Höhenminimierung einfacher Tangles

Bachelorarbeit
von

Jakob Baumann

An der Fakultät für Informatik und Mathematik
Chair of Theoretical Computer Science

Betreuer: Prof. Dr. Ignaz Rutter

Bearbeitungszeit: 14. Juni 2020 – 14. September 2020

Eigenständigkeitserklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbständig und ohne unzulässige Hilfe verfasst und keine anderen als die angegeben Quellen und Hilfsmittel benutzt sowie die wörtlich uns sinngemäß übernommenen Passagen aus anderen Werken kenntlich gemacht habe. Die Arbeit ist weder von mir noch von einer anderen Person an der Universität Passau oder an einer anderen Hochschule zur Erlangung eines akademischen Grades bereits eingereicht worden.

Passau, 15. September 2020

Danksagung

Bedanken möchte ich mich bei diversen Kommilitonen und Mitbewohnern für anregende Diskussionen und Hinweise. Besonderer Dank gilt meinem Betreuer Prof. Dr. Rutter für außerordentlich viel Engagement und Feedback, das wesentlich zum Gelingen dieser Arbeit beigetragen hat.

Zusammenfassung

Wir betrachten das so genannte *Tangle-Height Minimization Problem* für einfache Listen, welches wie folgt definiert ist. Für n y-monotone Stränge bestimmt ein Tangle die Anordnung dieser auf mehreren horizontalen Schichten, sodass sich zwei benachbarte Schichten nur durch Swaps (Vertauschungen) zweier benachbarter Stränge unterscheiden. Dabei ist pro Schicht jeder Strang in maximal einem Swap enthalten. In dieser Arbeit wird ein Spezialfall des Problems betrachtet, nämlich solche Tangles, in denen jeder Swap maximal einmal vorkommt. Wir können diese entweder zu einer gegebenen Liste von Swaps bilden (mit Einschränkungen) oder zu einer Start- und Schlusspermutation, sodass die Stränge zu Beginn wie in der Startpermutation und am Ende des Tangles wie in der Schlusspermutation angeordnet sind. Das Problem besteht nun darin, die Anzahl an Schichten des Tangles zu minimieren, indem die Reihenfolge der Swaps ideal gewählt wird. Wir überführen das Problem auf einen Graphen, untersuchen dessen Struktur und präsentieren drei Algorithmen zum effizienteren Finden minimaler Tangles. Wir verbessern durch diese, aus anderen Artikeln bekannte Algorithmen und vergleichen die Laufzeiten experimentell.

Inhaltsverzeichnis

1. Einleitung

Diese Arbeit findet ihren Ursprung in einer offenen Frage des Artikels von Firman et al. [FKR$^+$19]. Wir untersuchen sogenannte Tangles, welche aus n y-monotonen Strängen bestehen, die in verschiedenen Schichten mit benachbarten Strängen vertauscht werden können, wie man in Abbildung 1.1 erkennen kann. Zu Beginn sind die Stränge in einer beliebigen Anordnung (meist von links nach rechts durchnummeriert) gegeben und werden durch sogenannte Swaps – also Vertauschung zweier benachbarter Stränge – in eine neue Reihenfolge gebracht. Über diese gegebenen Anordnungen am Anfang und Ende des Tangles lässt sich bereits eindeutig die Menge an Swaps herleiten, die im Tangle vorkommen müssen, wie Wang [Wan91] schon gezeigt hat. Indem man die Reihenfolge der Swaps im Tangle variiert, kann die Anzahl an Schichten verändert werden. Wir suchen in dieser Arbeit einen Tangle mit minimaler Anzahl an Schichten, hier als Höhe bezeichnet. In der Arbeit von Firman et al. [FKR$^+$19] werden allgemeinere Tangles behandelt, für die eine Liste an Swaps gegeben ist, welche genauso oft wie dort beschrieben durchgeführt werden müssen. Die Autoren zeigen, dass das Finden eines solchen minimalen Tangles NP-schwer ist und stoßen dabei auf die Frage, ob dies für die Höhenminimierung einfacher Tangles auch zutrifft. Wir betrachten in dieser Arbeit also Tangles auf einfachen Listen, die jeden Swap maximal einmal enthalten und so, wie oben erwähnt, direkt aus einer gegebenen Permutation berechnet werden können. Im Allgemeinen wird dieses Problem als *Tangle-Height Minimization Problem* bezeichnet.
Firman et al. [FKR$^+$19] zeigen, dass das Problem für einfache Listen in $\mathcal{O}(\phi^n n!)$ Zeit gelöst werden kann, wobei $\phi \approx 1,618$ den goldenen Schnitt und n die Anzahl an Strängen bezeichnet. Sie lösen einfache Listen, indem sie einen Graphen mit allen möglichen Zwischenanordnungen der Stränge aufbauen, wobei ein Knoten – also eine Anordnung – eine Kante zum Nächsten hat, wenn sich die Anordnungen nur durch Nachbarvertauschungen unterscheiden. Den kleinsten Tangle finden sie dann, indem sie mit Breitensuche den kürzesten Pfad von der Start- zur Zielpermutation suchen. Sie verbessern für allgemeine Listen den von Olszewski et al. [OMK$^+$18] gefundenen Algorithmus. Olszewski et al. beschäftigen sich in ihrer Arbeit mit der Visualisierung sogenannter *chaotic attractors*, welche durch Tangles dargestellt werden können. In einer während der Bearbeitung des Themas erschienenen Arbeit zeigen Firman et al. [FFK$^+$20] nun sogar, dass es bereits NP-schwer ist, herauszufinden ob zu einer beliebigen Liste ein Tangle existiert.
Wang [Wan91] betrachtet dasselbe Problem, um das Design des Channel-Routings auf Leiterplatten und VLSI-Chips zu verbessern. Sie findet einen parallelen Bubblesort-Algorithmus, welcher in einer Laufzeit von $\mathcal{O}(n^2)$ Tangles zu einer gegebenen Permutation der Stränge bildet und Lösungen mit maximal einer Schicht zu hoch liefert. Sie zeigt außerdem, dass

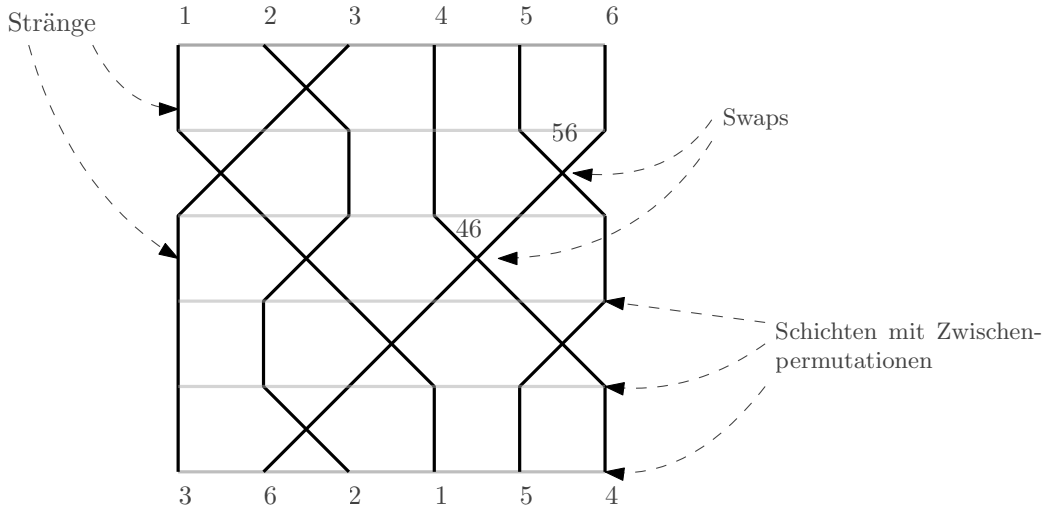

Abbildung 1.1: Ein Tangle zur Illustration des allgemeinen Problems. Der Tangle hier besitzt insgesamt 6 Schichten. Oben sehen wir die Startpermutation 123456 und unten die Zielpermutation 362154. Die Swap-Liste ist hier $L = \{(2,3),(1,3),(1,2),(1,6),(2,6),(5,6),(4,6),(4,5)\}$. Beachte, dass die Pfeile nur exemplarisch einzelne Elemente markieren, der Rest wurde zur Übersichtlichkeit weggelassen.

