eine Laufzeit von 1453.886ms gemessen. Die höhere



(a) Aus der von Roflsen zusammengestellten Liste von Knoten



(b) Durch ELKs Knotenlayout-Algorithmus erzeugt

Abbildung 6.5. Vergleich des durch den implementierten Algorithmus erzeugten 6_3-Knotens.

Die folgenden Layoutoptionen wurden angepasst, um den Knoten authentischer zu gestalten:

```
algorithm=Knot
edgeRouting=SPLINES
org.eclipse.elk.alg.knot.rotationValue=1
org.eclipse.elk.alg.knot.enableAdditionalBendPoints=true
org.eclipse.elk.alg.knot.shiftValue=1
org.eclipse.elk.alg.knot.shiftThreshold=500
org.eclipse.elk.alg.knot.curveWidthFactor=0.3
org.eclipse.elk.alg.knot.curveHeight=20
```

Laufzeit lässt sich vermutlich durch häufiger genutzte Verschiebungen begründen.

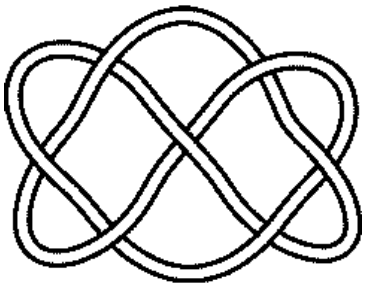
6.1.6. Endlosknoten

Der Endlosknoten, oder auch 7_4-Knoten, besteht aus sieben Überkreuzungen welche symmetrisch angeordnet sind.

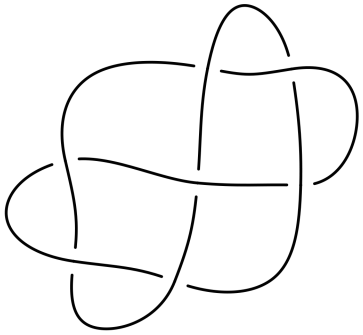
Für das Visualisieren wurde eine Laufzeit von 1282.201ms gemessen.

6.1.7. Fazit

Wie an den Beispielen erkannt werden kann, sind die durch den Algorithmus entstandenen Resultate ähnlich zu anderen Visualisierungen von Knotendiagrammen. Für ein möglichst übereinstimmendes Aussehen mussten allerdings mehrere der im-



(a) Aus der von Roflsen zusammengestellten Liste von Knoten



(b) Durch ELKs Knotenlayout-Algorithmus erzeugt

Abbildung 6.6. Vergleich des durch den implementierten Algorithmus erzeugten Endlosknotens.

Die folgenden Layoutoptionen wurden angepasst, um den Knoten authentischer zu gestalten:

```
algorithm=Knot
edgeRouting=SPLINES
org.eclipse.elk.alg.knot.nodeRadius=13
org.eclipse.elk.alg.knot.desiredNodeDistance=25
org.eclipse.elk.alg.knot.rotationValue=2
org.eclipse.elk.alg.knot.enableAdditionalBendPoints=true
org.eclipse.elk.alg.knot.additionalBendPointsThreshold=150
```

plementierten Layoutoptionen für den Knoten angepasst werden. Welche Parameter genau gebraucht sind, kann nicht gesagt werden.

Insgesamt funktioniert der Algorithmus, allerdings sollten in weiteren Arbeiten dazu bessere Standardwerte für die Parameter eruiert werden, damit möglichst viele Knoten direkt anschaulich dargestellt werden und ein experimentelles Justieren nicht nötig ist.

6.1.8. Potenzielle Probleme bei komplexen Knoten

Unter bestimmten Voraussetzungen, kann der Algorithmus keine guten Ergebnisse mehr erzeugen. Es ist denkbar, dass es bei vielen Knotenpunkten durch die Verschiebung zu einem verzerrtem Graphen kommt. Eine kleine Bewegung könnte ausreichen, damit die Stresswerte von mehreren anderen Knotenpunkten verschlechtern. Die

müssten dann auch wieder verbessert werden und wie eine Kaskade könnte dies zu einem Haufen an Verschiebungen führen, wodurch das Knotendiagramm am Ende nicht mehr erkennbar ist.

