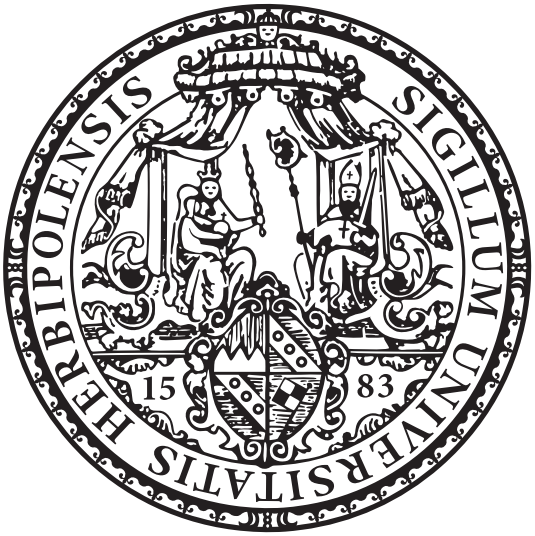
Bachelorarbeit

Komplexität des Zeichnens von Chord-Link Diagrammen

Jan Sauer

Abgabedatum:	18. Februar 2021
Betreuer:	Prof. Dr. Alexander Wolff
	Dr. Philipp Kindermann



Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen und Komplexität

Zusammenfassung

In dieser Arbeit wird die Komplexität des Zeichnens von Chord-Link Diagrammen, nach dem Modell von Angori et al. untersucht. Dabei wird zunächst die entsprechende Arbeit „Chord-Link: A New Hybrid Visualization Modell“zusammengefasst. Im Anschluss werden die beiden Hauptprobleme des Papers erneut aufgegriffen und durch Reduktionen bewiesen, dass beide NP-schwer sind.

Inhaltsverzeichnis

1 Einleitung

Graphen, die reale Beziehungen aus den verschiedensten Bereichen modellieren, weisen häufig lokale, kompakte Cluster auf, während die globale Struktur eine weitaus geringere Dichte vorweist. Zum Beispiel weisen biologische Zusammenhänge wie Protein-Protein-Interaktionen und der Stoffwechsel, solche Strukturen auf(Abbildung 1.1). Auch bei Graphen, die soziale Gefüge wie Zusammenarbeit oder Finanznetzwerke (Abbildung 1.2) erfassen, lassen sich diese Eigenschaften erkennen.

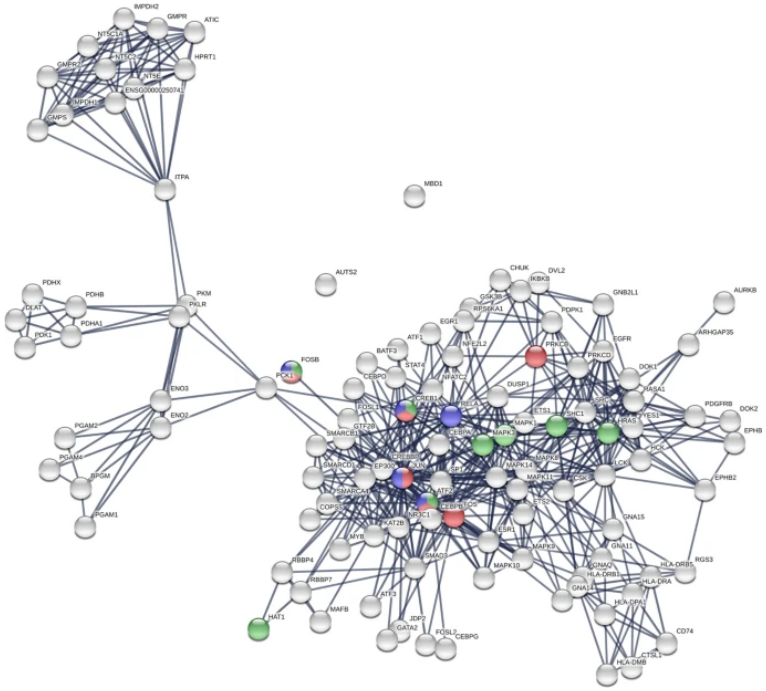


Abb. 1.1: Graph einer Protein-Protein-Interaktion [CLC+19]

Die durch diese Graphen modellierten Zusammenhänge sind häufig zu komplex um in einer üblichen Visualisierungsform, beispielsweise dem Node-Link Diagramm (Abschnitt 1.1.1), von einem Betrachter leicht gelesen und interpretiert werden zu können. Daher ist es sinnvoll, eine Darstellungsform zu wählen, beziehungsweise zu entwickeln, die die im Graphen vorhandenen Informationen übersichtlich darstellt und es dem Betrachter so ermöglicht einfach und intuitiv an die notwendigen Informationen zu gelangen.

Diese Problematik wurde bereits in diversen Arbeiten aufgegriffen. Ein grundsätzlicher Ansatz ist dabei die Verwendung eines hybriden Darstellungsmodels. In hybriden Darstellungsmodellen werden mehrere unterschiedliche Arten der Visualisierung kombi-

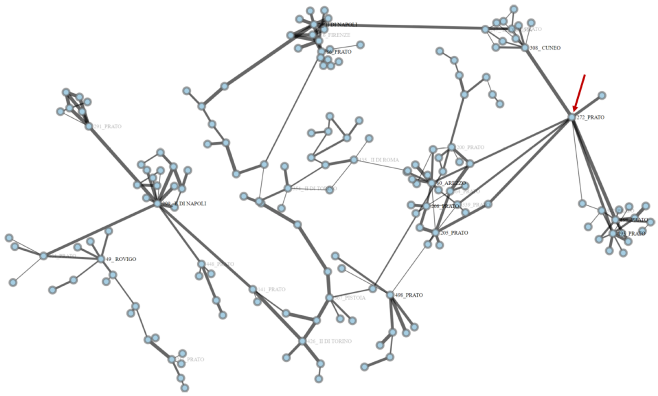


Abb. 1.2: Graph in der NodeLink Darstellung [ADM+19]

niert. Die Darstellung von Graphen mithilfe dieser Systeme wurde bereits vielfach thematisiert. Ein relativ einfacher, jedoch effizienter Ansatz ist dabei das *NodeTrix* Modell. Dabei werden ausgewählte Cluster mit Hilfe einer Ajdazentsmatrix dargestellt. Diese ist in ein Node-Link Diagramm eingebettet. Auf diese Weise werden die dichten Cluster übersichtlich als Matrix dargestellt, während der Gesamtzusammenhang weiterhin erkennbar ist. (Abbildung 1.3) [HRFM07]

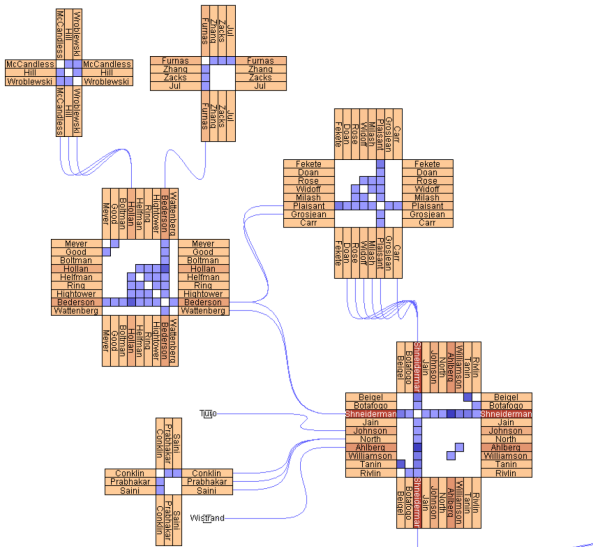


Abb. 1.3: Ausschnitt eines Graphen in Node-Trix Darstellung [HRFM07]

Ähnlich dem Node-Trix Ansatz verwendet auch das *TreeMatrix* Modell Matrizen. Dieses Modell beschäftigt sich zusätzlich zu der übersichtlicheren Darstellung von dichten Clustern auch mit deren Hierarchie. Dazu werden die Visualisierungsformen Node-Link, Adjazenzmatrizen und Bogendiagramme, sowie weitere Strukturen zur Darstellung der

Baumstruktur, verwendet. (Abbildung 1.4) [RMF12]

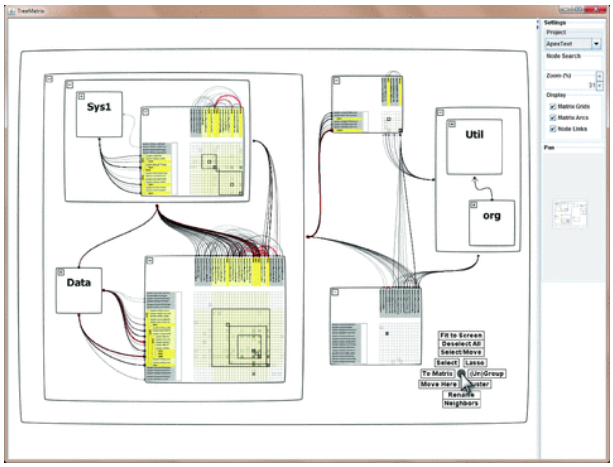


Abb. 1.4: Darstellung eines Graphen in TreeMatrix Darstellung [RMF12]

Sowohl NodeTrix als auch TreeMatrix visualisieren dabei zwar die Daten übersichtlich und mit der Reduktion von Kreuzungen. Auf den ersten Blick ist es jedoch wenig intuitiv die Informationen aus den Matrizen zu entnehmen. Die beiden Darstellungsformen helfen also zweifellos dabei, die Daten in Cluster einzuordnen und zu strukturieren. Eine intuitive, optisch ansprechende Visualisierung ist durch sie allerdings nicht gegeben.

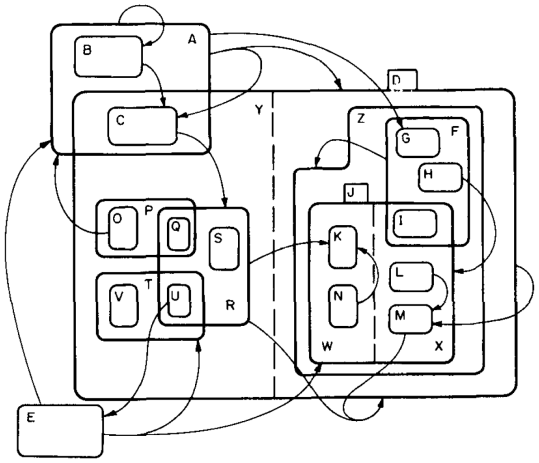


Abb. 1.5: Darstellung eines Graphen in HiGraph Darstellung [Har88]

Ein anderes hybrides System sind die so genannten *Higraphs*.(Abbildung 1.5) Zu Visualisierung werden hierbei Rechtecke verwendet, die Mengen an Knoten darstellen. Für jeden Knoten und für jede nicht leere Menge werden Rechtecke erstellt. Befindet sich ein Knoten in einer Menge, so ist er in deren Rechteck eingebettet. Zusätzlich wird die Möglichkeit zur Darstellung von kartesischen Produkten geboten. Dieses System aus Rechtecken wird außerdem mit konventionellen Kanten, das heißt Strecken zwischen

den Elementen kombiniert. Dadurch ergeben sich sehr effiziente Möglichkeiten, komplexe Graphen einfach und übersichtlich zu visualisieren, oder auch Arbeitsabläufe gut abzubilden. [Har88]

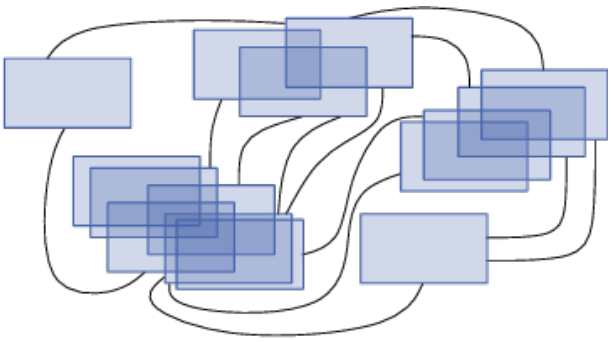


Abb. 1.6: Darstellung eines Graphen in Intersection-Link Darstellung mit 5 Cliquen [ADLB+17]

Ein weiteres System, dass sich Rechtecke zunutze macht, ist Intersection-Link.(Abbildung 1.6) Zur Zeichnung werden alle Knoten als Rechtecke dargestellt. Die Knotenmenge wird in möglichst wenige, möglichst große Cliquen partitioniert. Die Rechtecke aller, sich in der selben Clique befindlichen Knoten überlappen sich und visualisieren so die Kanten der Clique. Alle anderen Kanten werden wie in der Node-Link Darstellung durch Strecken zwischen den Rechtecken abgebildet.[ADLB+17]

Diese Beispiele sind nur ein Einblick in das bereits vielfältig bearbeitete Gebiet der hybriden Visualisierungsmodelle. Ebenso interessant wie die eigentlichen Modelle sind hierbei die Anwendungsmöglichkeiten und Algorithmen, die aus den gegebenen Graphendaten eben jene Abbildung erstellen. Dabei spielt die Komplexität der Algorithmen eine wichtige Rolle für die Einsatzgebiete des Modells. Eine lange Laufzeit eines Algorithmus ist dabei grundsätzlich negativ zu betrachten. Ein interaktives System beispielsweise, bei dem ein Benutzer aus bestehenden Daten nach Belieben eine Abbildung aufbaut benötigt eine möglichst kurze Berechnungsdauer, um dem Nutzer ein angenehmes Arbeiten zu ermöglichen.

Ein eben solches interaktives System wurde auch im Rahmen des Chord-Link Modells entwickelt. Chord-Link ist dabei ein hybrides Darstellungsmodell, das auf der Kombination von Node-Link (Abschnitt 1.1.1) mit Chord-Diagrammen (Abschnitt 1.1.2) beruht. Ähnlich des Node-Trix Modells werden die dort als Adjazentsmatrix abgebildeten Daten eines Clusters in Chord-Diagrammen dargestellt. Diese Chord-Diagramme bestehen aus einem Kreis auf dessen Rand Knoten, beziehungsweise deren Kopien, positioniert sind, die im Kreisinneren mit Sehnen verbunden werden. Der Benutzer erhält einen Graphen in Node-Link Form und kann aus diesem, durch die Auswahl von Knoten, nach seinen Wünschen Chord-Graphen erstellen.