die Höhe eines minimalen Tangles maximal $n+1$ beträgt. Eine Laufzeit von $\mathcal{O}(n^2)$ scheint für dieses Problem auch minimal zu sein, da es für n Stränge bis zu $n(n-1)/2 \in \mathcal{O}(n^2)$ viele Swaps geben kann, welche alle in den Tangle geordnet werden müssen.
In Kapitel 2 führen wir die betrachteten Tangles und die für die späteren Ausführungen benötigten Begriffe formal ein und finden mit dem Satz 2.4 eine wichtige Erkenntnis zur Struktur der Tangles, die es uns später erlaubt, den Algorithmus von Firman et al. zu verbessern. Wir übertragen dann in Kapitel 3 das Problem auf einen Graphen und auf das Finden einer bestimmten Orientierung. Diese Graphen werden wir als Swapgraphen bezeichnen. Wir zeigen, dass aus einer azyklischen Orientierung des Swapgraphen unter Beachtung gewisser Regeln für das Richten der Kanten in polynomieller Zeit ein eindeutiger Tangle berechnet werden kann und dass das Finden einer Orientierung mit minimalem längsten Pfad einen minimalen Tangle liefert. Wir zeigen in Abschnitt 3.3, von welcher Struktur zyklische Orientierungen im Swapgraphen sind, wenn die Richtung der Kanten den vorher bestimmten Regeln folgt. Damit können Zyklen einfacher vermieden und hierdurch schneller Orientierungen gefunden werden, welche einen Tangle erzeugen. In Kapitel 4 stellen wir drei Algorithmen vor, die auf den vorher gefundenen Ergebnissen beruhen. In ersten Abschnitt 4.1 nutzen wir den vorher eingeführten Swapgraphen und finden eine Orientierung, die uns einen minimalen Tangle liefert, indem wir ein *Integer Linear Program* verwenden. Dieser Algorithmus läuft bereits schneller als der von Firman et al. [FKR$^+$19] gefundene Algorithmus, jedoch überbieten wir diesen in Abschnitt 4.2 nochmal deutlich und beweisen, dass die Laufzeit des Algorithmus in $\mathcal{O}(\psi^n n!)$ mit $\psi \approx 1,325$ liegt. Wir geben eine Methode zur Implementierung des Algorithmus an, welche nicht den ganzen Graphen überprüft, sondern einen DAG (directed azyklic graph/gerichteter azyklischer Graph) zweigweise absucht und abbricht, wenn die Höhe des von Wang gefundenen Tangles unterboten wurde. Dies bringt weitere deutliche Verbesserungen für das praktische Finden minimaler Tangles.
Im letzten Abschnitt 4.3 zeigen wir noch einen Algorithmus, der in einer Laufzeit von $\mathcal{O}(n^2)$ läuft und durch Priorisierung einzelner Stränge sehr häufig optimale Ergebnisse berechnet, jedoch nicht immer korrekt ist. In der Praxis liefert dieser öfter als Wangs Algorithmus optimale Ergebnisse, ist allerdings auf Grund höherer Komplexität signifikant langsamer.

Auch gibt es keine Eingrenzung für die Höhe der Lösungen. Es ergibt sich weiter empirisch, dass die Wahrscheinlichkeit, dass Wangs Algorithmus zu hoch ist, unabhängig davon ist, ob dies auch für unseren Algorithmus zutrifft, sodass beim Verwenden beider Algorithmen bei unseren Tests nur zu etwa 3% falsche Tangles berechnet werden, während Wang alleine zu etwa 37% falsch liegt (bei bis zu $n = 31$ Strängen).

2. Terminologie und Problemstellung

Dieser Abschnitt dient der Einführung der Terminologie und als Referenzpunkt für die späteren Ausführungen. Es kann nützlich sein, sich zuerst Einleitung und Abbildung 1.1 anzusehen, um eine Intuition für die Begriffe und die Thematik zu bekommen.

2.1 Permutationen und Listen

Eine Permutation ist eine bijektive Abbildung der Menge $[n] := \{1, \dots, n\}$ auf sich selbst, wobei ein Element in $[n]$ hier als *Strang* bezeichnet wird. Die Menge S_n beinhaltet alle Permutationen auf $[n]$ und $(S_n, \circ)$ bildet die symmetrische Gruppe, deren neutrales Element die Identität id_n ist und die Verknüpfung $(\pi_1 \circ \pi_2)(x) = \pi_1(\pi_2(x))$ die klassische Verkettung der Funktionen darstellt (die Verknüpfung $\circ$ wird auch manchmal weggelassen). Wir schreiben eine Permutation π als die Sequenz von Zahlen $\pi^{-1}(1)\pi^{-1}(2)\dots\pi^{-1}(n)$, also die resultierende Ordnung der Zahlen nach Anwendung von π auf $1, \dots, n$. Zum Beispiel $\pi \in S_4$ mit $\pi(1) = 3, \pi(2) = 4, \pi(3) = 2, \pi(4) = 1$ ergibt 4312.
Zu einer Permutation π definieren wir auf $[n]$ die Ordnung $a <_\pi b \Leftrightarrow \pi^{-1}(a) < \pi^{-1}(b)$ für $a, b \in [n]$. Zwei Elemente $a, b \in [n]$ sind *benachbart* in $\pi \in S_n$, falls für alle zu a und b verschiedenen $c \in [n]$ gilt, dass $c <_\pi a <_\pi b$ oder $a <_\pi b <_\pi c$ sind (mit $a < b$ ohne Einschränkung).
Die Menge $S_{n,2} := \{\pi \mid \pi \circ \pi = id_n\} \subseteq S_n$ sei diejenige, die alle Permutationen enthält, die ihr eigenes Inverses bilden (auch Elemente der Ordnung 2 genannt). Für $i, j \in [n]$ wird eine Permutation, die nur die Stränge i und j tauscht, auch Swap genannt und als ij notiert. Eine Menge S an Swaps heißt *disjunkt*, wenn jeder Strang in $[n]$ in maximal einem Swap von S enthalten ist. Ein Swap ist benachbart in π, wenn die Stränge i, j in π benachbart sind.
Firman et al. entdeckten für die Menge $S_{n,2}$ und disjunkten Swapmengen einen interessanten Zusammenhang.

Lemma 2.1 (nach [FKR+19])**.** *Zu jeder Permutation $\varepsilon \in S_{n,2}$ existiert eine eindeutige disjunkte Menge $S(\varepsilon)$ an Swaps, sodass das Produkt $\Pi_{\pi \in S(\varepsilon)}\pi = \varepsilon$ ist (und die Reihenfolge des Produkts ist egal).*
Außerdem gilt für jede disjunkte Menge S von Swaps, dass $\Pi_{\pi \in S(\varepsilon)}\pi \in S_{n,2}$ ist.

Eine disjunkte Menge an Swaps lässt sich also als Permutation in $S_{n,2}$ auffassen und umgekehrt. Wie wir später sehen werden, kann man auch jede Permutation (auch die, die

nicht der Ordnung 2 sind) als Menge von Swaps auffassen, die allerdings nicht disjunkt sein muss. Dazu definieren wir Listen von Swaps und zwei Funktionen, die aus einer Permutation eine Menge an Swaps bilden und, falls möglich, auch umgekehrt aus Swaps die zugehörige Permutation bilden.
Eine *(Swap-) Liste* $L = (l_{ij})$ der *Größe* n ist eine symmetrische $n \times n$-Matrix mit Nullen auf der Diagonalen und allen Einträgen null oder eins (im Artikel von Firman et al. [FKR$^+$19] werden diese als einfache Listen bezeichnet). Eine *Subliste* $L' \subseteq L$ ist eine Liste der gleichen Größe mit $l'_{ij} \leq l_{ij}$ in allen Einträgen.
Zu einer Permutation π ist die Liste $L(\pi) = (l_{ij})$ mit $0 \leq i < j \leq n$ diejenige, für die gilt: $l_{ij} = 0$ für $\pi(i) < \pi(j)$, und $l_{ij} = 1$ sonst.
Wir definieren die Funktion:

$$\pi L : [n] \to [n], i \mapsto \pi(i) + |\{j \mid \pi(i) < \pi(j) \leq n, l_{ij} = 1\}| - |\{j \mid 1 \leq \pi(j) < \pi(i), l_{ij} = 1\}|$$

Für jeden Strang $i \in [n]$ ist $\pi L(i)$ bildlich gesprochen die Position des Stranges, nachdem jeder Swap in L durchgeführt wurde. Eine Liste L ist *π-konsistent*, wenn $\pi L \in S_n$ ist, wenn also πL eine Permutation auf $[n]$ bildet.

Lemma 2.2 ([FKR$^+$19])**.** *Sei $\pi \in S_n$ eine beliebige Permutation, dann gilt:* $\mathrm{id}_n L(\pi) = \pi$.

Zu jeder Permutation können wir also eine Liste bilden und aus jeder id_n-konsistenten Liste wiederum eine Permutation.