Aus Zeitgründen konnte nicht getestet werden, ob mit dem Algorithmus und den richtigen Layoutoptionen ein großer Knoten umsetzbar ist.

Kapitel 7

Konklusion

Diese Arbeit hat gezeigt, dass ein automatisches Konstruieren beziehungsweise Ausrichten eines Knotendiagramms mittels Stresswertberechnung möglich ist. Eigene Kriterien für die Stresswerterzeugung und Funktionen zum Rotieren und Verschieben von Komponenten ermöglichen es den Graphen eine organische Form von Knotendiagrammen annehmen zu lassen. Mit weiteren Kontrollpunkten und dem Anpassen des finalen Renderings, sind große, akkurate Bögen und eine eindeutige Darstellung der Überkreuzung von Kanten möglich.

Die durch den Algorithmus entstandenen Resultate in Kapitel 6 sind ähnlich zu anderen Visualisierungen von Knotendiagrammen. Für ein möglichst übereinstimmendes Aussehen mussten mehrere der implementierten Layoutoptionen für den Knoten individuell justiert werden. Wegen der hohen Anpassbarkeit der Parameter des Algorithmus sind potenziell viele Knotendiagramme erzeugbar.

Allerdings gibt es keine allgemeine Einstellung für das Erzeugen zufriedenstellender Knotendiagramme und es kann passieren, dass für bestimmte, nicht-getestete Knoten die Fähigkeit dieses Algorithmus nicht ausreicht. Die gewählten Verfahren und Initialisierung funktionieren für getestete Knoten, sind aber möglicherweise nicht die Besten. Insgesamt wurde für den Eclipse Layout Kernel des KIELER-Projektes eine funktionierender und anpassbarer neuer Layout-Algorithmus für mathematische Knoten hinzugefügt.

7.1. Zusammenfassung

Das Layout der Stressreduktion versucht Knotenpunkte anhand gewünschter Distanzen zu verbundenen Knotenpunkten zu positionieren. Sollten die Distanzen nicht eingehalten werden, wird ein Stresswert erzeugt. Der Stresswert wird verringert durch die Repositionierung der Knotenpunkte, bis ein möglichst kleiner Wert erreicht und somit ein gleichmäßig ausgelegter Graph entsteht.

Damit die organische Optik von Knotendiagrammen durch eine automatische Anordnung der Komponenten entstehen kann, musste dieser Ansatz angepasst werden.

7.1.1. Mathematische Knoten als Graph

Ein Knoten besteht aus mehreren Überkreuzungen und diese werden von den Knotenpunkten dargestellt. Die durchgängige, an ihren Enden verbundene Linie besteht aus mehreren Kanten, die an den Knotenpunkten nahtlos zusammenführen. Jeder Knotenpunkt hat somit immer vier Kanten, und ein Kontrollpunkt sorgt dafür, dass jeder dieser Kanten orthogonal in den Knotenpunkt führt. Um ein Knoten in ein Graph zu übersetzen, müssen entsprechend der Anzahl Überkreuzungen die Knotenpunkte definiert werden. Je nach Art des Knotens werden Kanten definiert und verbinden die Knotenpunkte. Das Stresslayout kann zur Initialisierung genutzt werden, da eine gleichmäßige Verteilung der Knotenpunkte eine gute Startvoraussetzung für Knoten ähnlich dem Trefoil ist.

7.1.2. Erfassen der Kriterien und Stresswerterzeugung

Die Kanten sollen in einem möglichst direkten Weg vom Startknotenpunkt zum Zielknotenpunkt führen. Dafür müssen die Winkel an den Kontrollpunkten der Kanten im besten Fall 180° haben. Der Stresswert durch Winkel wird erhöht, je kleiner der Winkel ist. Der Abstand zwischen verbundenen Knotenpunkten erzeugt ebenfalls einen Stresswert, der abhängig von der Distanz zum gewünschten Abstand ist.

Die aktuelle Verteilung der Komponenten bestimmt welcher Stresswert bei jedem Knotenpunkt individuell berechnet wird.