Die besonderen Hindernisse bei dieser Darstellungsform liegen in der Erstellung der Chord-Diagramme. Zur Erzeugung dieser, werden Kopien von Knoten, eines ausgewählten Clusters, zunächst auf einem Kreis verteilt. Der erste Algorithmus (Abschnitt 2.3) ordnet die Kopien in einer möglichst günstigen Reihenfolge. Dabei werden Kopien des

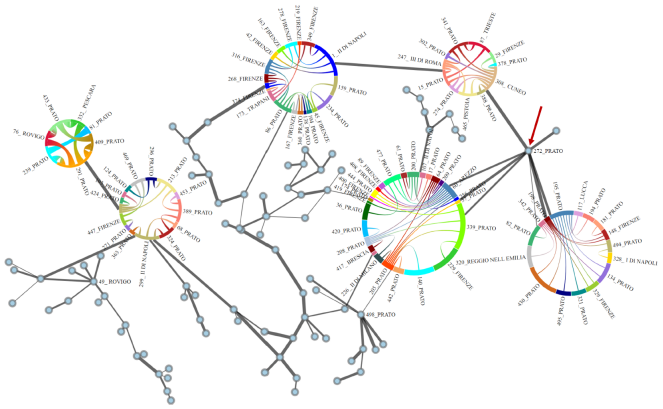


Abb. 1.7: Graph aus Abbildung 1.2 in der ChordLink Darstellung [ADM+19]

gleichen Knotens möglichst nebeneinander platziert. Ein weiterer Algorithmus fügt in einem darauffolgenden Schritt Sehnen in das Diagramm ein, die die Kanten repräsentieren (Abschnitt 2.5). Dabei wird versucht eine möglichst geringe Anzahl an Kreuzungen zu erzeugen. Die genaue Vorgehensweise wird in Kapitel 2 erläutert.[ADM+19]

Aufgrund des interaktiven Systems, sowie dem eigentlichen Hindernis des Erstellens, ist in diesem Fall also die Effizienz um diese Graphen zu zeichnen umso wichtiger. Diese wird als Thema in dieser Arbeit aufgegriffen und offene Fragestellungen nach der Komplexität der Probleme gelöst.

1.1 Grundlagen

1.1.1 Node-Link

In dieser Darstellungsform sind die Knoten meist als Punkte, Kreise, Polygone oder Text dargestellt. Die Kanten werden durch Strecken oder Kurven zwischen den zwei Objekten ihrer Knoten visualisiert. Graphen werden oft standardmäßig als Node-Link Diagramm dargestellt, da sie grundsätzlich sehr intuitiv und einfach zu lesen sind. (Abbildung 1.2) Nachteil dieser Art von Zeichnungen ist jedoch, dass sie bei großen oder komplexen Graphen schnell unübersichtlich werden. Im Fall des im Chord-Link Modell verwendeten Algorithmus wird ein Node-Link Diagramm eines Graphen als Ausgangslage verwendet, um aus bestimmten Clustern Chord-Link Diagramme zu erstellen. (Abbildung 1.7)

1.1.2 Chord-Diagramm

Als Chord-Diagramm oder auch Chord-Graph wird in dieser Arbeit eine besondere Form der Chord-Diagramme bezeichnet. Diese wurden von Angori et al. in „ChordLink: A New Hybrid Visualization Model“[ADM+19] eingeführt. Da sich die Beweise dieser Arbeit auf die dort vorgestellten Probleme beziehen, müssen entsprechend auch die selbe Art von Diagrammen verwendet werden. Generell versteht man unter einem Chord-Diagramm

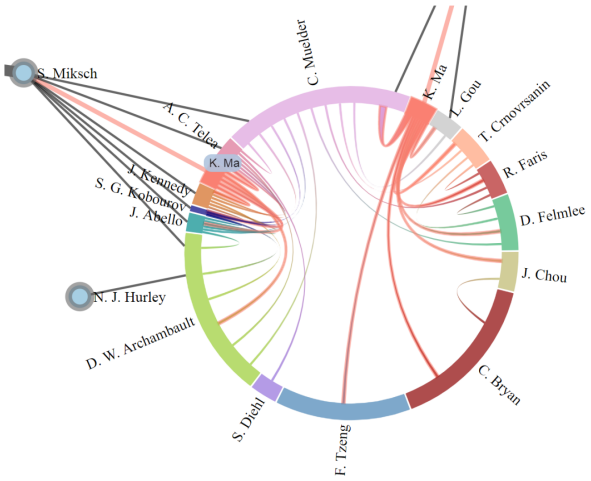


Abb. 1.8: Graph in der Chord Darstellung mit angrenzenden Knoten [ADM+19]

einen geometrischen Kreis, auf dessen Rand die $|V|$ Knoten des Graphen $G = (V, E)$ verteilt sind. Diese Knoten werden mit Sehnen im Inneren des Kreises miteinander verbunden [Wes87]. Die genauen Definitionen, so wie Arten der Verbindung und Darstellung der Knoten variieren dabei stark. Die hier verwendeten Chord-Diagramme bestehen aus einem Kreis R. Auf dem Rand dieses Kreises R werden die Kopien der Knoten V als Teilkreisstücke, ohne Überlappung platziert. Dabei existiert für jeden Knoten $v \in V$ mindestens eine Kopie und ein Teilkreisstück. Die Kanten E werden als Sehnen oder auch Chords visualisiert. Diese Sehnen sind Kurven innerhalb des Kreises, die zwei entsprechende Kreisstücke miteinander verbinden. Sollten von einen Knoten mehrere Teilstücke existieren, so kann ein beliebiges zum Einfügen der Sehne verwendet werden. Mathematisch kann der Kreis der Chord-Diagramme auch als zirkuläre Liste an Kopien betrachtet werden. Dabei wird ein beliebiger Punkt gewählt. Von dort aus geht man den Kreis im Uhrzeigersinn ab bis man am Ursprung ankommt und erhält so eine Reihenfolge an Kopien. Diese Reihenfolge als zirkuläre Liste ist eine mathematische Darstellung des visualisierten Kreises.

Diese Betrachtungsweise wird in späteren Beweisen aufgegriffen.

1.2 Eigener Beitrag

Zunächst werden in Kapitel 2 die Algorithmen und Ergebnisse der Arbeit über das Chord-Link System zusammengefasst.

Ein offenes Problem, das bei der Vorstellung des Chord-Link Modells erwähnt wird, ist der Beweis über die Komplexität der Probleme der perfekten Permutation, sowie des Sehnen Einfügens. Die dort verwendeten Algorithmen, welche in Abschnitt 2.3 und Abschnitt 2.5 beschrieben werden, dienen lediglich als Heuristiken und liefern nur für besondere Fälle eine exakte Lösung.

Der erste betrachtete Algorithmus versucht die Kopien, die sich auf dem Kreis eines

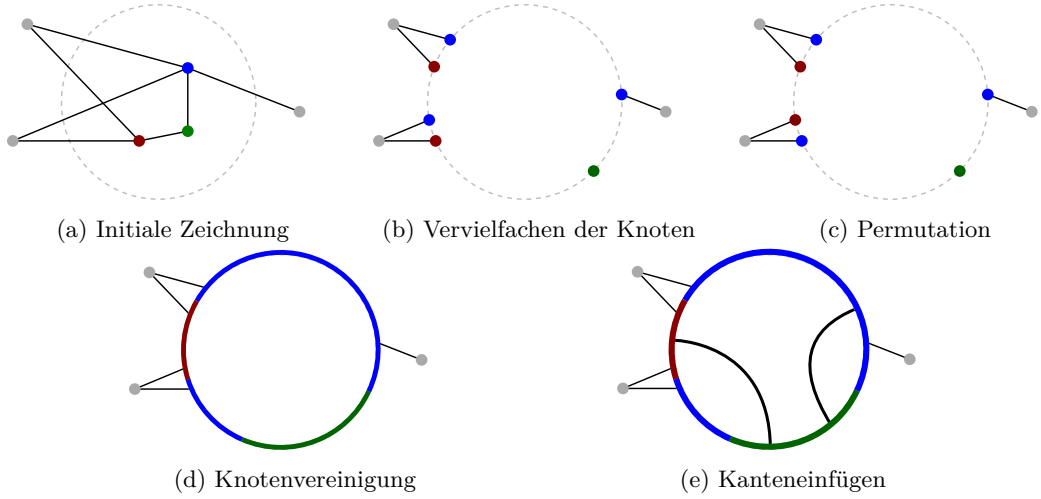
Chord-Diagramms befinden, so zu arrangieren, dass Kopien des gleichen Knotens möglichst nebeneinander liegen. Dabei dürfen nur Knoten miteinander vertauscht werden, welche zum selben Knoten außerhalb des Diagramms adjazent sind. Dieses Problem wird als Problem der perfekten Permutation bezeichnet. In Kapitel 3 wird mittels Reduktion von *Set-Cover* gezeigt, dass dieses Arrangieren nur für Ausnahmefälle effizient möglich ist. Die Reduktion bildet dabei eine Set-Cover Instanz als Reihenfolge von Knoten auf einem Chord-Graphen ab. Findet man eine optimale Lösung für die Permutation des entstandenen Chord-Graphen, so gibt es eine intuitive Lösung für die entsprechende Set-Cover Instanz.

Anschließend wird in Kapitel 4 ein Algorithmus für einen Spezialfall des Permutationsproblems (Abschnitt 2.3) betrachtet, der für diesen eine exakte Lösung liefert. Bei diesem Fall handelt es sich um eine starke Einschränkung, die entsteht wenn die zu tauschenden Kopien lediglich mit zwei Knoten außerhalb des Chord-Diagramms verbunden sind. Weiter wird in Kapitel 5 eine Reduktion von *SAT* erläutert. Diese beweist, dass das in Abschnitt 2.5 beschriebene Problem des Einfügens der Sehnen NP-schwer ist. Dabei sollen die Sehnen möglichst ohne Kreuzung eingefügt werden. Es wird dabei, von einer SAT Instanz ausgehend, ein Chord-Graph erzeugt. Sollten alle Sehnen ohne Kreuzung eingefügt werden können, so ist die SAT Instanz erfüllbar. Dabei ist zu beachten, dass jede Kante mehrere Optionen haben kann. Des Weiteren ist durch diesen Beweis nicht nur gezeigt, dass das Problem NP-schwer, sondern durch eine Übertragbarkeit auf *Max-SAT* auch nicht beliebig gut approximierbar ist. Durch Max-SAT kann außerdem gezeigt werden, dass das Problem auch mit starken Einschränkungen immer noch NP-schwer ist.

2 Zusammenfassung ChordLink: A New Hybrid Visualization Model

Diese hier erwähnten bisherigen Ergebnisse beziehen sich auf die Arbeit *ChordLink: A New Hybrid Visualization Model* [ADM+19] von Angori et al., welches ein interaktives System vorstellt und sich mit der Transformation eines Graphen in Node-Link Darstellung in die eigens definierte Chord-Link Darstellung beschäftigt. Im Folgenden werden nun die notwendigen Grundlagen zur Verständnis des Papers erläutert, sowie die Herangehensweise und Ergebnisse zusammengefasst.

2.1 Übersicht



(a) Initiale Zeichnung (b) Vervielfachen der Knoten (c) Permutation

(d) Knotenvereinigung (e) Kanteneinfügen

Abb. 2.1: Grundlegendes Vorgehen des Algorithmus zur Bildung einer Chord-Darstellung aus einer Node-Link-Darstellung.

Das im Paper beschriebene System beschäftigt sich mit der Konstruktion von Chord-Diagrammen aus Teilgraphen eines in Node-Link Darstellung visualisierten Graphen. Der beschriebene Algorithmus kommt in diesem interaktiven System zur Verwendung, bei dem der Benutzer zu Beginn initial eine Node-Link Darstellung eines Graphen $G = (V, E)$ gegeben hat. Der Nutzer wählt eine Kreisregion K im Node-Link Diagramm. M ist die Menge der Knoten, die in der gewählten Region K liegen. Aus dieser Teilmenge M erstellt der Algorithmus ein, in das Node-Link Diagramm eingebettetes,

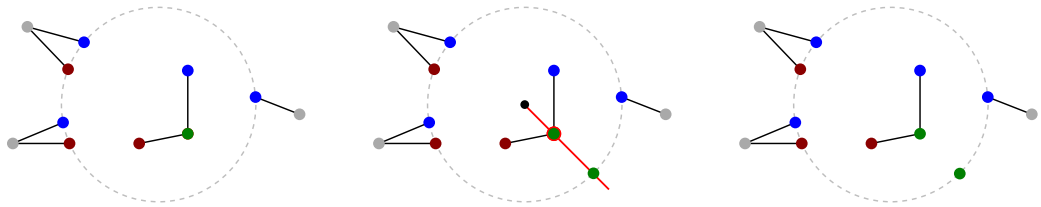
Chord-Diagramm. Der Algorithmus gliedert sich dabei in vier Teile:

1. Vervielfachen der Knoten der Menge M auf K (Abbildung 2.1b)
2. Permutieren der entstandenen Instanzen auf K (Abbildung 2.1c)
3. Vereinigung von Kopien des gleichen Knotens zu Kreisteilstücken (Abbildung 2.1d)
4. Einfügen der Kanten $(u, v) \in E$, mit $u.v \in M$ als Sehnen (Abbildung 2.1e)

Diese werden in den kommenden Abschnitten beschrieben. Dabei wird festgestellt, dass die Permutation nur für Spezialfälle optimal in $O(|E|^3)$ gelöst wird. Für das Einfügen wird lediglich eine Heuristik angegeben.

Als Grundbedingung ist außerdem festgelegt, dass die Struktur außerhalb des entstehenden Chord Diagramms nicht verändert werden darf. Das bedeutet, dass Knoten die nicht in M, beziehungsweise Kanten die keinen Punkt in M haben, nicht verändert werden dürfen. Kanten, die lediglich einen Knoten in M haben, dürfen außerdem nicht verschoben oder gedreht werden und enden immer beim Schnittpunkt mit dem Rand von K. Es darf jedoch die Kopie verändert werden, bei der die Kante endet, was später beim Permutieren zum Tragen kommt. Diese Einschränkungen sollen helfen, sich nach dem Erstellen des Chord-Diagramms nicht neu orientieren zu müssen.