2.2 Tangles

Eine Permutation $\pi \in S_n$ *unterstützt* eine Permutation $\varepsilon \in S_{n,2}$, wenn jeder Swap $ij \in S(\varepsilon)$ in π benachbart ist. Die Permutation 1234 unterstützt zum Beispiel die Swaps 12 und 34, aber nicht den Swap 13.
Zwei Permutationen π, σ sind *adjazent*, wenn es eine Permutation $\varepsilon \in S_{n,2}$ gibt, sodass π die Permutation ε unterstützt und $\sigma = \pi\varepsilon$ gilt. Dann gilt außerdem $\sigma\varepsilon = \pi\varepsilon\varepsilon = \pi$ und da offensichtlich die Ausführung eines Swaps ab einer in π benachbarten Permutation wieder eine Permutation ergibt, in der a, b benachbart sind, unterstützt auch σ wiederum ε (vgl. die Notation von Firman et al. [FKR$^+$19]).
Ein *Tangle* T der Höhe h ist eine Folge an konsekutiven Permutationen $(\pi_1, \ldots, \pi_h)$, sodass alle π_i, π_{i+1} adjazent sind und außerdem für $i \neq j$ die Formel $S(\pi_{i-1}^{-1}\pi_i) \cap S(\pi_{j-1}^{-1}\pi_j) = \emptyset$ gilt, sprich jeder Swap höchstens einmal vorkommt.
Zum Tangle ist dann die *Swapfolge* $A_T := (\varepsilon_1, \ldots, \varepsilon_{h-1})$ mit $\varepsilon_i = S(\pi_i^{-1}\pi_{i+1}), 1 \leq i < h-1$, die Folge von Mengen an Swaps, sodass $\pi_{i+1} = \pi_i \circ \varepsilon_i$ gilt. Zu einem Swap ab, der im Tangle enthalten ist, sei $\varepsilon(ab)$ dasjenige ε_i, das ab enthält. Wir sagen *ε_i steht vor* einem ε_j, wenn $i < j$ gilt. Entsprechend *steht* ein Swap *ab vor* einem Swap *cd*, wenn $\varepsilon(ab)$ vor $\varepsilon(cd)$ steht. Ein Tangle erzeugt dann durch eine Reihe an legalen Swaps aus einer *Startpermutation* π_1 die *Zielpermutation* π_h. Im Allgemeinen soll die Startpermutation die Identität sein, $\pi_1 = \mathrm{id}_n$. Die Höhe eines Tangles $T = (\pi_1, \ldots, \pi_h)$ ist $h(T) = h$. Zudem definieren wir die Swap-Liste $L(T) = (l_{ij})$, sodass $l_{ij} = 1 \Leftrightarrow ij$ ist ein Swap in T, $l_{ij} = 0$ sonst. Jeder Tangle mit einer anderen Startpermutation als id_n lässt sich zu einer mit der Identität beginnenden umformen, indem man die Stränge nur entsprechend umbenennt, sodass im Folgenden meist id_n als Startpermutation verwendet wird.
Eine Liste L ist *lösbar auf der Ordnung* π (oder nur *lösbar* für die Startordnung $\pi = \mathrm{id}_n$), wenn es einen Tangle T mit Startordnung π gibt, sodass $L(T) = L$ gilt; wir sagen T *realisiert* L. Eine Liste ist genau dann lösbar auf π, wenn sie π-konsistent ist [FKR$^+$19]. Da eine Liste in gewissem Sinne äquivalent zu einer Permutation ist (wie oben beschrieben), sagen wir auch, ein Tangle *realisiert eine Permutation* π, wenn $L(T) = L(\pi)$. Die *optimale*

$$L := \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

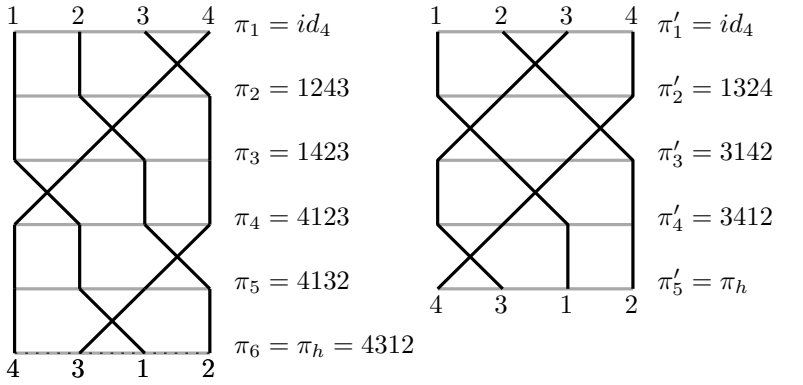

Abbildung 2.1: Für die Liste L links gibt es mehrere Möglichkeiten, einen Tangle zu konstruieren. Rechts davon sieht man zwei korrekte Tangles auf L unterschiedlicher Höhe. Der rechte ist von optimaler Höhe. Die Swapfolge zum rechten Tangle lautet beispielsweise $A_T = \{\varepsilon_1 = \{23\}, \varepsilon_2 = \{13, 24\}, \varepsilon_3 = \{14\}, \varepsilon_4 = \{34\}\}$ und $\varepsilon(13) = \varepsilon_2$.

Höhe $h(L)$ einer Liste L, ist die minimale Höhe aller möglichen Tangles die L realisieren, $h(L) = \min\{h(T) \mid T \text{ realisiert } L\}$. Das Problem besteht nun darin einen *optimalen Tangle* T so zu finden, dass $h(T) = h(L)$ gilt.
Als Beispiel betrachte man Abbildung 2.1, in der zwei Tangles verschiedener Höhe zu einer gegebenen Liste L dargestellt sind.

Lemma 2.3. *Sei $T := (\pi_1, \ldots, \pi_h)$ ein Tangle. Jede Subfolge $T' := (\pi_l, \pi_{l+1}, \ldots, \pi_r)$ mit $1 \leq l < r \leq h$ ist wieder ein Tangle mit Startpermutation π_l und Zielpermutation π_r.*

Beweis. Zu zeigen ist, dass jedes $\pi_i, \pi_{i+1} \in T'$ adjazent ist und jeder Swap nur einmal vorkommt. Das folgt direkt daraus, dass sie adjazent in T waren und kein weiterer Swap hinzukommt. □

Im folgenden Satz wird klar dass, um einen optimalen Tangle zu bilden kein Swap „freigelassen“ werden muss. Wann immer in einer Schicht ein weiterer Swap hinzugefügt werden kann, sollte das für einen optimalen Tangle auch geschehen.

Satz 2.4. *Sei $T := (\pi_1, \ldots, \pi_h)$ ein Tangle mit Swapfolge $\varepsilon_T := (\varepsilon_1, \ldots, \varepsilon_{h-1})$, wobei zwei beliebige Stränge a, b benachbart in π_1 sind und $S(\varepsilon_1) \cup \{ab\}$ disjunkt ist (also weder der Strang a noch b an einem Swap in ε_1 teilnimmt). Weiter sei der Swap $ab \in S(\varepsilon_{h-1})$ enthalten. Dann gibt es einen Tangle $T' = (\pi'_1, \ldots, \pi'_h)$ von kleinerer oder gleicher Höhe mit Swapfolge $\varepsilon_{T'} = (\varepsilon'_1, \ldots, \varepsilon'_{h-1})$, für den $ab \in S(\varepsilon'_1)$ und $L(T) = L(T')$ gilt.*

Beweis. Sei also T wie im Satz beschrieben. Wir bilden nun T' wie folgt: Die Anfangs- und Zielpermutation in T bleibt gleich: $\pi'_1 = \pi_1$ und $\pi'_h = \pi_h$ und für jede $2 \leq i \leq h-1$ ist die Permutation $\pi'_i = \pi_i \circ ab$. Klar ist, dass eine Umbenennung der Stränge in einem Tangle einen korrekten Tangle hinterlässt und was hier letztlich geschieht, ist genau eine Umbenennung von Strang a nach b und andersrum, also ist die resultierende Subfolge $(\pi'_2, \ldots, \pi'_{h-1})$ ein Tangle nach Lemma 2.3.
Die Permutationen π'_1, π'_2 sind adjazent, denn es gilt: $\pi'_2 = \pi_2 \circ ab = \pi_1 \circ \varepsilon \circ ab$ und $\varepsilon_1 \circ ab$ ist nach Voraussetzung in $S_{n,2}$, da $S(\varepsilon_1 \circ ab)$ disjunkt. Die selbe Argumentation gilt auch für $\pi'_h = \pi_h = \pi_{h-1} \circ \varepsilon_{h-1} = \pi'_{h-1} \circ ab \circ \varepsilon_{h-1}$, da auch hier wegen des Swaps ab in ε_{h-1} die Stränge a und b in den beiden Permutationen benachbart sein müssen. Beachte, dass $ab \circ ab = \mathrm{id}_n$ ist und somit die Swapmenge $S(\varepsilon_{h-1} \circ ab)$ den Swap ab nicht mehr

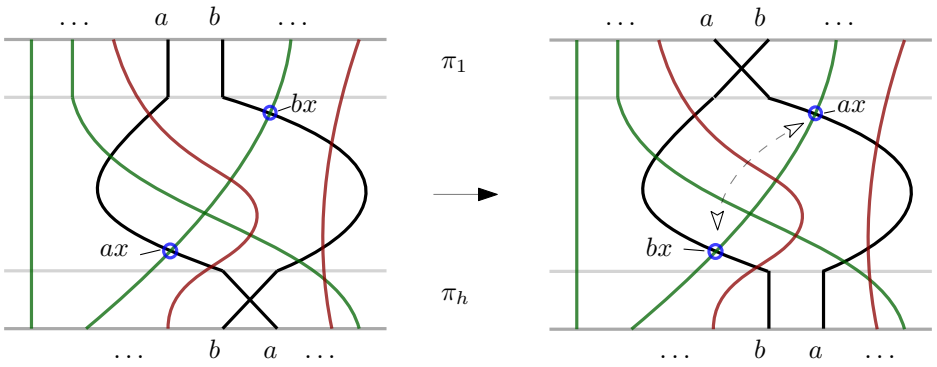

Abbildung 2.2: Die schwarzen Stränge a, b swappen links erst später. Zieht man den Swap hoch, so ändert sich für alle anderen (grünen) Stränge nur die Reihenfolge der Swaps; an der Stelle an dem ein Swap mit a passierte, ist nun einer mit b und umgekehrt. Andere Stränge (rot), sind nicht möglich, da ja jeder Swap im Tangle nur einmal vorkommen darf; insbesondere können in der ersten und letzten Ebene keine Stränge zwischen a und b stehen.