7.1.3. Implementierung der Stresswertreduktion

Um die Stresswerte zu verringern, gibt es zwei verschiedene Verfahren. In der Rotation werden die Kontrollpunkte um den Knotenpunkt herum rotiert, was die Ausrichtung von Kanten verbessert. Sollte ein Knotenpunkt trotz Rotation keine gute Ausrichtung finden, ist er an der falschen Position im Graphen und musste verschoben werden. Bei beiden Manövern wird sich der Stresswert vorher gemerkt und eine Bewegung in eine Richtung ausgeführt. Wenn es zu mehr Stress führt, wird ein größerer Schritt in die andere Richtung gemacht, wodurch alle Richtungen erreichbar sind, und den Knotenpunkt in die Richtung führt, wo der geringere Stresswert zu finden ist.

Die Verfahren der Rotation und Verschiebung werden in der Hauptiteration mehrmals hinter einander wiederholt.

7.1.4. Abschließende Verbesserungen

Da nach dem Stressreduktionsprozess die äußeren Bögen flach sind, werden weitere Kontrollpunkte in der Mitte der Kante hinzugefügt und verschoben. Es muss darauf geachtet werden, dass sie einen gleichmäßigen Abstand zu den Knotenpunkten haben, aber sich nicht zu weit distanzieren. Beim Rendern des Graphens durch **KlighD!** werden die Kontrollpunkte aufgeteilt, um mehrere Bézierkurven zu einer ganzen Kurve zu machen. Damit das nicht zu kaputten Kanten führt, braucht es noch zwei weitere Hilfskontrollpunkte, welche einen knickfreien Übergang der Bézierkurven gewährleistet.

Zusätzliche Einstellungen im Rendering erzeugt sowohl die Kurven, als auch die Lücken die essenziell zum Unterscheiden der überführenden und unterführenden Kanten einer Überkreuzung sind.

7.1.5. Anpassen von Parameter

Die resultierenden Knotendiagramme ähneln den herkömmlichen bekannten Versionen. Teilweise lassen sie sich nur zufriedenstellend erzeugen, wenn Parameter wie Abstände, Schrittweiten, Schwellwerte oder Gewichtung für den gewünschten Knoten angepasst werden. Die Implementierung hat viele dieser Parameter ergeben, die allesamt für den Nutzer über die Layoutoptionen veränderbar sind. Insgesamt ist so ein funktionierender und anpassbarer Layout-Algorithmus entstanden.

7.2. Zukünftige Arbeiten

Die Entwicklung dieses Layout-Algorithmus ist im Rahmen einer Bachelorarbeit entstanden, weshalb nicht alles in der Implementierung optimiert und viele Knoten nicht getestet werden konnten. So gibt es einige Aspekte die Inhalte für zukünftige Arbeiten bieten können.

7.2.1. Ausarbeitung des Algorithmus

Das implementierte Knotenlayout funktioniert und kann von mehreren Knoten erfolgreich Knotendiagramme erstellen. Dennoch gibt es Teile die weiter ausgearbeitet werden können. Auch wenn das Stresslayout als gutes Initiallayout für die Knotenpunkte fungiert, wurden keine anderen initialen Positionierungen getestet. Bei Knoten, welche von der Art her ähnlich dem Trefoil, Pentafoil oder auch dem Endlosknoten sind, hat das Stresslayout direkt die optimalen Positionen ergeben. Für

andere Knotenarten muss allerdings mit Verschiebungen gearbeitet werden. Wenn ein anderes Layout die Nutzung der stressreduzierenden Verschiebung minimiert, werden viele Berechnungen von Stresswerten wegfallen können.

Zurzeit läuft der Algorithmus über mehrere verschachtelte Schleifen, um das Layout zu generieren. Potenziell kann der Ablauf der Hauptiteration, sowie der Ablauf eines Stressminimierungsfunktion für einen Knotenpunkt, weiter optimiert werden, um die Performanz zu verbessern. Während der Verschiebungsfunktion werden Knotenpunkte abwechselnd auf den x- und y-Achsen vor oder zurückgeschoben. Hier wäre es auch denkbar, ab der zweiten Iteration, den Knotenpunkt auf beiden Achsen gleichzeitig zu bewegen. Dafür werden ab die Schrittweiten für die x- und y-Achse in der ersten Iteration ermittelt und sind abhängig davon welcher der Achsenrichtungen den Stresswert mehr reduzieren konnte. Das ermöglicht ein schnelles Erreichen eines minimalen Stresswertes, wodurch auch die Laufzeit verkürzt wird. Nach dem Stressreduktionsprozess erfolgt lediglich das Hinzufügen weiterer Kontrollpunkte für größere Bögen. An dieser Stelle können auch noch weitere Prozesse zum abschließenden Ausbessern des Knotens eingebaut werden.