2.2 Vervielfachen



(a) Initiale Zeichnung

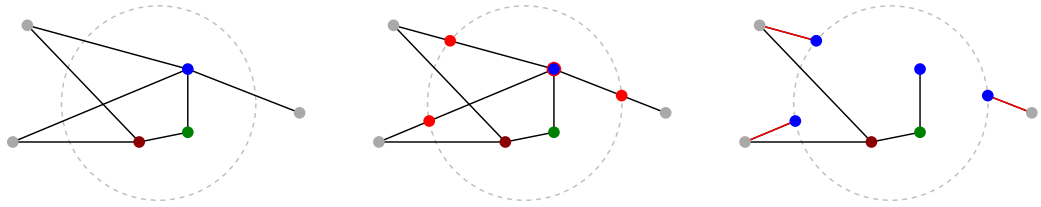
(b) Erzeugen eines Strahls vom Mittelpunk durch den Knoten

(c) Vervielfachen des Knotens am Schnittpunkt des Strahls mit dem Rand von K

Abb. 2.2: Vervielfachen eines nach innen gerichteten Knotens.

Beim Vervielfachen werden die Knoten der Teilmenge M auf den Rand der Kreisregion K verschoben. Dabei wird zwischen nach *außen gerichteten* Knoten und nach *innen gerichteten* Knoten unterschieden. Nach außen gerichtete Knoten sind alle Knoten $v \in M$, so dass eine Kante $(v, u) \in E$ existiert, mit $u \notin M$. Die Menge der nach innen gerichteten Knoten ist definiert als alle Knoten $v \in M$, so dass für alle Kanten $(v, u) \in E$ gilt, dass $u \in M$. Es ist also festzustellen, dass Knoten, welche eine Kante zu einem Knoten außerhalb der Kreisregion haben, anders gehandhabt werden.

Zunächst wird das Vervielfältigen für nach innen gerichtete Knoten betrachtet. Hierzu wird eine gerade Linie ausgehend vom Mittelpunk der Kreisregion R durch den zu



(a) Initiale Zeichnung

(b) Vervielfachen der Knoten am Schnittpunkt jeder Kante mit dem Kreis

(c) Ersetzen der Kanten durch Teilstücke

Abb. 2.3: Vervielfachen eines nach außen gerichteten Knotens.

vervielfältigenden Knoten v gelegt, welche den Rand der Kreisregion schneidet. An diesem Schnittpunkt der Linie und dem Rand von K wird eine Kopie v_i des Knotens v platziert.(Abbildung 2.2) Dieses Vorgehen wird für alle nach innen gerichteten Knoten wiederholt.

Beim Vervielfältigen der nach außen gerichteten Knoten können mehrere Kopien entstehen. Für jeden nach außen gerichteten Knoten v werden alle Kanten $(v, u) \in E$ betrachtet, bei denen $u \notin M$. Per Definition folgt, dass diese Kanten einen Schnittpunkt mit dem Rand von K haben müssen. An jedem dieser n Schnittpunkte wird eine Kopie v_i von v erstellt wobei $i \in \{1, \ldots, n\}$. Abbildung 2.3

Im letzten Schritt werden alle ursprünglichen Knoten aus M aus der Darstellung entfernt, so dass lediglich die Kopien auf dem Rand von K erhalten bleiben.

2.3 Permutationsalgorithmus

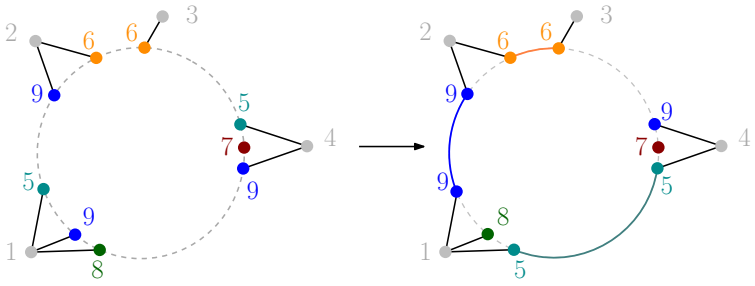


Abb. 2.4: Maximierung der direkt aufeinanderfolgenden Kopien desselben Knotens auf K durch Vertauschung

In diesem Schritt wird die Reihenfolge der bei der Vervielfältigung erzeugten Kopien verändert. Im Paper wurde festgestellt, dass es zu einer besseren Übersichtlichkeit führt, wenn Kopien des gleichen Knotens nebeneinander liegen, und so praktisch als ein Knoten betrachtet werden können. Hierzu soll der Algorithmus eine Permutation der Kopien finden, bei der möglichst viele Kopien der gleichen Knoten direkt nebeneinander liegen. Aufgrund der zu Beginn getroffenen Einschränkungen, die die Übersichtlichkeit erhalten

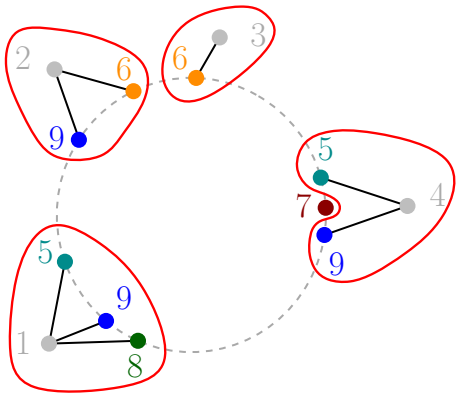


Abb. 2.5: Alle Gruppen der adjazenten Knoten v_i

soll, ist die einzige Operation, die dabei verwendet werden kann, das Vertauschen zweier Kopien u_i, w_j mit $u, w \in M$ auf K. Zusätzlich muss für u_i und w_j gelten, dass beide adjazent zum gleichen Knoten $v \notin M$ sind. Für Knoten ohne Verbindung nach außen wurde im Paper keine Option zur Verschiebung erläutert.

Zunächst wird für jeden Knoten v_i für den gilt, dass

1. $v_i \notin M$
2. Es existiert eine Kante $(v_i, u) \in E$ mit $u \in M$

eine Sequenz $\langle u_{i,1}, u_{i,2}, \ldots, u_{i,k} \rangle$ an Kopien nach außen gerichteter Knoten abgelesen. Man erhält diese Sequenz indem man die Kopien auf dem Rand von K im Uhrzeigersinn abgeht und alle Kopien, die adjazent zu v_i sind, hinzufügt. Diese Sequenzen sind also eine Reihenfolgen an Kopien, die zum gleichen Knoten außerhalb von K adjazent sind. Eine solche Sequenz wird *Gruppe von* v_i genannt (Abbildung 2.5).

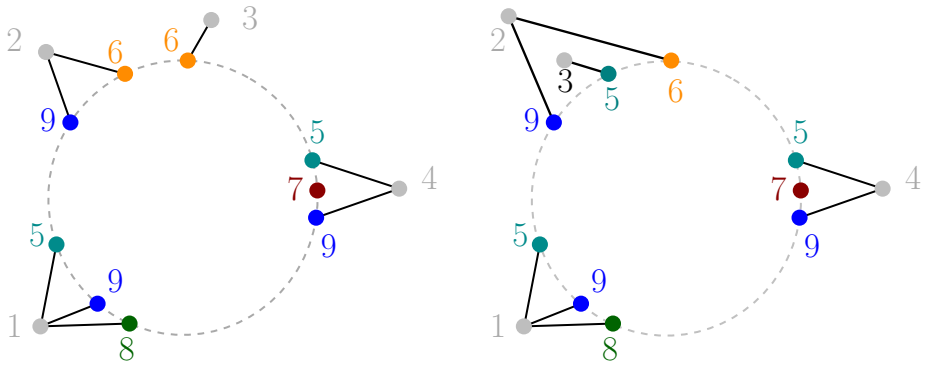
Die erlaubte Operation kann nun als das Tauschen von zwei Kopien u, w der Gruppe von v_i beschrieben werden.

Im folgenden sei L eine zirkuläre Liste welche die Kopien auf dem Rand von K im Uhrzeigersinn enthält. Dabei kann bei einer beliebigen Kopie begonnen werden. Für den vorgestellten Algorithmus wird davon ausgegangen, dass alle Kopien jeder Gruppe in L direkt aufeinanderfolgen (Abbildung 2.6). Folglich wurde im Paper lediglich ein Beweis für diesen Fall gezeigt, welcher hier vorgestellt wird. Für diesen Spezialfall liefert der Algorithmus ein exaktes Ergebnis. Ansonsten dient er lediglich als Heuristik. Die Güte der Heuristik wurde nicht weiter behandelt.

Um das Optimierungsproblem aufzustellen, wird die Kostenfunktion $\chi(u, n(u))$ aufgestellt. Dabei ist $u \in L$ die Kopie eines Knotens v und $n(u) \in L$ die nächste Kopie von v in L. (Abb. 2.7)

Für die Kostenfunktion gilt:

$$\chi(u,\ n(u)) = \begin{cases} 0 & \text{keine andere Kopie zwischen } u \text{ und } n(u) \\ 1 & \text{sonst} \end{cases}$$



(a) Gruppen sortiert (b) Gruppen unsortiert

Abb. 2.6: Gruppen sortiert (links) und unsortiert (rechts)

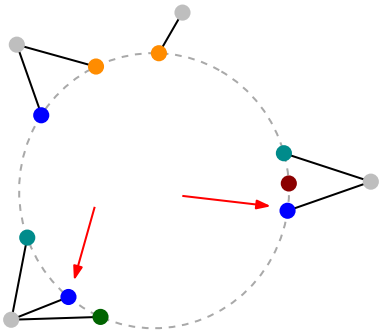


Abb. 2.7: Linker Pfeil: Kopie u, Rechter Pfeil: $n(u)$

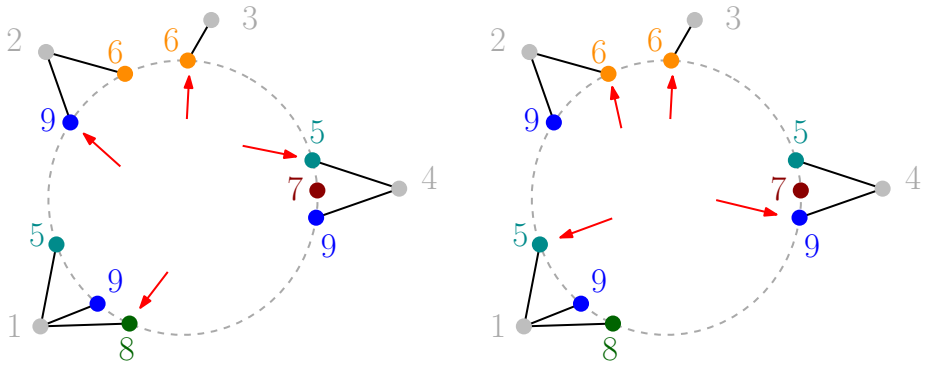
Als $B_1, B_2, \ldots, B_h$ wird anschließend die Reihenfolge der Gruppen in L im Uhrzeigersinn definiert. Da davon ausgegangen wird, dass alle Kopien jeder Gruppe direkt aufeinanderfolgend in L sind, muss es für jede Gruppe ein erstes und ein letztes Element geben. Sei e_i das erste und l_i das letzte Element der Gruppe von v_i (Abbildung 2.8b). Dabei kann man die Feststellung machen, dass l_i und e_{i+1} immer aufeinanderfolgend sind.

Ausgehend hiervon wird mit Hilfe der Kostenfunktion $\chi(u, n(u))$ das Optimierungsproblem wie folgt definiert:

Finde Permutationen der Kopien innerhalb der Gruppen B_i mit $i \in \{1, \ldots, h\}$, so dass $\sum_{u \in L} \chi(u, n(u))$ minimal ist.

Betrachtet man dies nun genauer, so kann man feststellen, dass die Kosten einer Permutation lediglich vom ersten und letzten Element abhängen, da für alle anderen Kopien einer Gruppe die Kostenfunktion immer 1 beträgt.

Um die Gesamtkosten rekursiv zu berechnen, wird ein dynamisches Programm genutzt dass $O_i(u_{i,j}, u_{i,z})$ berechnet. Dabei ist $O_i(u_{i,j}, u_{i,z})$ die Kostenfunktion, unter der Annahme, dass $e_i = u_{i,j}$ und $l_i = u_{i,z}$, wobei $u_{i,k}$ das k-te Element der Gruppe B_i ist.