enthält.
Es bleibt noch zu zeigen, dass die Listen $L(T), L(T')$ gleich sind. Sei dazu ax ein Swap in L. Wir zeigen nun, dass dann auch $bx \in L$ sein muss. Sei ohne Beschränkung der Allgemeinheit $x <_{\pi_1} a <_{\pi_1} b$ (dazwischen kann der Strang x nicht stehen, da a, b benachbart sind; wegen Spiegelsymmetrie ist der andere Fall analog). Irgendwo zwischen π_1 und π_h swapped also xa. Also gilt nach dem Swap ax die Ordnung $a < x < b$. Da allerdings a, b in der Schlusspermutation wieder benachbart sein müssen und x nur dann nicht mehr zwischen a und b stehen kann, wenn es auch mit einer der beiden Parteien swapped und ax nur einmal swappen darf, muss auch $bx \in L$ sein (zur Verdeutlichung betrachte Abbildung 2.2). Also gilt in diesem Fall: $ax \in L(T) \Leftrightarrow bx \in L(T)$ (die Rückrichtung gilt aufgrund der Symmetrie). Klar ist, dass jeder Swap in $L(T)$, der weder Strang a noch b beinhaltet, auch in $L(T')$ sein muss, da ja nur a und b verändert wurden. Durch die vorherige Beobachtung ist nun klar, dass durch die Umbenennung in T' also weder Swaps hinzukamen, noch entfernt wurden. Insbesondere ist der Swap ab immer noch genau einmal vertreten. □

Dieser Satz zeigt die Aussage nur für Swaps, die ganz am Anfang nicht gemacht, jedoch zum Schluss durchgeführt wurden. Wie aber vorher gezeigt ist jeder Sub-Tangle wieder ein Tangle, findet man also irgendwo in einem Tangle die selbe Situation vor, kann dieses Lemma dort auch angewandt werden. Nun wollen wir zeigen, dass man in keine „Sackgasse“ laufen kann – es ist also egal, in welcher Reihenfolge man Swaps durchführt, man kommt immer zu der Zielpermutation.

Lemma 2.5. *Sei $T' = (\pi_1, \ldots, \pi_r)$ ein Tangle zu einer beliebigen auf π_1 lösbaren Subliste $L' \subseteq L$, wobei L eine auf π_1 lösbare Liste ist mit $\pi_1 L = \pi_h$. Dann gilt für jeden Tangle $T'' = (\pi_r, \ldots, \pi_h)$ auf der Liste L'', dass $T = (\pi_1, ..., \pi_h)$ ein Tangle zu L ist und es gilt $L = L' \dot\cup L''$.*

Beweis. Klar ist, dass es einen Tangle T'' gibt, da es nach [Wan91] zu allen $\pi, \sigma \in S_n$ eine eindeutige lösbare Liste gibt, die mit π startet und mit σ endet. Es bleibt also zu zeigen, dass jeder Swap $ij \in L$ entweder in L' oder in L'' enthalten ist. Sei also $ij \in L$ mit $i <_{\pi_1} j$ und damit $j <_{\pi_h} i$ in der Schlusspermutation.

1. Fall $ij \in L'$: Dann gilt $j <_{\pi_r} i$. Wäre ij nun auch in L'' enthalten, würde sich in T'' die Ordnung wieder umkehren, sodass $i <_{\pi_h} j$ gelten müsste. Dies ist ein Widerspruch zur Annahme, dass $ij \in L$ und damit $j <_{\pi_h i}$ gilt.
2. Fall $ij \notin L'$: Dann gilt $i <_{\pi_r} j$ in der Permutation π_r. Da sich die Ordnung von π_r zu π_h für ij irgendwann umkehren muss wegen $ij \in L$, ist also $ij \in L''$.

In jedem der Fälle gilt also, dass jedes Element in nur einer Liste enthalten ist. Es gilt insbesondere, dass kein Swap $ij \in L' \cup L''$ mit $ij \notin L$ existieren kann, denn es ist L' eine Subliste zu L und wäre $ij \in L''$, könnte wegen der Eindeutigkeit einer Swapmenge zu einer Permutation nicht π_h herauskommen. Damit ist die Behauptung gezeigt. □

Mit dem Satz 2.4 und Lemma 2.5 stellt sich also heraus, dass man einen Tangle zu einer Liste bilden kann, indem man einfach immer weiter Swaps durchführt, bis man am Ziel ist. Insbesondere sollte man nie einen möglichen Swap auslassen, wenn man die optimale Höhe erreichen will.

3. Swapgraphen

Der *Swapgraph* $G = (V, E)$ zu einer Swapliste L ist ein Graph, dessen Knoten Swaps in der Liste L sind und in dem zwei Knoten eine Kante besitzen, wenn sie sich einen gleichen Strang teilen: $V := \{ij \mid l_{ij} = 1\}$ und $E := \{(ab, cd) \mid (a = c \vee a = d \vee b = c \vee b = d) \wedge ab \neq cd\}$. Dabei ist eine Kante (ab, ac) von ab nach ac *gerichtet*, die Ordnung spielt also eine Rolle. Da in E jeweils beide Richtungen enthalten sind, sprechen wir von *ungerichteten Kanten*. Zu jedem Tangle können wir nun einen Swapgraphen erstellen. In gewisser Weise sind in jedem Tangle die einzelnen Swaps voneinander abhängig; zum Beispiel kann im entsprechenden Tangle 13 nicht von Anfang an geswapped werden, da die Stränge 1 und 3 nicht benachbart sind. Zwangsläufig muss also 23 oder 12 vorher passieren (anschaulich gesprochen). Insbesondere können dann Abhängigkeiten nur zwischen Swaps mit gleichen Strängen entstehen, wie es im Graphen realisiert ist. Die Idee ist nun, je eine Orientierung für jeden Swapgraphen zu finden, die dem Tangle entspricht und die diese Abhängigkeiten möglichst gut einfängt.

3.1 Die geordnete Orientierung

Eine *Orientierung* $O_G \subseteq E$ zu einem Swapgraphen $G = (V, E)$ ist eine Teilmenge der Kanten, sodass jede Kante in O_G gerichtet ist, also für alle $(ab, ac) \in O_G$ gilt $(ac, ab) \notin O_G$. Außerdem soll für jede Kante $(ab, ac) \in E$ gelten, dass $(ab, ac) \in O_G$ oder $(ac, ab) \in O_G$ ist. Ein *Zyklus* $Z := (a_1b_1, \ldots, a_kb_k)$ in einer Orientierung O_G zum Swapgraphen G ist eine Folge an Knoten im Graphen, wobei für alle $a_ib_i, a_{i+1}b_{i+1}$ gilt, dass $(a_ib_i, a_{i+1}b_{i+1}) \in O_G$ ist, wobei der Index $i \in \mathbb{Z}_k$ ist, also $a_{k+1} = a_1$ gilt. Eine Orientierung O_G heißt dann *zyklisch*, falls sie mindestens einen Zyklus besitzt.