Auch wenn die ungewollte Überschneidung von Kanten mithilfe der Parameter bislang umgangen werden konnte, kann nicht garantiert werden, dass keine Schnittpunkte entstehen. Es benötigt eine Funktion die sämtliche Schnittpunkte von Kanten überprüft und gegebenenfalls Kontrollpunkte setzt oder bestehende verschiebt, damit Kanten um andere Komponenten herumgeführt werden können. Da eine effektive Rotation oder Verschiebung von Knotenpunkten durch diese zusätzlichen Kontrollpunkte erschwert wird, sollte die Funktion wahrscheinlich ebenfalls erst nach dem Stressreduktionsprozess ausgeführt werden.

7.2.2. Testen von weiteren Knoten

Aufgrund der Zeitlimitation konnten nur eine handvoll Knoten mit diesem Layout-Algorithmus getestet werden. Die getesteten Knoten sind, unter anderen, die ersten Knoten auf einer Liste von Knotendiagrammen und gelten dadurch mit zu den bekanntesten Knoten. Allerdings entspricht dies nur einen sehr kleinen Bruchteil der mehreren Milliarden Knoten, die in der Knotentheorie bislang gefunden werden konnten. Besonders fehlen Versuche komplexere Knoten mit vielen Überkreuzungen zu erzeugen, um zu schauen ob, der Ansatz der Stressreduktion ebenfalls für große Knotendiagramme brauchbar ist.

Damit einige Knoten möglichst ähnlich zu bestehenden Visualisierungen aussehen, mussten an den einstellbaren Parametern Feinjustierungen vorgenommen werden. Die verwendeten Standardwerte der Parameter haben sich während der Entwicklung als größtenteils gut ergeben, sind aber für einige getestete Knoten nicht die optimalen.

Wenn in weiteren Untersuchungen gute Werte für die Layoutoptionen gefunden werden, um beispielsweise noch nicht geprüfte Knoten zu erhalten, kann darüber unter Umständen auch eine allgemeine Parametereinstellung für sämtliche Knoten ermittelt werden.

Des Weiteren fehlt noch der Spezialfall einen trivialen Unknoten in seiner einfachsten Form anzuzeigen. Dieser Knoten hat keine Überkreuzung und besteht somit nur aus einem einzelnen Ring. In diesem Fall dürften keine Knotenpunkte definiert sein, allerdings braucht es eine einzelne Kante. Eine Möglichkeit besteht darin den Unknoten standardmäßig dazustellen, sollte der Graphen über keine definierten Komponenten verfügen, aber das Knotenlayout als Algorithmus ausgewählt worden sein.

7.2.3. Umwandeln eines Knotens als planarer Graph

Im jetzigen Stand muss ein Knoten zuerst als Graph in einer ELKT oder kgt definiert werden, um den Layout-Algorithmus darauf anwenden zu können. Dieser Übersetzungsprozess wird aufwendiger, desto komplexer der Knoten ist. Es muss darauf geachtet werden, dass die definierten Kanten mit den richtigen Knotenpunkten verbunden sind, damit der gewünschte Knoten richtig ausgelegt werden kann. Manchmal müssen dafür die Definitionen der Komponenten unintuitiv getauscht werden, was teilweise nur mit ausprobieren möglich ist, ehe ein richtiger Knoten aus der Visualisierung erkannt werden kann. Und wenn ein Knoten, welcher in der ELKT definiert wurde, funktioniert, ist es nicht sicher, ob genau die gleichen Definitionen auch als kgt denselben Knoten ergibt. In den getesteten Fällen mussten einige Reihenfolgen von Komponenten getauscht werden.