(a) Erste Elemente der Gruppen (b) Letzte Elemente der Gruppen

Abb. 2.8: Erste und letzte Elemente aller Gruppen

Für jedes mögliche Paar $u_{i,j}, u_{i,z}$ in B_i gilt:

$$O_i(u_{i,j}, u_{i,z}) = O_{i+1}(u_{i+1,j'}, u_{i+1,z'}) + \begin{cases} 0 & \text{falls } u_{i+1,j'} = u_{i,z} \\ 1 & \text{sonst} \end{cases}$$

Die optimale Lösung ist dann gegeben, wenn die Kostenfunktion minimal ist, das heißt

$$\chi_{opt} = min_{u_{1,j}, u_{1,z} \in B_1} O_1(u_{1,j}, u_{1,z}).$$

Diese Rekursion wird gelöst, indem man ein beliebiges e_1 fix setzt. Dadurch stoppt das Programm, sobald der Kreis abgegangen wurde. Für e_1 kann somit eine Tabelle berechnet werden, die alle möglichen Permutationen mit e_1 als erstes Element mit den dazugehörigen Kosten enthält. Die Laufzeit der Berechnung der Tabelle beträgt dabei:

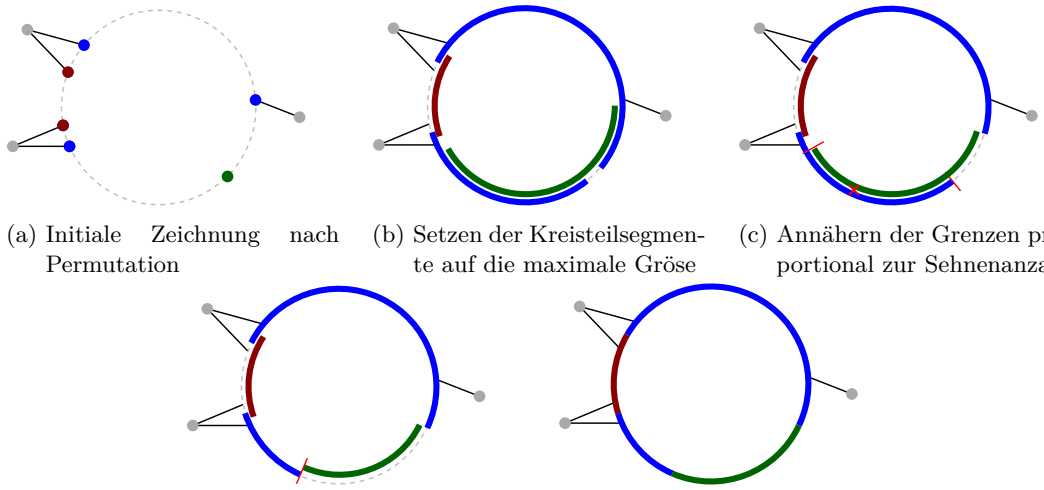
$$\sum_{i=1}^{h} = \binom{k_i}{2} \leq m^2$$

Es gilt dabei:

1. $m = |E|$
2. $h =$ Anzahl der Gruppen
3. $k_i =$ Anzahl der Kopien in Gruppe i

Im Folgenden muss für jedes mögliche e_1 die entsprechende Tabelle berechnet werden. Aus allen Tabellen kann dann das minimale Ergebnis als optimale Lösung verwendet werden. Aufgrund der Berechnung jeder einzelnen Tabelle, mit der oben beschriebenen Laufzeit, ergibt sich für den kompletten Algorithmus eine Laufzeit von $O(m^3)$, da es maximal $k_1 < m$ Kopien in jeder Gruppe geben kann.

Wie bereits erwähnt findet dieser Algorithmus nur für einen Spezialfall eine exakte Lösung. Der Beweis, dass dieses Problem in der Tat NP-schwer ist, ist in der Arbeit von Angori et al. als mögliches zukünftiges Thema erwähnt und wird in Kapitel 3 bearbeitet.



(a) Initiale Zeichnung nach Permutation

(b) Setzen der Kreisteilsegmente auf die maximale Gröse

(c) Annähern der Grenzen proportional zur Sehnenanzahl

(d) Anpassen der Kreisstücke

(e) Einfügen auf K

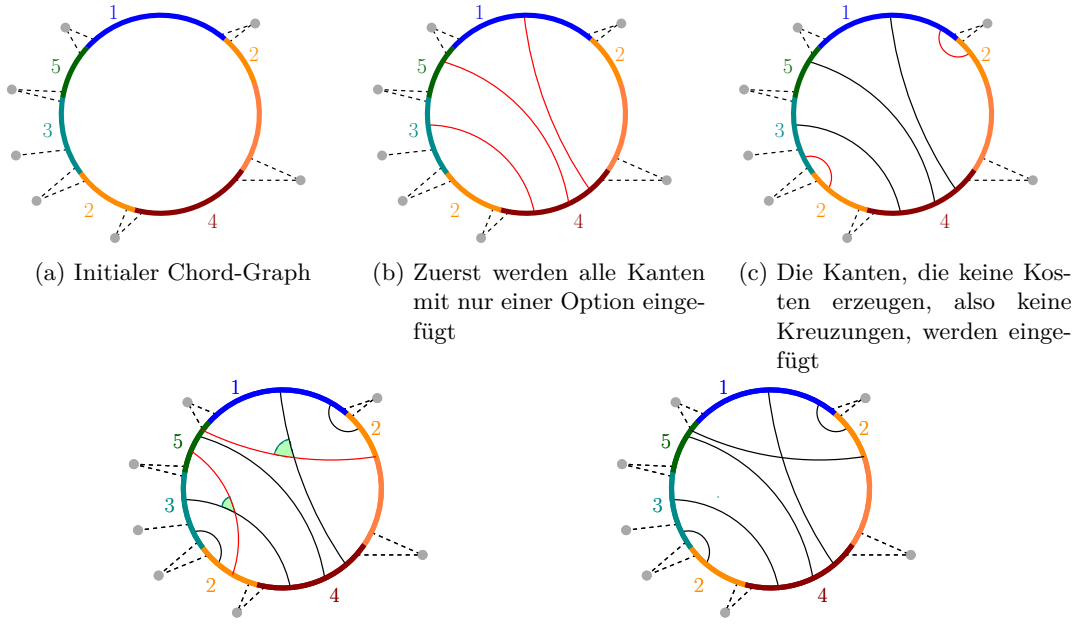
Abb. 2.9: Vereinigung der Knoten. Im Beispiel ist die Sehnenanzahl für rot und blau eins und für grün zwei

2.4 Vereinigung der Kreisstücke

In dieser Phase werden die Kopien, die sich auf dem Rand von K befinden in Kreisteilstücke, im Paper auch *Arcs* genannt, umgewandelt. Dabei werden Sequenzen, also mehrere direkt aufeinanderfolgende Kopien des selben Knotens w, zu einem Teilstück zusammengefasst. Dieses Teilstück wird C_w genannt. Da die externen Kanten nicht verschoben werden können, muss das Teilstück mindestens alle Kopien der Sequenz überspannen. Um entstehende Lücken zwischen zwei auf K benachbarten Arcs C_w und C_z zu überdecken werden die beiden benachbarten Endpunkte der Teilstücke schrittweise angenähert bis sie sich treffen und damit die Lücke schließen. Im späteren Verlauf wird der dadurch überspannte Bereich noch proportional zu den mit den Teilstücken verbundenen Sehnen aufgeteilt. (Abbildung 2.9)

2.5 Algorithmus zum Einfügen der Sehnen

Wie in mehreren Arbeiten, welche auch im Paper angegeben waren, gezeigt wurde, verschlechtert sich die Lesbarkeit einer Darstellung von Graphen mit einer zunehmenden Anzahl an Kreuzungen zwischen den Kanten, bzw. mit kleinen Kreuzungswinkeln zwischen den Kanten. Diese Erkenntnisse erhalten im letzten Schritt des Erstellens eines Chord-Diagramms Einzug. Im letzten Teil werden die zuvor gelöschten Kanten zwischen den Knoten aus M als Sehnen wieder eingefügt. Da für jeden Knoten $v \in M$ mehrere repräsentierende Arcs existieren können, wird mithilfe dieses Algorithmus eine Kombination gewählt, mit der die Anzahl an Sehnenkreuzungen minimiert wird. Sollte es keine Möglichkeit geben die Kanten ohne Kreuzungen einzufügen, so wird versucht den



(a) Initialer Chord-Graph

(b) Zuerst werden alle Kanten mit nur einer Option eingefügt

(c) Die Kanten, die keine Kosten erzeugen, also keine Kreuzungen, werden eingefügt

(d) Es wird die Kante mit dem größeren Kreuzungswinkel eingefügt.

(e) Alle Sehnen platziert

Abb. 2.10: Graph mit den Knoten $[1, 2, 3, 4, 5]$ und den Kanten $[(1, 2), (2, 3), (2, 5), (3, 4), (1, 4), (4, 5)$. Die zirkuäre Liste ist $[1, 2, 4, 2, 3, 5]$

Kreuzungswinkel zu maximieren. Der im Paper beschriebene und hier zusammengefasste Algorithmus dient hierbei lediglich als Heuristik und liefert keine exakte Lösung.

Das gegebene Problem wird in ein Optimierungsproblem umgewandelt. Zunächst wird dafür eine Kostenfunktion α aufgestellt. Seien e_{wz} und e_{xy} zwei Sehnen, die die Kanten (w, z), bzw. (x, y) repräsentieren. Weiter wird als $a(\overline{wz}, \overline{xy})$ der kleinste Winkel bezeichnet, der bei der Kreuzung der beiden Sehnen e_{wz} und e_{xy} entsteht. Da der Kreuzungswinkel a immer der kleinste ist, ist $a(\overline{wz}, \overline{xy}) \in]0; \pi/2]$. Die Kostenfunktion ist wie folgt definiert:

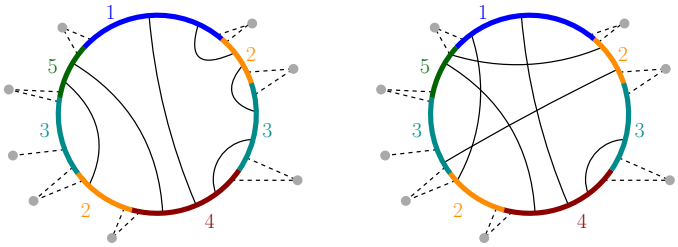
$$\alpha(e_{wz}, e_{xy}) = \begin{cases} 0 & e_{wz} \text{ und } e_{xy} \text{ kreuzen sich nicht} \\ 1 - a(\overline{wz}, \overline{xy}) \div \pi & \text{sonst} \end{cases}$$

Dadurch gilt im Falle einer Kreuzung, dass $\alpha(e_{wz}, e_{xy}) \in [0,5; 1[$.

Das Optimierungsproblem gestaltet sich folgendermaßen. Es wird eine Sehnenmenge S gesucht, die die Kanten zwischen den Knoten aus M abdeckt. Dabei muss

$$\alpha(S) = \sum_{\{e_{wz}, e_{xy}\} \in S \times S} \alpha(e_{wz}, e_{xy})$$

minimal sein.



(a) Einfügen mit möglichst wenig Kreuzungen

(b) Gleicher Graph mit mehr Kreuzungen

Abb. 2.11: Gutes (a) und schlechtes (b) Beispiel zum Platzieren der Sehnen

Um dieses Problem zu lösen, geht der Algorithmus nach einer Greedy-Strategie vor. Dafür wird die Menge der einzufügenden Kanten E_M in zwei Teilmengen E_{M1} und E_{M2} aufgeteilt. In E_{M1} befinden sich dabei alle Kanten, für die es lediglich eine Möglichkeit gibt, sie einzufügen. Das heißt, für beide Knoten dieser Kanten existiert nur ein repräsentierender Arc. Folglich befinden sich in E_{M2} alle Kanten, welche über mehrere potentielle Sehnen verfügen. Der Algorithmus beginnt, indem er alle Sehnen der Kanten aus E_{M1} in beliebiger Reihenfolge einfügt. Da für diese Sehnen nur eine Möglichkeit besteht ist dieses Vorgehen sehr einfach. Im Anschluss durchläuft der Algorithmus $|E_{M2}|$ Iterationen. Jede Iteration i mit $(1 \leq i \leq |E_{M2}|)$ entfernt dabei eine Kante aus E_{M2} und fügt eine die Kante repräsentierende Sehne in die Zeichnung ein. Zudem wird S_0 als die Menge an Sehnen definiert, die nach Einfügen der Sehnen für E_{M1} in der Zeichnung vorhanden sind. Weiter ist S_i die Menge an Sehnen, die nach der i-ten Iteration eingefügt wurden. Um auszuwählen welche Kante gelöscht wird, werden zu Beginn jeder Iteration für jede Sehne c jeder Kante aus $E(M)$ die Kosten berechnet, sollte c in die aktuelle Zeichnung eingefügt werden. Diese Kosten lassen sich also darstellen, als $\alpha(S_{i-1} \cup \{c\})$. Im Anschluss wird die Sehne mit den geringsten Kosten gewählt, in die Zeichnung eingefügt und die entsprechende Kante aus E_{M2} gelöscht.

Sei S' die Menge aller potenziellen Sehnen der Kanten aus E_M. Die Kosten $\alpha(S_{i-1} \cup \{c\})$ für das Einfügen einer Sehne c können einfach in $O(|S_{i-1}|)$ berechnet werden, da die Kosten von $\alpha(S_{i-1})$ bereits aus der vorherigen Iteration bekannt sind. Weiter wird festgestellt, dass $|S_{i-1}| = O(|E_M|)$. Die Laufzeit des Algorithmus lässt sich schlussendlich mit der oberen Schranke $O(|S'||E_M|^2)$ begrenzen. Die Berechnung aller Kosten braucht in jeder Iteration maximal $O(|S'||E_M|)$. Da der Algorithmus zudem noch maximal $|E_M|$ Iterationen durchläuft, ergibt sich diese Laufzeitbegrenzung.

Wie bereits erwähnt liefert dieser Algorithmus keine Garantie für ein optimales Ergebnis. Im Ausblick auf weitere Forschungen wurde daher auch für dieses Problem der Beweis der NP-Schwere im Paper gelistet. Dieses wird in Kapitel 5 bearbeitet und mit einer Reduktion von SAT bewiesen.

3 Beweis der NP-Schwere des Permutationsproblems

3.1 Problem der perfekten Permutation

Zunächst wird noch einmal das zu lösende Problem der perfekten Permutation definiert. Gegeben sei eine Menge von Farben C und eine Menge von Gruppen B. Sei L eine zirkuläre Liste von Tupeln (c, b) mit $c \in C$ und $b \in B$.

Für L sei $L(i)$ das i-te Tupel der Liste. Weiter sei $C(i) = c \in L(i)$ und $B(i) = b \in L(i)$ mit $c \in C$ und $b \in B$. Sei $i, j \in \mathbb{N}$. Für alle Tupel $L(i), L(j) \in L$ gilt $(B(i) = B(j) \rightarrow C(i) \neq C(j)) \wedge (C(i) = C(j) \rightarrow B(i) \neq B(j))$. Es gilt also, dass zwei Tupel nicht die selbe Farbe und die selbe Gruppe haben können. Die einzig erlaubte Operation ist dabei das Tauschen zweier Tupel $L(i), L(j)$ für die gilt $B(i) = B(j)$.