Definition 3.1. *Zu einem beliebigen Tangle T mit der Swapfolge $A_T = (\varepsilon_1, \ldots, \varepsilon_{h-1})$ auf der Liste L und dem zugehörigen Swapgraphen $G = (V, E)$ definieren wir die* Tangle-Orientierung *als $O(T) := \{(ab, ac) \in E \mid ab \in \varepsilon_i, ac \in \varepsilon_j, 1 \leq i < j \leq h - 1\} \subseteq E$.*

Die Tangle-Orientierung $O(T)$ ist tatsächlich eine Orientierung zu G. Zum einen sind zu jeder Kante $(u, v) \in O(T)$ die beiden Knoten u, v nicht im gleichen ε_i, weil alle ε_i in sich disjunkt sind. Weiter ist auch jede Kante in $O(T)$ gerichtet, da alle ε_i zueinander paarweise disjunkt sind und somit kein Swap sowohl vor als auch nach einem anderen sein kann. Zudem ist die Orientierung azyklisch, denn angenommen es gäbe einen Zyklus $(a_1b_1, a_2b_2, \ldots, a_kb_k)$

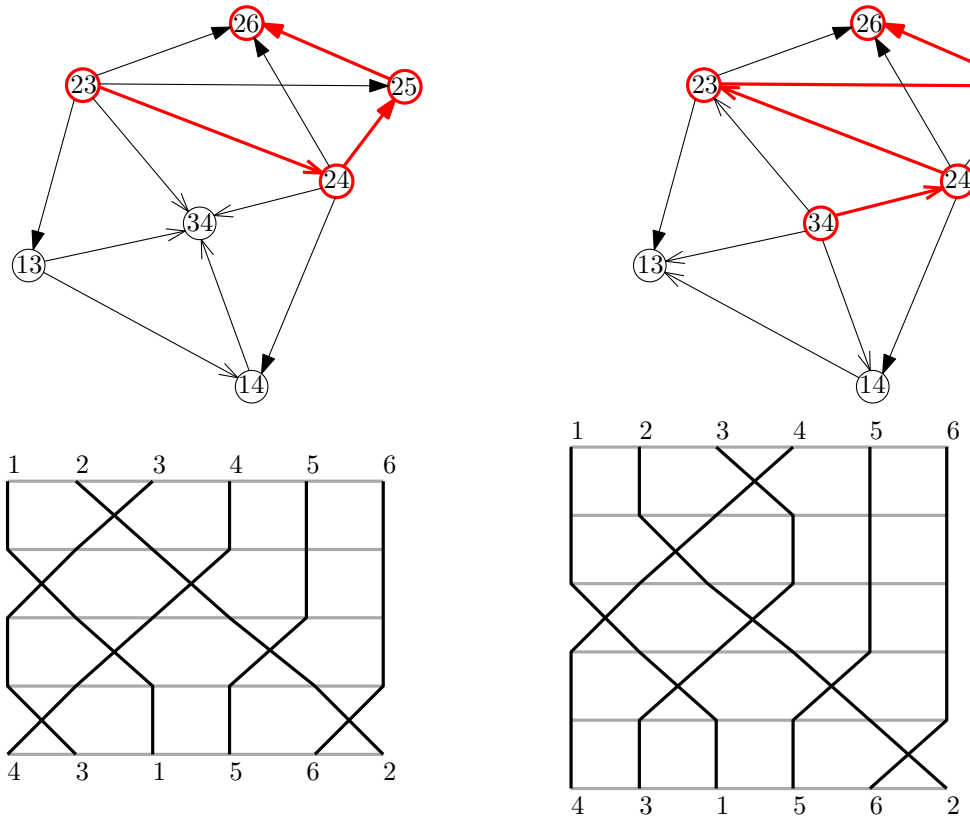

Abbildung 3.1: Zwei verschiedene, aber korrekte Tangles, die dieselbe Liste realisieren. Die gefüllten Pfeile stellen starke Abhängigkeiten dar, die ungefüllten eine mögliche Lösung für die schwach gerichteten in einem Swap-Tripel. Der längste Pfad (rot) ist links 4 und rechts 5. Die Höhe der Tangles ist entsprechend.

dann gibt es insbesondere einen im Tangle „tiefsten“ Swap a_ib_i, sodass für alle von i verschiedenen $1 \leq j \leq k$ gilt, dass die Swapmenge ε_l mit $a_jb_j \in \varepsilon_l$ vor der Swapmenge ε_m mit $a_ib_i \in \varepsilon_m$ steht, also $l < m$. Nun zeigt a_ib_i nach $a_{i+1}b_{i+1}$, was bedeutet, dass es ε_z gibt, sodass $a_{i+1}b_{i+1} \in \varepsilon_z$ mit $m < z$ gilt, was einen Widerspruch zu der Annahme ergibt, dass a_ib_i das tiefste Element ist.
Man betrachte beispielsweise die Abbildung 3.1, bei der zur gleichen Liste zwei verschiedene Tangles und Orientierungen zu sehen sind. In rot ist außerdem jeweils der längste Pfad markiert, der sich sehr ähnlich zur Höhe des Tangles verhält. Die Länge eines Pfades wird bei uns anhand der enthaltenen Knoten gezählt. Interessanterweise ergibt sich folgendes Lemma bezüglich der Höhe.

Lemma 3.2. *Sei T ein Tangle der Höhe $h(T) = h$ und l die Länge des längsten Pfades in der Orientierung $O(T)$. Dann gilt $l + 1 \leq h$ und es gibt auch strikt kleinere Fälle.*

Beweis. Aus der Konstruktionsvorschrift für die Richtungen ist klar, dass maximal so viele Knoten hintereinander gerichtet werden können, wie Elemente in $A_T = (\varepsilon_1, \ldots, \varepsilon_{h-1})$ sind, denn jeder Knoten a wird nach b gerichtet, wenn $a \in \varepsilon_i, b \in \varepsilon_j, i < j$ ist und sie einen Strang teilen. Zu jedem ε_i kann also höchstens ein Knoten in einem Pfad sein, also maximal $l \leq h - 1$ oder eben $l + 1 \leq h$.
Auf der anderen Seite betrachte man den Tangle $T' = (1234, 2134, 2143, 2413)$ in Abbildung 3.2. Wenn man den geordneten Graphen dazu bildet, erhält man einen längsten Pfad der Länge 2, was strikt kleiner ist als $h - 1 = 3$. □

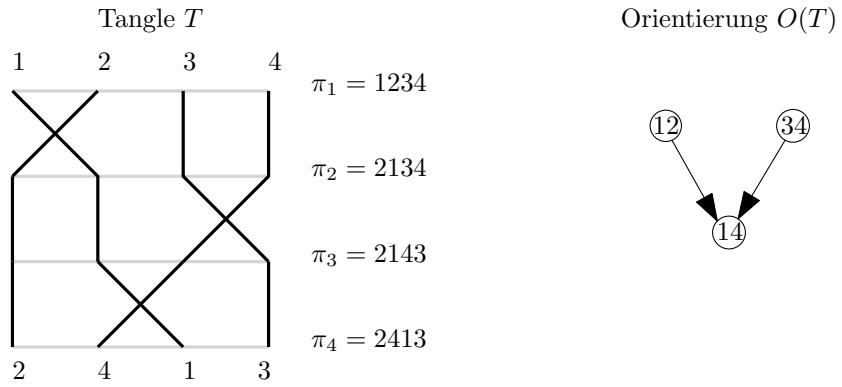

Abbildung 3.2: Links ein nicht höhen-minimaler Tangle, der durch Umformen in einen Tangle kleiner wird, also die Höhe des DAG plus eins ist kleiner als die Tanglehöhe.

Nun wollen wir in den Orientierungen Regeln oder Muster finden, nach denen sie aufgebaut sind, sodass man aus einer Orientierung, die wiederum nach den Regeln aufgebaut ist, einen Tangle bauen kann. Dazu folgende Beobachtung:
Der Graph lässt sich durch zwei wiederkehrende Muster charakterisieren.

1. Eine *Singleton-Clique* ist eine Clique im Graphen, die nur von einem Strang abhängt. Nachdem zwischen je zwei Knoten genau dann eine Kante existiert, wenn sie einen Strang teilen, bildet sich eine Clique zu jedem Strang. Im speziellen Fall soll eine Singleton-Clique $\{ab, ac, ad, \dots\}$ a-Clique heißen, da sie vom Strang a erzeugt wird. Trivialerweise befindet sich jede Kante in genau einer solchen Singleton-Clique.
2. Ein *Swap-Tripel* ist eine Clique aus drei Elementen der Form $\{ab, bc, ac\}$ und wird in diesem speziellen Fall als *abc-Tripel* bezeichnet. Jede Kante kann in maximal einem Swap-Tripel sein, dieser letzte Knoten ist bereits eindeutig zu jeder Kante definiert: Zu einer beliebigen Kante $(ab, bc) \in V$ kann das letzte Element nur ac sein. Ist $ac \in V$, so gehört es zu einem solchen Tripel, ansonsten nicht.

Interessant ist insbesondere, dass die Kanten vollständig durch Singleton-Cliquen überdeckt werden. Mit folgendem Lemma können wir aus der Lösbarkeit einer Liste und gewissen Eigenschaften Informationen über die Richtung von Kanten gewinnen. Insbesondere ist auch die Umkehrung interessant, mit der wir auf die Ordnung einzelner Stränge schließen oder zumindest manche Ordnungen ausschließen können.

Lemma 3.3. *In einer auf π lösbaren Liste L der Größe n mit $a <_\pi b <_\pi c$ für $a, b, c \in [n]$ und $ab, bc \in L$ folgt, dass auch $ac \in L$ ist.*

Beweis. Angenommen also es gäbe eine auf π lösbare Liste L, mit $a <_\pi b <_\pi c$ und $ab, bc \in V$, aber $ac \notin V$. Dann gibt es wegen der Lösbarkeit einen Tangle $T = (\pi_1, \dots, \pi_h)$ und die Swapfolge $A_T = (\varepsilon_1, \dots, \varepsilon_{h-1})$. Sei $\varepsilon_i := \varepsilon(ab)$ und $\varepsilon_j := \varepsilon(bc)$. Da sich die Swaps ab, bc einen Strang teilen, gilt außerdem $i \neq j$. Unterscheide also die beiden Fälle:

1. Fall: $i < j$: Also swapped ab vor bc und folglich gilt beim Swap bc die Ordnung $b <_{\pi_j} a <_{\pi_j} c$ und b, c sind in der Permutation π_j nicht benachbart, da ja $ac \notin V$ und ab nur einmal swappen darf, was einen Widerspruch zur Tangle-Eigenschaft von T ergibt.
2. Fall: $j < i$: Hier swapped also bc vor ab, also gilt die Ordnung $a <_{\pi_i} c <_{\pi_i} b$ beim Swap von ab, wie im ersten Fall. Damit sind a, b nicht benachbart, wenn sie in T geswapped werden, was den selben Widerspruch wie im 1. Fall ergibt.