In der Knotentheorie können Knotendiagramme umgeformt oder deformiert werden während sie immer noch den gleichen Knoten verbildlichen. Daher werden Knoten in der Mathematik auch einem Polynom zugeordnet, mit dem das Vergleichen von zwei Knoten erleichtert wird. Hierfür kann in einer zukünftigen Arbeit ein Übersetzer entwickelt werden, welcher von diesem eindeutigen Polynom einen entsprechenden Graphen erstellt, der dann von ELK verarbeitet werden kann. Ein ähnliches Verfahren wird in der Arbeit von Browne umgesetzt, bei der ein Knotendiagramm mithilfe einer Tutte-Einbettung generiert wird [Bro23]. Dort beschreibt ein Gausskode die Überkreuzungen eines Knotens und in welcher Reihenfolge diese besucht werden. Von diesem Gausskode aus werden dann zunächst die nötigen Knotenpunkte, Kanten und auch Facetten definiert, bevor ein Gittergraph des Knotens erzeugt wird.

Kapitel 8

Liste genutzter Abkürzungen

KIELER Kiel Integrated Environment for Layout Eclipse Rich Client

ELK Eclipse Layout Kernel

ELKT ELK-Textdatei

kgt kgt-Datei

KLighD KIELER Lightweight Diagams

IDE Integrated Development Environment

Anhang A

Anhang

A.1. GitHub

Die gesamte Implementierung des stresswertbasierten Knotenlayout-Algorithmus vom geforkten ELK-Projekt kann in meinem GitHub-Repository gefunden werden.
https://github.com/Stu224382/elk

Literatur

[AFL+22] Zoe Ashbridge, Stephen D. P. Fielden, David A. Leigh, Lucian Pirvu, Fredrik Schaufelberger und Liang Zhang. „Knotting matters: orderly molecular entanglements“. In: *Chem. Soc. Rev.* 51 (18 2022), S. 7779–7809. DOI: 10.1039/D2CS00323F. URL: http://dx.doi.org/10.1039/D2CS00323F.

[Bro23] Cameron Browne. „Drawing Knots with Tutte Embedding“. In: 2023. URL: https://archive.bridgesmathart.org/2023/bridges2023-149.pdf.

[DHS+23] Sören Domrös, Reinhard von Hanxleden, Miro Spönemann, Ulf Rüegg und Christoph Daniel Schulze. „The Eclipse Layout Kernel“. In: *arXiv preprint arXiv:2311.00533* (2023). URL: https://doi.org/10.48550/arXiv.2311.00533.

[Ead84] Peter Eades. „A Heuristic for Graph Drawing“. In: 1984. URL: https://api.semanticscholar.org/CorpusID:63936426.

[GKN04] Emden Gansner, Yehuda Koren und Stephen North. „Graph Drawing by Stress Majorization“. In: Bd. 3383. Sep. 2004. ISBN: 978-3-540-24528-5. DOI: 10.1007/978-3-540-31843-9_25.

[KK89] Tomihisa Kamada und Satoru Kawai. „An algorithm for drawing general undirected graphs“. In: *Information Processing Letters* 31.1 (1989), S. 7–15. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(89)90102-6. URL: https://www.sciencedirect.com/science/article/pii/0020019089901026.

[Liv93] Charles Livingston. *Knot Theory*. Carus Mathematical Monographs. Mathematical Association of America, 1993.

[LJ15] Nicole C H Lim und Sophie E. Jackson. „Molecular knots in biology and chemistry“. In: *Journal of Physics: Condensed Matter* 27 (2015). DOI: 10.1088/0953-8984/27/35/354101. URL: http://dx.doi.org/10.1088/0953-8984/27/35/354101.

[Rol03] D. Rolfsen. *Knots and Links*. AMS Chelsea Publishing Series. AMS Chelsea Pub., 2003. ISBN: 9780821834367. URL: https://books.google.de/books?id=s4eGEecSgHYC.

[Tai77] Tait. „VIII.-On Knots“. In: *Transactions of the Royal Society of Edinburgh* 28.1 (1877), S. 145–190. DOI: 10.1017/S0080456800090633.