Problem 1 (Problem der perfekten Permutation). *Finde eine Permutation von L, so dass die Anzahl der Übergänge $L(i), L(i+1)$ mit $C(i) \neq C(i+1)$ minimal ist.*

Lemma 1. *Eine Permutation heißt dann perfekt, wenn die Anzahl der Tupel $L(i), L(i+1)$ mit $C(i) = C(i+1)$, durch die erlaubte Operation nicht mehr erhöht werden kann.*

Wie bereits beschrieben lässt sich der Kreis von Chord-Diagrammen als zirkuläre Liste beschreiben. Betrachtet man die Tupel gleicher Farbe als Kopien des selben Knotens so ist zu erkennen, dass dies nur eine veränderte Ausdrucksweise des bereits in Abschnitt 2.3 beschriebenen Problems ist.

Um folglich die NP-Schwere des Problems zu zeigen soll folgender Satz bewiesen werden:

Satz 2. *3-Vertex-Cover ist auf das Problem der perfekten Permutation reduzierbar.*

3.2 Idee

Zunächst wird kurz erläutert, wieso Set-Cover, auch mit der Einschränkung einer maximalen Größe für die Teilmengen, noch NP-schwer ist. Anschließend soll ausgehend von diesem Problem eine Reduktion auf das Problem der perfekten Permutation erfolgen. Dabei werden die einzelnen Teilmengen von Set-Cover als Reihenfolge von Elementen in L dargestellt. Die einzelnen Elemente des Universums werden als separierte Elemente ans Ende von L gesetzt. Ein Algorithmus für die perfekte Permutation würde dann die Elemente des Universums auf die einzelnen Reihenfolgen an Elementen aufteilen. Jede Teilmenge, deren Reihenfolge ein Element des Universums enthält, wäre dann in einer Set-Cover Lösung vorhanden.

3.3 Reduktion Vertex-Cover $\leq$ Set-Cover

Wir wissen, dass Vertex-Cover auch für planare Graphen mit Grad 3 NP-schwer ist. Aus der nun gezeigten Reduktion folgt demnach, dass auch Set-Cover mit 3-elementigen Teilmengen NP-schwer sein muss. Ein solches Problem soll im Folgenden 3-Set-Cover genannt werden.

Die Reduktionsfunktion $f : (G = (V, E), k) \rightarrow (U, S_1, \ldots, S_m, k')$, die ein Vertex-Cover Problem in ein Set-Cover Problem transformiert sei wie folgt definiert: $U = E$

Für jeden Knoten $v \in V$: erstelle Menge S_v mit allen zu v inzidenten Kanten. $k' = k$ Die Teilmengenmenge des Set-Cover Problems ist die Vereinigung aller S_v. Das Universum U sind die in G vorhandenen Kanten. Es gilt, dass eine Lösung für das eine Problem direkt in eine Lösung des anderen überführt werden kann.

1. Lösung für Vertex-Cover$(G, k) \Longrightarrow$ Lösung für Set-Cover$(f(G, k))$
   Wenn eine Menge W von Knoten mit $|W| = k$ alle Knoten abdeckt, dann deckt die Vereinigung X aller $\{S_v | v \in V\}$ mit $|X| = k$ alle Elemente in U ab.

2. Lösung für Set-Cover$(f(G, k)) \Longrightarrow$ Lösung für Vertex-Cover(G, k)
   Wenn eine Teilmenge $S_{v_1}, \ldots, S_{v_k}$ alle Elemente von U abdeckt, dann ist $\{v_1, \ldots, v_k\}$ ein Vertex-Cover für G

[KT06]

3.4 Reduktion 3-Set-Cover $\leq$ Perfekte Permutation

Sei $\mathcal{I}$ eine Set-Cover Instanz mit Universum U und Familie von Teilmengen S, mit $S = \{S_0, S_1, \ldots, S_m\}$, wobei $S_0 \cup S_1 \cup, \ldots, \cup S_m = U$ mit $|S_i| = 3$ und $i \in \{0, \ldots, m\}$. Sei k die Größe des kleinsten Covers. Die Funktion $g : (U, S, k) \rightarrow (L, C, B, k')$ transformiert die Set-Cover Instanz wie folgt:
$n = |U| + 5|S|$
$C = \{c_1, c_2, \ldots, c_n, u\}$
$B = U \cup \{z\}$, wobei $z \notin U$

Sei L zudem eine zirkuläre Liste, deren Aufbau im folgenden beschrieben wird.

Tupel mit der Farbe u stellen das Universum da. Die Permutation von L soll sich am Ende nur noch durch das Verschieben dieser Tupel verbessern lassen. Daher darf u als einzige Farbe mehrfach in L vorkommen. Elemente der Gruppe z dienen als Trennelemente.

Für alle Elemente $u_1, \ldots, u_{|U|} \in U$ definieren wir ein einzigartiges Tupel (u, u_j) mit $j \in \{1, \ldots, |U|\}$. Füge diese Tupel wie folgt in L ein:

$$\underbrace{(u, u_1)}_{Intervall\ von\ u_1}, (c_1, z), \underbrace{(u, u_2)}_{Intervall\ von\ u_2}, (c_2, z), \ldots, \underbrace{(u, u_{|U|})}_{Intervall\ von\ u_{|U|}} (c_{|U|}, z)$$

Die Tupel zwischen den Elementen der Gruppe z sollen als *Intervalle von* u_j bezeichnet werden. Diese Intervalle werden im folgenden auch *u-Intervalle* genannt.

Anschließend wird für jede Teilmenge $S_i \in S$ mit $s_1, s_2, s_3 \in S_i$ und $i \in \{0, \ldots, m\}$ folgende Reihenfolge an Tupeln an L angehängt. Zur übersichtlicheren Darstellung sei $n_i = |U| + 5i$

$$\underbrace{(c_{n_i+1}, s_1), (c_{n_i+2}, s_2), (c_{n_i+3}, s_3), (c_{n_i+4}, s_1)}_{Intervall\ von\ S_i}, (c_{n_i+5}, z)$$

Diese Reihenfolge ermöglicht es, dass zwei Tupel $(v, s_j), (w, s_k)$ mit $s_k \neq s_j, s_k, s_j \in S_i$ und $v, w \in C$ garantiert nebeneinander liegen können. Die in L eingefügte Tupelfolge ohne das Tupel der Gruppe z sei als *Intervall von* S_i definiert. Intervalle der Mengen $S_0, \ldots, S_m$ werden im folgenden als *S-Intervalle* bezeichnet.

Es sollen außerdem alle Intervalle, die im Zuge einer perfekten Permutation Tupel der Farbe u enthalten, als *verwendetes Intervall* bezeichnet werden.

Lemma 3. *Wenn die Tupel* $(u, u_1), \ldots, (u, u_{|U|})$ *auf* q *Intervalle aufgeteilt sind, dann verringert sich die Anzahl der Farbwechsel um* $|U| - q$

Beweis. Da alle u-Intervalle lediglich ein Element beinhalten können keine zwei Elemente der selben Farbe dort vorkommen. Diese können also ignoriert werden.

Für jedes Intervall der Mengen aus S gilt zu Beginn, dass kein Tupel die selbe Farbe hat. Des Weiteren können in jedem Intervall nur 3 Tupel der Farbe u getauscht werden, welche als einzige mehrfach vorkommt.

Tauscht man nun lediglich ein Tupel der Farbe u in das Intervall der Menge S_i so verbessert sich die Permutation nicht, da immer noch alle Farben unterschiedlich sind.

Innerhalb eines S-Intervalls können die Tupel garantiert so getauscht werden, dass alle Tupel mit Farbe u nebeneinander liegen. Jedes weitere Tupel, dass in das Intervall von S_i verschoben wird verbessert die Permutation also um 1.

Sei t_i die Anzahl an Tupeln mit Farbe u in dem Intervall von S_i. Die Permutation verbessert sich also für jedes Intervall von S_i, das ein entsprechendes Tupel der Farbe u enthält um $t_i - 1$, da das erste Tupel keine Verbesserung bringt.

Die Verbesserung der Permutation lässt sich durch

$$\sum_{i=0}^{m} \begin{cases} t_i - 1 & S_i \text{ enthält Tupel der Farbe } u \\ 0 & sonst \end{cases}$$

mathematisch darstellen. Wie oben erkennbar ist m die Anzahl aller Intervalle, beziehungsweise die Kardinalität von S.

Vereinfacht lässt sich diese Summe als $|U| - q$ schreiben, da die Summe aller t_i zwangsläufig $|U|$ sein muss wovon $q \cdot 1$ abgezogen werden. □

Wie in Lemma 3 gezeigt wird gilt, dass je weniger Intervalle verwendet werden, desto besser ist die Permutation und desto weniger Teilmengen benötigt das Set-Cover.

Aus einer perfekten Permutation von L ließe sich wie dann folgt, eine minimale Lösung für die Set-Cover-Instanz ablesen: Betrachte die Positionen der Tupel $(u, u_1), \ldots, (u, u_{|U|})$.

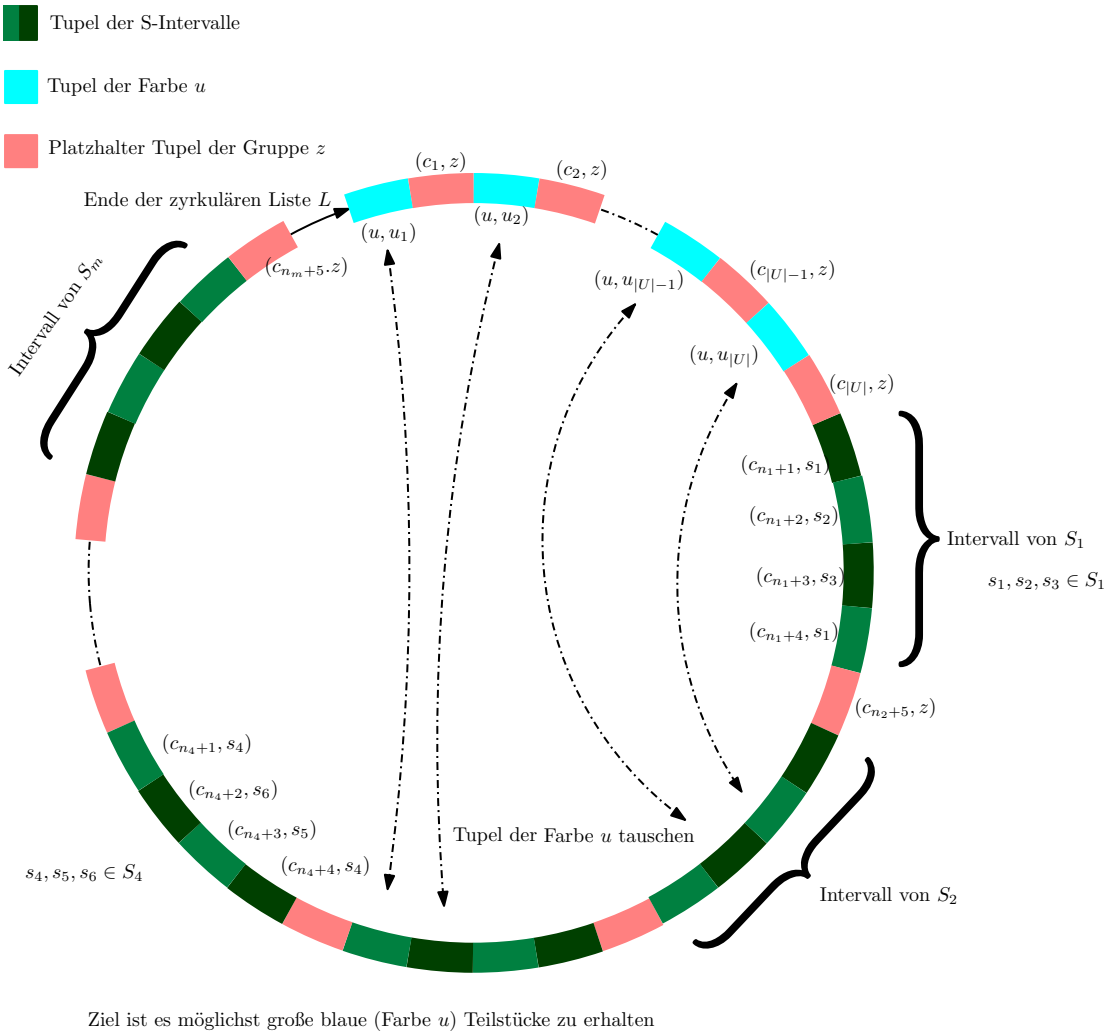


Abb. 3.1: Aufbau der zirkulären Liste L

Jedes dieser Tupel (u, u_j) muss entweder in ein Intervall einer Teilmenge S_i verschoben worden sein, oder in einem Intervall von u_j. Aus Lemma 3 folgt, dass wir die Tupel der Farbe u, die noch in u-Intervallen sind in beliebige S-Intervalle getauscht werden können, da sich die Anzahl der verwendeten Intervalle dadurch nicht erhöht und die Permutation sich dadurch auch nicht verschlechtert. Vertauscht man diese Tupel also in S-Intervalle, so ist ein kleinstes Set-Cover die Menge aller S_i, deren Intervall mindestens ein Tupel mit Farbe u enthält.