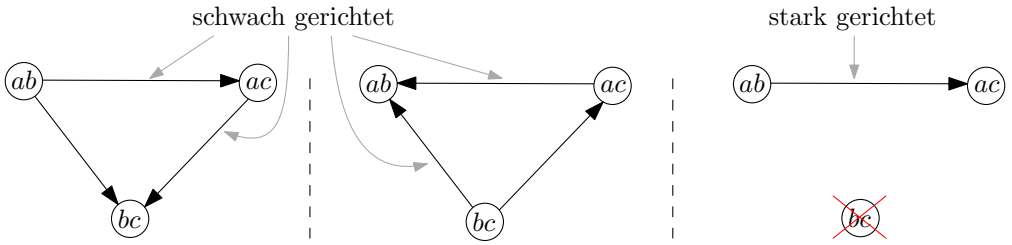

Abbildung 3.3: Die Ordnung sei $a < b < c$. Hier sehen wir die Möglichkeiten, nach den Regeln in Definition 3.4 Kanten zu richten. Für *abc*-Tripel gibt es zwei Möglichkeiten und die Kanten werden dann als schwach gerichtet bezeichnet. Für eine Kante, die in keinem Swap-Tripel ist, gibt es nur eine Möglichkeit und die Kante ist stark gerichtet.

Damit ist keiner der Fälle möglich und die Behauptung ist gezeigt, da alle möglichen Fälle beachtet wurden. □

Aus der obigen Charakterisierung lassen sich nun folgende Regeln ableiten, wie ein Swapgraph gerichtet werden kann.

Definition 3.4. *Eine Orientierung O_G zu einem Swapgraphen $G = (V, E)$, die nach folgenden Regeln vollständig orientiert ist, heißt* geordnet. *Die Ordnung '<' beruht hier auf der Ordnung der Startpermutation π (im Allgemeinen also $\pi = \mathrm{id}_n$).*

1. *Regel: Ist eine Kante (ab, ac) in keinem Swap-Tripel, also ist $bc \notin V$, so ist sie* stark gerichtet *und zwar $(ab, ac) \in O_G$, für $a < b < c$, bzw. umgekehrt für $c < b < a$.*
2. *Regel: Sei $a < b < c$ in einem abc-Tripel. Dann gilt $(ab, ac), (ab, bc), (ac, bc) \in O_G$ oder umgekehrt $(bc, ac), (bc, ab), (ac, ab) \in O_G$. Wir bezeichnen solche Kanten als* schwach gerichtet.

Starke Richtungen sind also für alle geordneten Orientierungen zu einem Graphen G gleich, während es zu jedem Swap-Tripel mit schwach gerichteten Kanten zwei genau entgegengesetzte Möglichkeiten gibt. Betrachte zur Verdeutlichung die Abbildung 3.3. Damit ist das komplette Swap-Tripel bereits eindeutig gerichtet, wenn nur eine einzige enthaltene Kante gerichtet wird. Insgesamt ergibt sich also eine binäre Entscheidung zu jedem Swap-Tripel beim Erstellen einer geordneten Orientierung zu einem Graphen.
In der vorher bereits gesehenen Abbildung 3.1 sieht man also zwei geordnete und azyklische Orientierungen zu einem Schnittgraphen und außerdem jeweils einen zugehörigen Tangle mit gleichartigem Aufbau.

3.2 Geordnete, azyklische Orientierungen und Tangles

In diesem Kapitel wollen wir also Tangles und ihr Verhältnis zu geordneten Orientierungen genauer untersuchen und kommen zum Ergebnis, dass geordnete und azyklische Orientierungen immer einen Tangle induzieren und umgekehrt. Eine Orientierung ist auch immer gleichwertig zu einem Tangle, bis auf eine mögliche Höhenverringerung beim Bilden der Orientierung, wie wir im Laufe des Kapitels sehen werden.
Folgende Eigenschaft azyklischer und geordneter Orientierungen ist wichtig für die Umwandlung von ebendiesen in Tangles. Beachte, dass die Tangles spiegelsymmetrisch bezüglich einer vertikalen Achse sind und sich die Ordnungen im Allgemeinen umdrehen lassen. Gilt also eine Aussage für die Ordnung $a < b < c$ mit $a, b, c \in [n]$ so gilt sie auch für $c < b < a$.

Lemma 3.5. *Im Swapgraph $G = (V, E)$ mit geordneter, azyklischer Orientierung O_G, deren zugrundeliegende Swap-Liste L lösbar auf π ist, ist jede Quelle ac ein benachbarter Swap in π.*

Beweis. Sei ac eine Quelle in der Orientierung O_G. Angenommen ac wäre nicht benachbart in π, dann gibt es ein Element $b \in [n]$ mit $a < b < c$ ($<$ ist hier $<_\pi$). Da L lösbar ist, muss gelten $ab \in L$ und/oder $bc \in L$. Insbesondere gilt nach der Definition des Schnittgraphen, dass aus $ab \in L \Rightarrow ab \in V$ folgt.

1. Fall $ab, bc \in L$: Damit ist $bc \in V$. Dann existiert das abc-Tripel, sodass wegen der Ordnung $a < b < c$ entweder $(ab, ac) \in O_G$ oder $(bc, ac) \in O_G$ gelten muss (vgl. Regel 2 in Definition 3.4). Das ergibt einen Widerspruch zu der Annahme, dass ac eine Quelle ist, also keine nach ac gerichtete Kante existiert.
2. Fall $ab \in L, bc \notin L$: Damit ist $bc \notin V$. Dann ist die Kante $(ab, ac) \in O_G$ stark gerichtet, da $a < b < c$ (vgl. Regel 1 in Definition 3.4), was wieder einen Widerspruch zur Annahme ergibt, dass ac eine Quelle ist.

Der Fall dass $bc \in L$, aber $ab \notin L$ ist, ist symmetrisch zum Fall 1 und damit ist die Fallunterscheidung vollständig. □

Wir erhalten folgendes Lemma für die Tangle-Orientierung $O(T)$. Man beachte, dass die Reihenfolge zweier Stränge a, b sich nur ändern kann, wenn die beiden swappen. Mit $\varepsilon_i := \varepsilon(ab)$ und $a <_{\pi_1} b$ gilt also: $a <_{\pi_j} b$ für $j \leq i$ und $b <_{\pi_j} a$ für $i < j$.

Lemma 3.6. *Sei T ein Tangle, dann ist $O(T)$ azyklisch und geordnet.*

Beweis. Sei $T = (\pi_1, ...\pi_h)$ ein Tangle auf der Liste L. Dass jede Kante gerichtet ist und der Graph azyklisch ist, ist bereits in der Anmerkung unter der Definition 3.1 gezeigt. Zeige also noch, dass jede Kante nach den Regeln in der Definition 3.4 gerichtet ist. Sei $e = (ab, ac) \in O(T)$ eine nach ac gerichtete Kante. Unterscheide folgende Fälle:

1. Fall $bc \notin V$: Dann ist e in keinem Swap-Tripel. Um Regel 1 zu erfüllen, muss also gelten, dass $a < b < c$ oder $c < b < a$ ist. Wir schließen die anderen Fälle aus: Klar ist bereits, dass die Ordnungen $b < a < c, c < a < b$ nach Lemma 3.3 ausgeschlossen werden können, da $bc \notin V$ ist. Angenommen der Graph wäre nicht geordnet; es gelte also $a < c < b$ (der Fall $c < b < a$ ist aufgrund von Symmetrie analog). Da $(ab, ac) \in O(T)$ ist, muss im Tangle ab vor ac sein. Seien also $\varepsilon_i = \varepsilon(ab)$ und $\varepsilon_j = \varepsilon(ac)$ mit $1 \leq i < j < h$. Da alle aufeinanderfolgenden Permutationen adjazent sind, müssen a und c benachbart in π_j sein. Da sich die Ordnung zweier Elemente nur ändern kann, wenn der entsprechende Swap der beiden passiert, bleibt die Startordnung $a < c < b$ erhalten bis zum Schritt i. Also sind a und b in π_i nicht benachbart. Dies ist ein Widerspruch dazu, dass der Swap ab vor ac stattfindet. Damit ist Regel 1 erfüllt, für Kanten in keinem Swap-Tripel.
2. Fall $bc \in V$: Damit existiert das abc-Tripel im Swapgraphen. Wir definieren die Swapmengen als $\varepsilon_i := \varepsilon(ab), \varepsilon_j := \varepsilon(ac), \varepsilon_k := \varepsilon(ac)$, wobei wegen der Richtung von e klar ist, dass $i < j$ gilt. Betrachte die drei verschiedenen Anordnungsmöglichkeiten der Stränge (und die drei symmetrischen Fälle):
   a) Fall $a < b < c$ ($c < b < a$): Die einzig mögliche Anordnung nach Regel 2 ist damit $(ab, ac), (ac, bc), (ab, ac) \in O(T)$, wir zeigen also, dass alle anderen Möglichkeiten auszuschließen sind. Es bleiben drei Möglichkeiten, wann bc swapped, also wo k steht:

i. Fall $k < i < j$: Damit gilt $a <_{\pi_i} c <_{\pi_i} b$ und damit sind a und b nicht benachbart in π_i und könnten folglich nicht in Schritt i geswapped werden, was einen Widerspruch zur Tangle-Eigenschaft ergibt.

ii. Fall $i < k < j$: Damit gilt $b <_{\pi_k} a <_{\pi_k} c$ in Schritt k und diesmal sind b und c nicht benachbart, wenn sie geswapped werden, was wie im vorigen Fall widersprüchlich ist.

Es bleibt also nur noch die Möglichkeit, dass $i < j < k$ gilt und damit ist genau die Regel erfüllt.

b) Fall $a < c < b$ ($b < c < a$): Wegen Regel 2 muss $(bc, ab), (bc, ac), (ab, ac) \in O(T)$ gelten, also $k < i < j$. Wir schließen hier die anderen Möglichkeiten $i < k < j$ und $i < j < k$ sofort aus, da dann a, b nicht benachbart in π_i sind.

c) Fall $b < a < c$ ($c < a < b$): Nach Regel 2 müsste $(ab, bc), (ab, ac), (bc, ac) \in O(T)$ gelten, also $i < k < j$. Wir betrachten wieder die anderen beiden Fälle:

i. Fall $k < i < j$: Offensichtlich kann bc nicht als erstes swappen, da es nicht benachbart in π_1 und damit π_k ist.

ii. Fall $i < j < k$: Hier gilt dann nach dem ersten Swap ab in Schritt i, dass $a <_{\pi_j} b <_{\pi_j} c$ ist. Demnach ist ac nicht benachbart in π_j, was unmöglich ist.

Die Fallunterscheidung ist offensichtlich vollständig. Damit wurde gezeigt, dass der Graph azyklisch und vollständig nach den Regeln 3.4 gerichtet wurde. Die Orientierung $O(T)$ ist damit geordnet. □

Nun wollen wir einen Tangle aus einer azyklischen und geordneten Orientierung bilden. Wir gehen dazu wie folgt vor:

1. Beginne mit der gegebenen Startpermutation $\pi_1 = \pi$.
2. Zu jeder Permutation π_i bilde die Folgepermutation π_{i+1} wie folgt (bis der Restgraph keine Knoten mehr enthält):
   a) Entferne jede Quelle aus dem Graphen mitsamt allen Kanten, die mit ihnen verbunden waren.
   b) Sei S_i die Menge aller entfernten Swaps. Dann ist die gesuchte adjazente Permutation $\pi_{i+1} = \pi_i \circ \Pi_{s \in S_i} s$, also die Ausführung aller entfernten Swaps.

Den hier erzeugten Tangle aus der geordneten, azyklischen Orientierung O bezeichnen wir als $T(O)$.

Proposition 1. *Zu jeder geordneten und azyklischen Orientierung O_G zu einem Graphen G auf der Liste L ist $T(O_G)$ ein Tangle auf L.*

Beweis. Zu Beginn ist die Ordnung auf dem Graphen azyklisch und geordnet mit der Startpermutation π_1 (nach Voraussetzung). Nach jedem Schritt $i \to i + 1$ entsteht wieder ein geordneter und azyklischer Restgraph G' auf der Ordnung π_{i+1} (siehe Anmerkung a weiter unten) und die Permutation π_i ist adjazent zu π_{i+1} (siehe Anmerkung b weiter unten). Schlussendlich terminiert der Algorithmus, denn der Graph ist nach Voraussetzung azyklisch, sodass jeder Knoten irgendwann zur Quelle wird. Also entsteht auch ein endlicher Tangle $T = (\pi_1, \ldots, \pi_h)$ mit jeweils adjazenten Permutationen, der jeden Swap im Graphen genau einmal enthält, damit ist $L(T) = V$. Beachte, dass alle $\pi_i \neq \pi_j$ für $i \neq j$ sind, da alle Swaps in V verschieden sind und nur einmal in eine Swapmenge überführt werden.

Es bleiben also die Beweise für a und b:
(a) *Es entsteht immer ein geordneter und azyklischer Restgraph*: Wir zeigen also, dass weiterhin jede Kante nach den Regeln geordnet sind. Die beiden Regeln in 3.4 beruhen lediglich auf der Ordnung der involvierten Stränge. Für den Fall, dass ab eine Quelle wäre, fällt die Kante (ab, ac) raus, weswegen diese nicht zu beachten ist. Außerdem bleibt der Graph immer azyklisch, da sich keine Kanten verändern und nur welche entfernt werden, was keinen Zyklus verursachen kann.
Sei also $(ab, ac) \in O_G$ eine nach ac gerichtete Kante, wobei ab keine Quelle in G ist. Unterscheide folgende Fälle:

1. Fall $bc \in V, bc$ ist keine Quelle: Dann bleiben alle Kanten in G' zwischen ihnen erhalten, wie in G. Da kein Paar aus a, b, c im Schritt getauscht wird (denn keines davon ist eine Quelle), bleibt auch die Ordnung dieser drei Elemente in π_{i+1} gleich zur Vorherigen und damit bleiben die Richtungen korrekt.

2. Fall $bc \in V, bc$ ist eine Quelle: Da bc eine Quelle ist, sind $(bc, ab), (bc, ac) \in O_G$, genau wie nach Voraussetzung (ab, ac). Wegen Regel 2 muss also vor dem Entfernen der Quelle die Ordnung $a <_{\pi_i} c <_{\pi_i} b$ (oder umgekehrt; analog wegen Spiegelsymmetrie) gegolten haben. Wird nun bc in der neuen Permutation geswapped, ergibt sich die Ordnung $a <_{\pi_{i+1}} b <_{\pi_{i+1}} c$ und damit ist die Kante (ab, ac) nun korrekt nach Regel 1 stark gerichtet. Beachte wieder, dass die Reihenfolge der drei Elemente untereinander ansonsten erhalten bleiben muss, unabhängig davon welche anderen Quellen existieren.

3. Fall $bc \notin V$: Hier bleibt die Reihenfolge über den Schritt von i nach $i + 1$ für die Stränge a, b, c auf jeden Fall erhalten, da ab, ac keine Quellen sind. Also bleibt auch die Regel 1 erfüllt.

(b) *Adjazenz der Permutationen π_i, π_{i+1}*: Da die Menge aller in Schritt i entfernten Swaps S_i nur aus Quellen besteht, muss sie disjunkt sein (sonst würden zwei Elemente einen Strang teilen und damit eine Kante zwischen sich haben, sodass ein Element keine Quelle wäre). Außerdem besteht sie nur aus Swaps, sodass $\Pi_{s \in S_i} s \in S_{n,2}$ gilt. Weiter ist nach Lemma 3.5 jede Quelle benachbart in π_i, also auch jeder Swap in S_i und somit unterstützt π_i die Permutation $\Pi_{s \in S_i} s$ und damit sind π_i und π_{i+1} adjazent.
Der Algorithmus terminiert also und wir erhalten einen gültigen Tangle. □

Mit dem folgenden Lemma wird insbesondere klar, dass der Algorithmus tatsächlich nützlich ist.

Lemma 3.7. *Sei O eine geordnete und azyklische Orientierung zum Swapgraphen G, dann hat $T(O_G)$ genau die Höhe $l + 1$ mit l der Länge des längsten Pfades in O_G.*

Beweis. In jedem Schritt wird eine Permutation erzeugt. Die Höhe entspricht also der Anzahl an ausgeführten Schritten plus der Startpermutation. Sei $P := (v_1, \ldots, v_l)$ ein längster Pfad in G. Da immer nur Quellen entfernt werden, kann pro Schritt immer nur genau das vorderste Glied pro Schritt entfernt werden. Man braucht also mindestens l Schritte.
Sei nun angenommen, der Algorithmus braucht mehr als l Schritte. Dann gibt es einen Knoten v'_1, der erst nach mindestens l Schritten zur Quelle wird. Er hat also einen Vorgänger v'_2, der nach $l - 1$ Schritten zur Quelle wird. Würde er früher eine Quelle, würde auch v'_1 früher zu einer. Dieses Vorgehen lässt sich fortführen, sodass dann ein Pfad $P' = (v'_{l+1}, \ldots, v'_1)$ entsteht, der länger ist als der längste Pfad P, was zu einem Widerspruch führt. Also entspricht die Länge des längsten Pfades genau der Anzahl an Schritten und damit der Höhe des Tangles minus eins. □

Also können wir nun aus einem Tangle eine Orientierung zum Swapgraphen bilden und daraus wieder einen Tangle. Besonders interessant ist, dass die Höhe des Tangles nach der Überführung und Rücküberführung sogar kleiner werden kann nach Lemma 3.2 (leider offensichtlich nur einmal) und das Umwandeln in polynomieller Zeit möglich ist. Das bleibt hier ohne Beweis, ist allerdings bei Betrachten der Algorithmen offensichtlich.