Sei $k' = k$ die Anzahl der verwendeten Intervalle. Es gilt zu beweisen, dass:

Lemma 4. *Die 3-Set-Cover Instanz $\mathcal{A}$ hat ein kleinstes Set-Cover der Größe $k \iff$ In einer durch g abgebildeten Liste $L_{\mathcal{A}}$ lassen sich alle Tupel der Farbe u auf nicht weniger als k Intervalle aufteilen.*

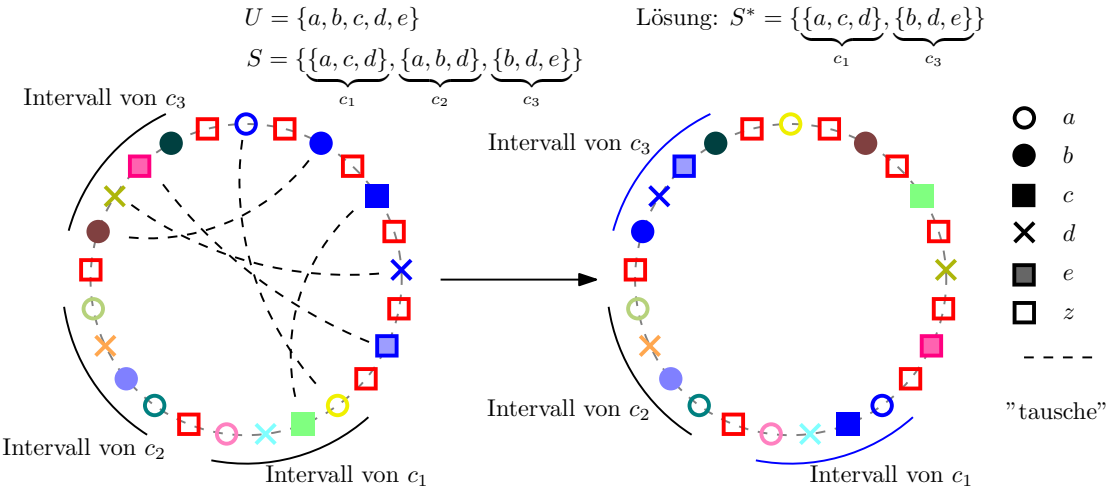


Abb. 3.2: Beispiel für eine Set-Cover Instanz mit entsprechenden Chord-Diagrammen. Links ungelöst und rechts mit einer perfekten Permutation. Die Gruppen sind durch verschiedene Formen visualisiert. Verwendete Intervalle sind in blau markiert.

3.5 Beweis der Gültigkeit

Gegeben sei eine beliebige 3-Set-Cover Instanz $\mathcal{A}$ mit dem Universum U und den Teilmengen $S_1, \ldots, S_m$. Die durch die Funktion g entstandene Liste sei $L_\mathcal{A}$. Sei $\mathcal{S}^*$ eine minimale Lösung für $\mathcal{A}$. Tausche alle Tupel mit Farbe u in beliebige Intervalle der Mengen von $\mathcal{S}^*$, so dass die Tupel der Farbe u in einem Intervall immer nebeneinander liegen. Die dadurch entstandene Permutation von $L_\mathcal{A}$ ist perfekt. Dies soll nun gezeigt werden.

Aus obigem Lemma leitet sich ab, dass sich eine Permutation nur über die Reduzierung der verwendeten Intervalle reduzieren lässt. Folglich verbessert sich die Permutation nur dann, wenn sich alle Tupel mit Farbe u von zwei oder mehr verwendeten Intervallen, fortan definiert als $\mathcal{S}_{DELETE}$, auf die Intervalle der Menge $(\mathcal{S}^* \setminus \mathcal{S}_{DELETE}) \cup \mathcal{S}_{NEW}$ aufteilen lassen, wobei $|\mathcal{S}_{DELETE}| > |\mathcal{S}_{NEW}|$.

Diese Permutation verwendet weniger Intervalle als $|\mathcal{S}^*|$. Die Mengen der verwendeten Intervalle könnten als Lösung für die Set-Cover Instanz $\mathcal{A}$ genutzt werden. Da $\mathcal{S}^*$ jedoch als minimal definiert ist, ist dies ein Widerspruch.

Gleichzeitig gilt, dass aus einer perfekten Permutation der Liste $L_\mathcal{A}$ einfach ein minimales Set-Cover für $\mathcal{A}$ abgeleitet werden kann.

Sei $\mathcal{S}^*$ die Menge der Mengen, der verwendeten Intervalle einer perfekten Permutation von $L_\mathcal{A}$ mit $|S'| = k$. Verschiebe alle Tupel der Farbe u in S-Intervalle. Aus Lemma 3 folgt, dass die Permutation dadurch nicht schlechter wird. Da alle Tupel der Farbe u, die das Universum darstellen, verteilt wurden ist S' ein Set-Cover für $\mathcal{A}$. Wenn es ein kleineres Set-Cover geben würde, so gäbe es zwangsläufig auch Darstellungen dieser Mengen, so dass alle Tupel mit u, darauf aufgeteilt werden könnten. In diesem Fall wäre die Anzahl der verwendeten Darstellungen kleiner als k. Da k jedoch minimal ist, kann kein kleineres Set-Cover für $\mathcal{A}$ existieren.

4 Algorithmus zur Lösung des Permutationsproblems mit zwei Gruppen

4.1 Perfekte Permutation mit zwei Gruppen

Sei $\mathcal{G}$ eine Chord-Graph Instanz mit der Menge an Ursprungsknoten C und den Gruppen $B = \{b_1, b_2\}$. Es gilt, dass jedes Kreisteilstück t aus $\mathcal{G}$ durch die bijektive Funktion

$$\gamma(t) \longrightarrow (c, b),$$

mit $c \in C$ und $b \in B$, als Tupel abgebildet wird. Zusätzlich ist als $\langle t_1, \ldots, t_q \rangle$ die Reihenfolge der Kreisteilstücke von $\mathcal{G}$ definiert. Dabei kann bei einem beliebigen Kreisstück begonnen werden, von dem aus dann der Kreis im Uhrzeigersinn abgegangen wird. Sei L eine zirkuläre Liste an Tupeln, wobei $L(i)$ mit $i \in \{1 \ldots, q\}$ das i-te Tupel in L beschreibt, die wie folgt, aus $\langle t_1, \ldots, t_q \rangle$ hervorgeht:

$$L(i) = \gamma(t_i)$$

Sei demnach $\gamma(\mathcal{G}) = L$. . Als S ist die Menge an Paaren (t_k, t_l) definiert, für die gilt $\gamma(t_k) = (c, b_1) \wedge \gamma(t_l) = (c, b_2)$. Zudem wird mit $C(i)$ der Ursprungsknoten, sowie mit $B(i)$ die Gruppe des i-ten Tupels in L beschrieben.

Gesucht ist, wie schon zuvor, eine Permutation der Elemente von L, so dass die Anzahl der Übergänge $L(i), L(i+1)$ mit $C(i) \neq C(i+1)$ minimal ist. Hierzu wird die Anzahl der Übergange mit $C(i) = C(i+1)$ maximiert (Lemma 1). Weiterhin gilt auch, dass es nicht mehrere Tupel mit gleichem Ursprungsknoten und Gruppen geben kann und die einzig zugelassene Operation das Tauschen zweier Tupel mit der gleichen Gruppe ist.

4.2 Erläuterung

Folgender Satz soll nun bewiesen werden.

Satz 5. *Eine perfekte Permutation der Kopien eines Chord-Graphen mit lediglich zwei Gruppen, ist in effizienter Zeit zu finden.*

Der Algorithmus sucht zunächst die maximale Anzahl an gleichzeitig betrachtbaren Wechseln. Ein Wechsel ist dabei definiert als Tupel zweier Indizes i, j wobei $i = j - 1$. Ein Wechsel beschreibt einen Übergang zweier Tupel mit unterschiedlichen Gruppen. Die Gruppen der Elemente in L an den Stellen i und j müssen also unterschiedlich sein. Es dürfen nur Wechsel betrachtet werden für die gilt, dass keiner ihrer Indizes bereits in

einem anderen betrachteten Wechsel vorhanden ist. Diese Wechsel geben später an, an welchen Stellen ein Übergang zweier Tupel mit gleichem Ursprungsknoten möglich ist.

Präzise formuliert ergibt sich folgendes Lemma:

Lemma 6. *Es gilt $i = j + 1$ und $i \in \{1, \ldots, q\}$. Sei W eine Menge an Wechseln (i, j) und w ein beliebiger Wechsel mit $j \in w \oplus i \in w$. Weiter gilt für W:*

$$(i, j) \in W \longrightarrow w \notin W$$

Der erste Algorithmus sucht also die Menge W der Indizes aus L, wobei $|W|$ maximal ist.

Dabei arbeitet er nach dem Greedy-Prinzip und nimmt immer das erstbeste Objekt. Man unterscheidet zwei Fälle:

1. In L existiert mindestens ein Index i, so dass $B(i) = B(i-1)$: Man betrachte das Teilstück zwischen $i \in \{1, \ldots q\}$ mit $B(i) = B(i-1) \neq B(i+1)$ und dem im Uhrzeigersinn nächsten Index $j \in \{1, \ldots q\}$ mit $B(j) = B(j+1) \neq B(j-1)$ mit $i \neq j$, so gibt es für die Tupel dazwischen zwei Möglichkeiten:
   a) Anzahl Tupel gerade: In diesem Fall muss $B(i) \neq B(j)$ sein. Der Listenabschnitt $L(i)$ bis $L(j)$ (inkl.) besteht ist aus Zweierpaaren der Gruppen b_x *und* b_y mit $x, y \in \{1, 2\}$, $x \neq y$, die sich wie folgt aneinander reihen:

   $$[B_x(i), B_y(i+1)], [B_x(i+2), B_y(i+3)], \ldots, [B_x(j-1), B_y(j)]$$

   Hier können alle Tupel in Wechseln gleichzeitig betrachtet werden, indem immer die Indizes der Zweierpaare genommen werden. Der Greedy-Algorithmus beginnt beim ersten Tupel des Listenabschnitts und wählt den Wechsel $(i, i+1)$. Da die Wechsel immer zwei Indizes enthalten springt der Algorithmus zwei Indizes weiter und betrachtet folglich immer Paare $(i+k, i+1+k)$ *mit* $z \in \mathbb{N} +$ *und* $k = 2 * z$. Da diese Paare garantiert noch nicht betrachtet wurden und nach Definition der Reihenfolge von oben, gültige Wechsel sein müssen wird der Algorithmus das Paar akzeptieren und zwei Indizes weiter gehen. Es werden also die Indizes aller Tupel aufgeteilt, die Anzahl ist also maximal.
   b) Anzahl der Tupel ungerade: Sei $L(i)$ bis $L(j)$ immer noch der betrachtete Listenabschnitt. Jeder ungerade Abschnitt hat folgenden Aufbau: $[B_x(i), B_y(i+1)], [B_x(i+2), B_y(i+3)], \ldots, [B_x(j-2), B_y(j-1)], [B_x(j)]$. In wurde gezeigt, dass der Teil $[B_x(i), B_y(i+1)], [B_x(i+2), B_y(i+3)], \ldots, [B_x(j-2), B_y(j-1)]$ perfekt aufgeteilt wird. Möchte man nun $[B_x(j)]$ noch in einem Wechsel betrachten, so müsste man nach Lemma 6 einen anderen Wechsel verwerfen. Die Anzahl ist also maximal.

   Da für alle Abschnitte gilt, dass mit dem Algorithmus die maximale Anzahl an Wechseln ermittelt wird, muss dies folglich auch für L gelten, da die Anzahl der Wechsel in L die Summe über alle Wechsel der Abschnitte ist.

2. In L existiert kein solcher Index: Sollte kein Index mit den gegebenen Bedingungen existieren folgt daraus, dass L eine gerade Anzahl an Tupel besitzen muss, da sonst garantiert zwei Tupel der gleichen Gruppe aufeinanderfolgen müssen. In diesem Fall können wir einen beliebigen Startpunkt wählen, da sich die Tupel wie in a) gezeigt vollständig aufteilen lassen.

Im zweiten Schritt des Algorithmus werden die Tupel auf die Wechsel verteilt. In der Menge S sind alle Tupel als Paar enthalten, deren Ursprungsknoten zweimal in L vorkommen. Der Algorithmus teilt diese Tupel auf die betrachteten Wechsel auf. Dabei gibt es zwei zu betrachtende Szenarios:

1. $|S| <= |W|$: In diesem Fall kann jedes Tupelpaar mit $[(c, b_1), (c, b_2)]$ und $c \in C$ nebeneinander platziert werden. Die Anzahl der Übergange mit $C(i) = C(i + 1)$ kann also nicht mehr erhöht werden und die Permutation ist perfekt.

2. $|S| > |W|$: In diesem Fall werden so viele Paare wie möglich an die Stellen mit betrachteten Wechseln gesetzt. Da die Anzahl der Wechsel maximal ist, sind bereits so viele Tupel mit selbem Ursprungsknoten wie möglich in L direkt aufeinanderfolgend. Für die übrigen Paare gibt es keine Option sie günstiger zu platzieren, ohne dabei ein bereits vorhandenes Paar in L zu zerstören. Auch hier ist die Permutation also perfekt.

Es folgt daher das Lemma:

Lemma 7. *Sei $\mathcal{G}$ eine Chord-Diagramm Instanz, sowie L die durch $\gamma(\mathcal{G})$ abgebildete Liste. Sei $|W|$ die Kardinalität der größten Menge an gleichzeitig betrachtbaren Wechseln aus L. Dann gibt es bei einer perfekten Permutation in L genau $|W'|$ aufeinanderfolgende Tupel $(a, b_1), (c, b_2)$ mit $a = c$*

Für die übrigen Kopien, welche nicht in einem Paar in S vorhanden waren, ist die Position egal.

4.3 Pseudocode

Input: Die Zirkuläre Liste L bestehend aus Tupeln (c, b) mit $b \in \{b_1, b_2\}$, sowie die Menge S wie bereits oben definiert
Result: Perfekte Permutation der Liste L, s.d. möglichst viele Tupel mit gleichen c direkt aufeinanderfolgen

```
/* maximale Anzahl an betrachteten Gruppenwechseln identifizieren
   */
```
Liste W = empty
int i = beliebiger Index von L für den gilt $L(i) \in b_x$, $L(i-1) \in b_x$ und $x \in \{1, 2\}$
```
 /* Zwei Tupel der selben Gruppe hintereinander */
```

Falls ein solcher Index nicht existiert: $i = 0$;
int $a = 0$ `/* Counter */`

```
/* wähle greedy immer erstbesten Wechsel und füge ihn zur Liste
   hinzu                                                        */
```
while $a < |L|$ **do**
  if $B(a + i) \neq B(a + i + 1)$ **then**
    W add (a+i, a+i+1))
    $a = a + 2$
  else
    $a = a + 1$

```
/* Befülle die betrachteten Wechsel mit passenden Paaren        */
```
while S is not empty **do**
  wähle beliebigen Wechsel $w = (n, m)$ aus W mit $B(n) = b_1, B(m) = b_2$
  wähle beliebigen Paar $p = (p_1, p_2)$ aus S mit $b_1 \in p_1, b_2 \in p_2$
  /* w enthält die Indizes n für Gruppe b_1 und m für b_2 */

  /* p enthält die ein Tupel p_1 mit b_1 und ein Tupel p_2 mit b_2 */

  tausche $L(n)$ mit p_1 und $L(m)$ mit p_2
  entferne p aus S
  entferne w aus W'

5 Beweis der NP-Schwere beim Einfügen der Sehnen

5.1 Überblick

In diesem Kapitel soll gezeigt werden, dass das Einfügen von Sehnen in ein Chord-Graphen, s.d. die Anzahl der Kreuzungen minimal ist, NP-Schwer ist. Dazu wird *SAT* auf dieses Problem reduziert. Anschließend kann außerdem gezeigt werden, dass das Problem nicht beliebig gut approximierbar ist, also APX-schwer ist.

5.2 Problem der kreuzungsminimalen Sehnenmenge

Gegeben sei ein Graph G mit Knotenmenge V und Kantenmenge E. Sei L eine zirkuläre Liste an Instanzen. Sei $\phi : L \to V$ eine Abbildung, die jeder Instanz $v' \in L$ einen Knoten $v \in V$ zuweist. Es gilt $|V| \leq |L|$. Sei außerdem $|v'|$ der Index der Instanz v' in L. Eine Kreuzung zweier Sehnen $\overline{u'v'}, \overline{w'y'}$ existiert, falls $|u'| < |w'| < |v'| < |y'|$.

Problem 2 (Problem des kreuzungsminimalen Sehnenmenge). *Finde eine Sehnenmenge $S \subseteq \binom{L}{2}$, so dass*

1. *zu jeder Kante $uv \in E$ eine Sehne $\overline{u'v'} \in S$ existiert, so dass $\phi(u') = u$ und $\phi(v') = v$.*
2. *die Anzahl an Kreuzungen aller Sehnen der Menge S minimal sind.*

Um die NP-Schwere des kreuzungsminimalen Sehneneinfügens zu zeigen, wird folgender Satz bewiesen.

Satz 8. *SAT ist auf das Problem des kreuzungsfreien Sehneneinfügens reduzierbar.*

Präziser formuliert lässt sich sagen, dass eine Abbildung $F \to (G_F, S_F)$ existiert, so dass die CNF-Formel F genau dann erfüllbar ist, wenn eine kreuzungsfreie Sehnenmenge existiert.

5.3 Reduktion

Um zu zeigen, dass das Problem des Sehneneinfügens NP-schwer ist, wird mit *SAT* reduziert. Sei $\mathcal{F}$ eine aussagenlogische Formel, in konjunktiver Normalform. Wenn die Formel erfüllbar ist, so werden die Sehnen ohne Kreuzungen eingefügt.

Die Reduktionsfunktion sei $g : (\mathcal{F}) \to (L, V, E)$. Sie transformiert die SAT-Instanz wie folgt:

Seien $c_0, \ldots, c_m$ die Klauseln von $\mathcal{F}$ und $x_0, \ldots, x_n$ alle in $\mathcal{F}$ vorkommenden Variablen.

$$V = \left\{ \underbrace{v_0, \ldots, v_m,}_{\text{einen Knoten pro Klausel}} \quad \underbrace{\overbrace{w_0, u_0}^{\text{Knoten für } x_0} \;,\; \overbrace{w_1, u_1}^{\text{Knoten für } x_1} \;, \ldots, \; \overbrace{w_n, u_n}^{\text{Knoten für } x_n}}_{\text{zwei Knoten für jede Variable}}, \; s \right\}$$

$$E = \left\{ (v_0, s),\ (v_1, s), \ldots, (v_m, s), \quad \underbrace{(w_0, u_0),\ (w_1, u_1), \ldots, (w_n, u_n)}_{\text{Kante zw. den zwei Knoten einer Variable}} \right\}$$

Des Weiteren geht die zirkuläre Liste L wie folgt aus $\mathcal{F}$ hervor:

Für jede Variable x_i mit $i \in \{0, \ldots, n\}$ seien zunächst folgende Knotenmenge definiert:

Sei T_i die Menge aller Knoten v_j mit c_j ist erfüllt, falls x_i wahr ist.

Sei F_i die Menge aller Knoten v_j mit c_j ist erfüllt, falls x_i falsch ist.

Sei $\mathcal{T}_i$ eine Menge an Instanzen, so dass jede Instanz aus $\mathcal{T}_i$ durch ϕ auf genau einen Knoten aus T_i abbildet wird. Analog gilt dies für $\mathcal{F}_i$ und F_i.

Für jede Variable x_i hänge folgende Reihenfolge an Instanzen an L an.

$$f_i, \quad \mathcal{T}_i, \quad p_i, \quad \mathcal{F}_i, \; t_i$$

Seien dabei f_i, t_i, p_i Instanzen, für die gilt $\phi(f_i) = \phi(t_i) = w_i$ und $\phi(p_i) = u_i$.

Zuletzt muss noch eine Instanz s' des Knotens s ganz am Ende der Liste eingefügt werden. Dieser fungiert als Startpunkt für die Kanten $(v_0, s),\ (v_1, s), \ldots, (v_m, s)$, die anzeigen, welche der Variablen ihrer Klausel $c_0, c_1, \ldots, c_m$ erfüllt ist.

Sollte Variable x_i **falsch** sein, wird für die Kante (w_i, u_i) die Sehne zwischen den Instanzen f_i und p_i gewählt. Dadurch können die Sehnen der Kanten (v_k, s), mit $v_k \in F_i$, ohne Kreuzung eingefügt werden. Diese Kanten repräsentieren die entsprechenden Klauseln c_k welche x_i in negierter Form enthalten. Analog ist, falls x_i **wahr** ist, die Sehne zwischen den Instanzen t_i und p_i zu wählen. Dadurch können äquivalent zu oben alle Sehnen der Kanten (v_k, s) mit $v_k \in T_i$ ohne Kreuzung eingefügt werden.

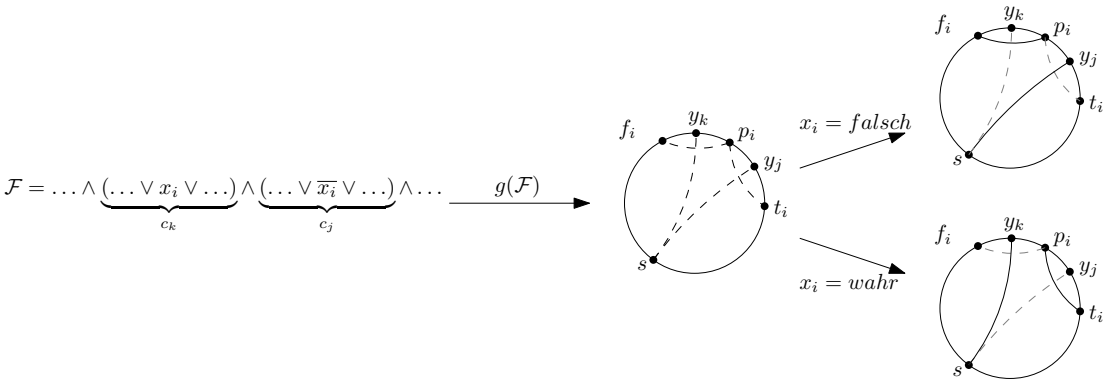


Abb. 5.1: Durch g entstehende Abbildung einer Variablen

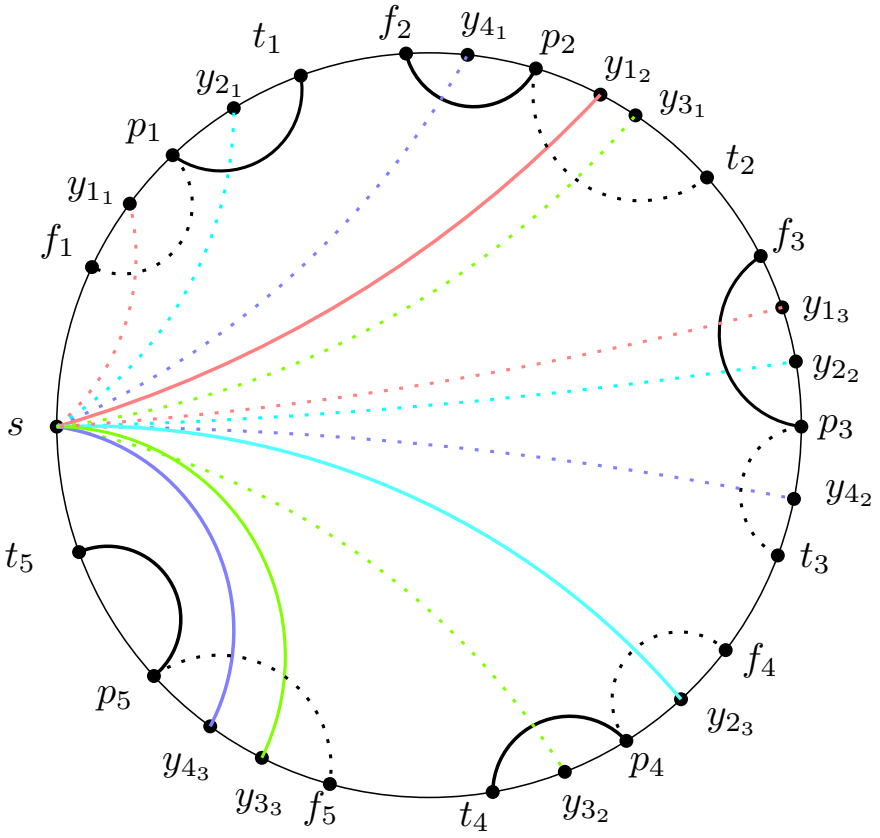
Anschließend wird nun folgendes Lemma bewiesen.

Lemma 9. *Wenn die Formel $\mathcal{F}$ erfüllbar ist, so fügt ein potentieller Algorithmus die Sehnen der Kanten aus E so ein, dass es keine Kreuzungen gibt.*

Anhand der Wahl für die Kanten (w_0, u_0), $(w_1, u_1), \ldots, (w_n, u_n)$ kann dann abgeleitet werden, ob die Variable den Wert *wahr* oder den Wert *falsch* annehmen muss, damit $\mathcal{F}$ erfüllt ist.

$$\mathcal{F} = \underbrace{(x_1 \vee \overline{x_2} \vee x_3)}_{c_1} \wedge \underbrace{(\overline{x_1} \vee x_3 \vee x_4)}_{c_2} \wedge \underbrace{(\overline{x_2} \vee \overline{x_4} \vee x_5)}_{c_3} \wedge \underbrace{(x_2 \vee \overline{x_3} \vee x_5)}_{c_4}$$

Lösung: $x_1, \overline{x_2}, \overline{x_3}, x_4, x_5$



Es gilt: $\phi(y_i) = v_i$ $\qquad$ $y_{i_k} = k$-te Instanz die auf v_i abgebildet wird

Abb. 5.2: Beispiel: Abbildung einer KNF Formel mit entsprechender Umsetzung der einer möglichen Lösung

5.4 Beweis

Sei $\mathcal{F}$ die aussagenlogische Formel in konjunktiver Normalform. Sei $\mathcal{B}$ außerdem eine Variablenkonstellation, die für jede Variable $x_0, \ldots, x_n$ aus $\mathcal{F}$ den Wert *wahr* oder *falsch* enthält. Sei $\mathcal{F}$ mit $\mathcal{B}$ erfüllt.

Um zu zeigen, dass die Reduktion gültig ist, soll zuerst gezeigt werden, dass sich aus $\mathcal{B}$ ableiten lässt, wie die Kanten E die aus der Funktion $g : (\mathcal{F}) \to (L, V, E)$ hervorgingen, ohne Kreuzung als Sehnen in L eingefügt werden können.

Da $\mathcal{F}$ mit $\mathcal{B}$ erfüllt ist muss es für jede Klausel $c_0, \ldots, c_m$ mindestens eine Variable x_i mit $i \in \{0, \ldots, n\}$ geben, mit deren Wert diese erfüllt ist. Diese wird repräsentiert von v_i Füge die Sehne für (w_i, u_i) wie oben beschrieben ein. Sollte Variable x_i **falsch** sein wird für die Kante die Sehne zu *Instanz 1* f_i gewählt. Falls x_i **wahr** ist, ist die Sehne zwischen zwischen der *Instanz 2* und der Instanz p_i zu wählen. Dabei ist folgendes zu beobachten:

Beobachtung 10. *In der durch g entstandenen Liste L können sich die Sehnen der Kanten w_i, u_i und (w_j, u_j) mit $i, j \in \{0, \ldots, n\}$ nie kreuzen.*

Da alle Instanzen der Knoten w_i, u_i entweder vor oder hinter w_j und u_j eingefügt wurden, kann die Definition der Kreuzung nicht erfüllt werden.

Es müssen noch Sehnen für alle Kanten der Form (v_k, s) mit $k \in \{0, \ldots, m\}$ gewählt werden. Hierbei kann folgendes festgestellt werden:

Beobachtung 11. *In einer durch g entstandenen Liste können sich zwei Sehnen der Kanten $(v_i, s), (v_j, s)$ mit $i, j \in \{0, \ldots, m\}$ nie kreuzen.*

Da s nur eine Instanz in L hat kann die Definition der Kreuzung nicht erfüllt werden.

Wie oben erwähnt, gibt es für jede Klausel c_k eine erfüllende Variable x_i. Durch das Einfügen aller Instanzen der Knoten aus der Menge T_i zwischen f_i und der Instanz p_i und dem Platzieren der Instanzen zu Menge F_i zwischen der Instanz p_i und der *Instanz 2*, t_i muss eine der folgenden Reihenfolgen in L vorkommen. Sei y eine Instanz mit $\phi(y) = v_k$

$$x_i \text{ ist wahr} \longrightarrow f_i, \ldots, y, \ldots, p_i, \ldots, t_i$$
$$x_i \text{ ist falsch} \longrightarrow f_i, \ldots, p_i, , \ldots, y \ldots, t_i$$

Sollte x_i wahr sein folgt, dass p_i und t_i verbunden wurden. Die Sehne der Kante (v_k, s) kann also zwischen der Instanz y und s' gespannt werden. Dadurch ergibt sich für die gewählten Instanzen der Sehnen, die die Kanten (v_k, s) und (w_i, u_i) abbilden folgende Reihenfolge:

$$y, \ldots, p_i, \ldots, t_i, \ldots, s'$$

Somit gilt:

$$|y| < |p_i| < |t_i| < |s'|$$

Daher können sich die beiden Sehnen nicht kreuzen.

Sollte x_i falsch sein folgt, dass p_i und f_i verbunden wurden. Die Sehne der Kante (v_k, s) wird auch hier zwischen der Instanz y und s' gespannt. In diesem Fall gibt es ähnlich zu oben folgende Reihenfolge:

$$f_i, \ldots, p_i, \ldots, y, \ldots, s'$$

Somit gilt:

$$|f_i| < |p_i| < |y| < |s'|$$

Auch hier ist eine Kreuzung ausgeschlossen.

Es lässt sich außerdem beobachten, dass:

Beobachtung 12. *Wenn $\overline{y, s'}$ eine Sehne von (v_k, s) ist, dann kann sich diese nur mit Sehnen der Kante zwischen w_i und u_i kreuzen.*

Jede Sehne der Form $\overline{y, s'}$ kann sich also nur einmal kreuzen.

Für alle Variablen x_h, x_i, x_l mit $h, l \in \{0, \ldots, n\} \setminus \{i\}$ und $h < i < l$ gilt aufgrund dessen, wie die Knoten in L eingefügt wurden, dass $|f_h|, |p_h|, |t_h| < |f_i|, |p_i|, |t_i|, |y| < |f_l|, |p_l|, |t_l| < |s'|$. Dadurch kann für die Sehnen der Kanten $(v_k, s), (w_y, u_y)$ und (w_z, u_z) die Definition der Kreuzung nicht erfüllt werden.

Zusammenfassend gilt also, dass Kanten der Form (w_i, u_i) sich nicht untereinander schneiden können.(Beo. 10) Ebenso können sich alle Kanten der Form (v_k, s) nicht untereinander schneiden.(Beo. 11) Falls $\mathcal{B}$ erfüllend ist existiert für jede Kante (v_k, s) mindestens eine Instanz y mit der die Sehne $\overline{y, s'}$ ohne Kreuzung eingefügt werden kann. Diese kann sich außerdem auch nicht mit anderen Sehnen der Kanten (w_i, u_i) kreuzen.(Beo. 12)

Somit können alle Kanten aus E ohne Kreuzung eingefügt werden, falls $\mathcal{F}$ von $\mathcal{B}$ erfüllt ist.

Nun muss noch gezeigt werden, dass aus einer kreuzungsfreien Kombination an Sehnen der Kanten aus E eine erfüllende Variablenkonstellation $\mathcal{C}$ folgt.

Um die Sehnen der Kanten (v_k, s) einzufügen müssen Instanzen $y_1, \ldots, y_m$, mit $\phi(y_k) = v_k$ und $k \in \{0, \ldots, m\}$ gewählt werden, von welcher aus die Sehne gespannt wird. Jede Instanz y_k muss mindestens Element einer Menge $\mathcal{T}_i$ oder $\mathcal{F}_i$ sein. Die Sehne der Kante (w_i, u_i) wird so gespannt, dass sie alle potentiellen Sehnen zu den Instanzen in $\mathcal{F}_i$ kreuzt, falls x_i wahr ist. Falls x_i falsch ist schneidet sie alle potentiellen Sehnen zu den Instanzen in $\mathcal{T}_i$. Sei $j, k, h \in \{0, \ldots, m\}$. Es gelten folgende Äquivalenzen:

x_i wahr $\iff$ Die Klauseln c_j mit $v_j \in T_i$ sind erfüllt $\iff$ alle Sehnen der Kanten mit Knoten v_j können ohne Kreuzung eingefügt werden

x_i falsch $\iff$ Die Klauseln c_k mit $v_k \in T_i$ sind erfüllt $\iff$ alle Sehnen der Kanten mit Knoten v_k können ohne Kreuzung eingefügt werden

Jede Klausel c_h hat mindestens eine Variable. Demnach ist v_h mindestens Element einer Menge T_i oder F_i. Dadurch kann man die Äquivalenz vereinfachen:

Die Klausel c_h ist erfüllt $\Longleftrightarrow$ die Sehne der Kante mit Knoten v_h kann ohne Kreuzung eingefügt werden.

Wenn also alle Sehnen ohne Kreuzung eingefügt wurden folgt daraus, dass alle Klauseln erfüllt sind. Weiter lässt sich feststellen, dass die Formel nicht erfüllt sein kann, wenn eine Kreuzung vorkommt, eine Klausel also nicht erfüllt ist. Da ein potentieller Algorithmus die Anzahl der Kreuzungen minimiert, ist anzunehmen, dass es keine Möglichkeit gibt die Sehnen ohne Kreuzung zu platzieren und die aussagenlogische Formel nicht erfüllbar ist.

5.5 Max-SAT

Satz 13. *Das Einfügen von Sehnen in ein Chord-Diagrammm, so dass die Anzahl der Kreuzungen minimal ist, ist APX-schwer, selbst wenn für jede Sehne maximal zwei Optionen zum Einfügen bestehen.*

Durch die oben erarbeitete Äquivalenz lässt sich zudem zeigen, dass das Problem auch dann NP-schwer ist, wenn für jede Kante nur zwei Optionen bestehen. Weiter lässt sich dadurch außerdem zeigen, dass das Problem sogar mindestens APX-schwer ist. Hierzu wird das APX-vollständige Problem Max-SAT verwendet [HZ]. Die aussagenlogische Formel der Max-SAT-Instanz wird dazu wie bereits SAT mit g transformiert. Es gilt also auch hier:

Die Klausel c_h ist erfüllt $\Longleftrightarrow$ die Sehne der Kante mit Knoten v_h kann ohne Kreuzung eingefügt werden.

Aus Beo. 12 geht hervor, dass jede Sehne der Kante (y, s') maximal eine Kreuzung haben kann. Daher gibt es auch für jede nicht erfüllte Klausel genau eine Kreuzung. Folglich ist wenn die Anzahl der Kreuzungen minimal ist, die Anzahl der erfüllten Klauseln maximal. Da Max-SAT auch NP-schwer ist, wenn jede Klausel nur zwei Variablen enthält und außerdem APX-vollständig liegt, folgt, dass das Einfügen der Sehnen mindestens genau so schwer ist.

6 Fazit

Ziel dieser Arbeit war es, die Schwierigkeit beim Erstellen von Chord-Diagrammen nach dem Vorbild von Angori et al. aus deren Arbeit über das Chord-Link Modell zu zeigen.

In Kapitel 3 wurde mit dem Beweis des Satzes

Satz 2. *3-Vertex-Cover ist auf das Problem der perfekten Permutation reduzierbar.*

die NP-Schwere des Permutationsproblems bewiesen. Diese wurde bereits in der vorangegangenen Arbeit vermutet, jedoch nicht belegt. Dass diese Schwierigkeit erst bei einer gewissen Komplexität des Problems auftritt, konnte in Kapitel 4 gezeigt werden. Dort wurde durch den Beweis von

Satz 5. *Eine perfekte Permutation der Kopien eines Chord-Graphen mit lediglich zwei Gruppen, ist in effizienter Zeit zu finden.*

gezeigt, dass das Permutieren eines Chord-Graphen effizient möglich ist, sollte dieser maximal zwei adjazente äußere Knoten besitzen.

Weiterhin offen ist die Frage ab, wann genau die Problematik NP-schwer ist. In den geführten Beweisen wurde gezeigt, dass die NP-Schwere gegeben ist, wenn die Gruppenanzahl, die Anzahl der Elemente in jeder Gruppe und die Verteilung der Kopien auf dem Kreis nicht eingeschränkt sind. Das Problem ist nicht mehr NP-schwer sobald die Anzahl der Gruppen auf zwei limitiert wird. Es stellt sich also die Frage, mit welchen Einschränkungen eine effiziente Lösung gefunden werden kann.

Weiter wurde in Kapitel 5 noch die Schwere des Einfügens der Sehnen betrachtet. Auch hier wurde bereits im Paper die NP-Schwere vermutet und es konnte mit einer Reduktion

Satz 8. *SAT ist auf das Problem des kreuzungsfreien Sehneneinfügens reduzierbar.*

gezeigt werden, dass das diese Vermutung zutrifft. Falls $P \neq NP$ wurde durch

Satz 13. *Das Einfügen von Sehnen in ein Chord-Diagrammm, so dass die Anzahl der Kreuzungen minimal ist, ist APX-schwer, selbst wenn für jede Sehne maximal zwei Optionen zum Einfügen bestehen.*

außerdem belegt, dass diese Problematik in effizienter Zeit nicht beliebig gut lösbar ist, also nicht beliebig gut approximiert werden kann. Dies erschwert die Entwicklung einer guten Heuristik merklich.

Im Gegensatz zum Permutationsproblem konnte hier bereits gezeigt werden, dass auch starke Beschränkungen das Problem nicht vereinfachen. Zwar ist es möglich, eine kreuzungsfreie Kombination in effizienter Zeit zu finden, falls maximal zwei Optionen für jede Sehne bestehen. Das Finden einer kreuzungsminimalen Zeichnung ist jedoch auch

dann NP-schwer. Eine mögliche Einschränkung, die das Problem vereinfachen könnte, wäre die Permutation des Chord-Diagramms einzuschränken.

Obwohl dieses neue Wissen beweist, dass Einzelschritte des Frameworks bei der Zeichnung von Chord-Diagrammen nicht effizient lösbar sind, kann dies kaum als großes Hindernis für diese hybride Darstellungsform angesehen werden. Dies ist zunächst auf die Größe der als Chord-Diagramm darzustellenden Knotenmenge zurückzuführen. Die Visualisierung von großen Knotenmengen als Chord-Graph ist weder vorteilhaft noch im Chord-Link Modell vorgesehen. Sollte also ein exaktes Ergebnis forciert werden, so ließen sich exakte Lösungen für die thematisierten Probleme mit optimierten Algorithmen wahrscheinlich in annehmbarer Zeit generieren.

Sollten diese Zeiten jedoch für ein, wie im Paper vorgestelltes, interaktives System ungeeignet sein, so ist die Verwendung von Heuristiken in diesem Falle durchaus sinnvoll. Ob ein großer Vorteil einer exakten Lösung gegenüber einer guten Heuristik besteht ist in diesem Zusammenhang fraglich. Schlussendlich muss die Visualisierung für den Betrachter verständlich und nicht mathematisch exakt sein. Die Entwicklung solcher Heuristiken oder optimierter Algorithmen ist weiterhin ein interessantes Forschungsthema. Auch die Fragestellung, ob sich speziell für das Einfügen der Sehnen ein Algorithmus finden lässt, der dem in Abschnitt 2.5 beschriebenen Vorgehen in der Güte des Ergebnisses übertrifft, ist noch offen.

Eine Verbesserung, die das Zeichnen einfacher und die Übersicht verbessern könnte, wäre meiner Auffassung das Lockern der Struktur außerhalb der Chord-Graphen. Ob sich diese Vermutung jedoch bewahrheitet, kann anhand dieser Arbeit nicht geklärt werden.

Im Allgemeinen kann also gesagt werden, dass das Chord-Link Modell durchaus den angedachten Zweck erfüllt, unabhängig von der Komplexität des Zeichnens.

Literaturverzeichnis

[ADLB+17] Patrizio Angelini, Giordano Da Lozzo, Giuseppe Battista, Fabrizio Frati, Maurizio Patrignani und Ignaz Rutter: Intersection-Link Representations of Graphs. *Journal of Graph Algorithms and Applications*, 21:731–755, 2017, 10.7155/jgaa.00437.

[ADM+19] Lorenzo Angori, Walter Didimo, Fabrizio Montecchiani, Daniele Pagliuca und Alessandra Tappini: ChordLink: A New Hybrid Visualization Model. 2019.

[CLC+19] Shaw Ji Chen, Ding Lieh Liao, Chia Hsiang Chen, Tse Yi Wang und Kuang Chi Chen: Construction and Analysis of Protein-Protein Interaction Network of Heroin Use Disorder. *Scientific Reports*, 9(1), 2019, 10.1038/s41598-019-41552-z. `https://doi.org/10.1038/s41598-019-41552-z`.

[Har88] David Harel: On Visual Formalisms. *Commun. ACM*, 31(5):514–530, 1988, 10.1145/42411.42414, ISSN 0001-0782. `https://doi.org/10.1145/42411.42414`.

[HRFM07] Nathalie Henry Riche, Jean Daniel Fekete und Michael McGuffin: NodeTrix: a Hybrid Visualization of Social Networks. *IEEE transactions on visualization and computer graphics*, 13:1302–9, 2007, 10.1109/TVCG.2007.70582.

[HZ] Eran Halperin und Uri Zwick: Approximation Algorithms for MAX 4-SAT and Rounding Procedures for Semidefinite Programs. *Journal of Algorithms*, 40:184–211, 10.1006/jagm.2001.1162.

[KT06] J. Kleinberg und É. Tardos: *Algorithm Design.* Alternative Etext Formats. Pearson/Addison-Wesley, 2006, ISBN 9780321295354. `https://books.google.de/books?id=OiGhQgAACAAJ`.

[RMF12] Sebastien Rufiange, Michael J. McGuffin und Christopher P. Fuhrman: TreeMatrix: A Hybrid Visualization of Compound Graphs. *Computer Graphics Forum*, 31(1):89–101, 2012, https://doi.org/10.1111/j.1467-8659.2011.02087.x. `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2011.02087.x`.

[Wes87] W. Wessel: Chord Graphs - New Means for Representing Graphs. *Zastosowania Matematyki Applicationes Mathematicae*, 1987.

Erklärung

Hiermit versichere ich die vorliegende Abschlussarbeit selbstständig verfasst zu haben, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, und die Arbeit bisher oder gleichzeitig keiner anderen Prüfungsbehörde unter Erlangung eines akademischen Grades vorgelegt zu haben.

Würzburg, den 18. Februar 2021

. . . . . .
Jan Sauer