3.3 Minimale Zyklen in geordneten Orientierungen

Wie wir gesehen haben, entspricht ein Swapgraph mit der geordneten Orientierung O_G zu einer Liste genau dann einem Tangle, wenn O_G azyklisch ist. Nun wollen wir herausfinden, wie man Zyklen vermeiden kann – insbesondere mit dem Ziel eine geordnete und azyklische Orientierung mit minimalem längsten Pfad zu finden. Nach Lemma 3.7 ist der längste Pfad nämlich entscheidend für die Höhe des resultierenden Tangles.
Ein *kleinster* oder *minimaler Zyklus* im Graphen ist jener, der von allen Zyklen die wenigsten Knoten umfasst. Wir wollen nun zeigen, dass jede zyklische und geordnete Orientierung einen kleinsten Zyklus der Länge 3 enthält. Dann erhalten wir einen azyklischen Graphen, wenn wir jedes Tripel zyklenfrei halten. Um dies zu beweisen, stellen wir zuerst einige Lemmata in der Widerspruchsannahme auf, dass es in einem Swapgraphen einen kleinsten Zyklus größer als 3 geben kann.

Lemma 3.8. *Sei $Z := (a_1b_1, \dots, a_kb_k)$ ein kleinster Zyklus in der Orientierung O_G zum Swapgraph G mit $k \geq 4$. Dann gilt für jeden Strang $x \in [n]$, der in einem Swap a_ib_i enthalten ist, dass es genau einen weiteren Swap a_jb_j mit $i \neq j$ in Z gibt, in welchem x enthalten ist. Außerdem sind a_ib_i und a_jb_j benachbart in Z.*

Beweis. Wir machen zuerst folgende Beobachtung, aus welcher der Rest schnell folgt. Es gilt im Allgemeinen, dass in einem minimalen Zyklus $Z = (a_1b_1, \dots, a_kb_k)$ keine Kante zwischen zwei im Zyklus nicht benachbarten Knoten a_ib_i und a_jb_j sein kann; besitzen also zwei Swaps einen gemeinsamen Strang, müssen sie bereits benachbart sein (für $i < j$ gilt also $i + 1 = j$). Für den Beweis dieser Beobachtung sei nun angenommen Z wäre minimal, wobei die Kante $(a_ib_i, a_jb_j) \in O_G$ existiert und der Zyklus Z von der Form $Z = (a_1b_1, \dots, a_ib_i, a_{x_1}b_{x_1}, \dots, a_{x_l}b_{x_l}, a_jb_j, \dots, a_kb_k)$ mit $l \geq 1$ ist. Dann ist der Zyklus $Z' = (a_1b_1, \dots, a_ib_i, a_jb_j, \dots, a_kb_k)$ offensichtlich kürzer als Z. Das beweist die letzte Aussage des Lemmas, wie auch in Abbildung 3.4 zu sehen ist.
Angenommen es gäbe einen Strang x in mehr als zwei Knoten. Ohne Beschränkung der Allgemeinheit sei $x = a_{i_1} = a_{i_2} = a_{i_3}$ mit $1 < i_1 < i_2 < i_3 \leq k$ (es steht also zumindest der Swap a_1b_1 zwischen $a_{i_1}b_{i_1}$ und $a_{i_3}b_{i_3}$; die Indizes lassen sich immer so rotieren, da $k \geq 4$). Da der Graph orientiert ist und zwei Knoten eine Kante besitzen, wenn sie einen Strang teilen, gibt es zwischen $a_{i_1}b_{i_1}, a_{i_2}b_{i_2}$ und $a_{i_3}b_{i_3}$ jeweils Kanten, wobei $a_{i_1}b_{i_1}$ und $a_{i_3}b_{i_3}$ nicht benachbart sind. Damit kann der Zyklus nach obiger Beobachtung also nicht minimal sein.
Wäre nun ein Strang x nur einmal enthalten, müsste es ein anderer dreimal sein, denn betrachte den Teil (ab, cx, de) aus dem Zyklus. Ohne Beschränkung der Allgemeinheit sei $a = c$ (ein Strang muss gleich sein, da sie eine Kante verbindet). Da nun $e \neq x \neq d$ gilt, muss $e = a$ oder $d = a$ gelten und damit ist a drei mal enthalten. Wie gerade davor gezeigt, kann aber kein Strang drei mal enthalten sein. □

Korollar 3.9. *Jeder minimale Zyklus in der geordneten Orientierung O_G ist von der Form $(ab, bc, cd, \dots, xa)$ mit $a, \dots, x \in [n]$ paarweise verschiedenen Strängen.*

Beweis. Da die Reihenfolge bei Swaps keine Rolle spielt, also $ab = ba$ und Lemma 3.8 gilt, sind alle $a, \dots, x$ paarweise verschieden und die Knoten so angeordnet. □

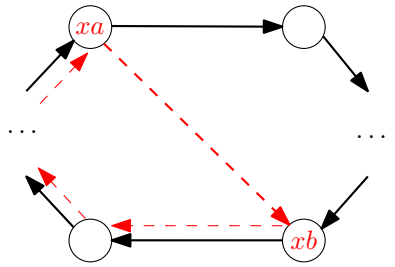

Abbildung 3.4: Die schwarzen Pfeile stellen einen beliebigen Zyklus mit vier oder mehr Elementen dar. Ist dieser minimal, so kann es keinen Strang x geben, der in zwei Swaps vorkommt, welche im Zyklus nicht benachbart sind, denn dies würde einen kürzeren Zyklus erzeugen, wie man in rot sieht.

Seien der Einfachheit halber nun alle minimalen Zyklen von der in Korollar 3.9 erwähnten Form $Z = (ab, bc, cd, \ldots, xa)$.

Lemma 3.10. *Sei G ein Swapgraph mit Orientierung O_G und sei Z ein minimaler Zyklus der Länge $m \geq 4$. Dann enthält Z keine zwei aufeinanderfolgenden Kanten, die in einem Swap-Tripel sind.*

Beweis. Sei $Z = (ab, bc, cd, \ldots, xa)$ ein minimaler Zyklus der Länge $m \geq 4$ in einer geordneten Orientierung O_G des Swapgraphen G. Im Folgenden habe der Zyklus zwei aufeinanderfolgende Kanten $(ab, bc), (bc, cd)$, die in Swap-Tripeln enthalten sind, also $ac, bd \in V$. Zum Verständnis ist es sinnvoll, sich die Regeln nochmal kurz klarzumachen, siehe Definition 3.4 und das Lemma 3.3 und jeweilige Umkehrungen; wir bezeichnen das Lemma 3.3 als 3. Regel. Wir verwenden auch folgende Notation für die Richtung von Kanten: $ab \to bc \Leftrightarrow (ab, bc) \in O_G$. Für die Bilder verwenden wir außerdem folgendes einheitliches Schema.

- Schwarz und etwas dickere Linien sind die gegebenen Kanten im minimalen Zyklus, die Dünnen sind weitere Kanten, die entweder so auftreten oder auf Grund einer Annahme gerichtet sind. Für die Übersichtlichkeit werden diese auch teilweise weggelassen.
- Grün für Kanten, die auf Grund der Regel 2 für geordnete Orientierung gerichtet werden, also in Swap-Tripeln sind.
- Blau, wenn ein kürzerer Zyklus vermieden wird.
- Violett für Kanten, die nach Regel 1 auf Grund einer gefolgerten Ordnung und Nicht-Existenz eines Swaps stark gerichtet sein müssen.
- Rot ist die Kante, die in keine Richtung orientiert werden kann, ohne irgendeine Regel zu verletzen oder einen kürzeren Zyklus zu erzeugen.
- Die Nummern zeigen die Reihenfolge an, in der die Kanten gerichtet werden.

Wir betrachten zuerst den Fall $m = 4$. Sei also (ab, bc, cd, ad) dieser Zyklus. Wir unterscheiden die Anordnung der Stränge im abc-Tripel. Vergleiche auch die Abbildung 3.5.

1. Fall $a < b < c$: In der Abbildung 3.5 links. Aus Regel 2 mit der Ordnung $a < b < c$ und $ab \to bc$ folgt, dass $ab \to ac$ und $ac \to bc$ enthalten sein müssen (grün). Weiter dürfen ja keine Zyklen der Größe 3 entstehen, also muss $ac \to cd$ sein, sonst ist (ac, bc, cd) ein Zyklus (blau). Nun kann die Kante (ac, ad) nicht gerichtet werden, ohne einen der beiden kürzeren Zyklen (ab, ac, ad) oder (ad, ac, cd) zu erzeugen (rot), was der Annahme widerspricht, dass Z ein minimaler Zyklus ist.

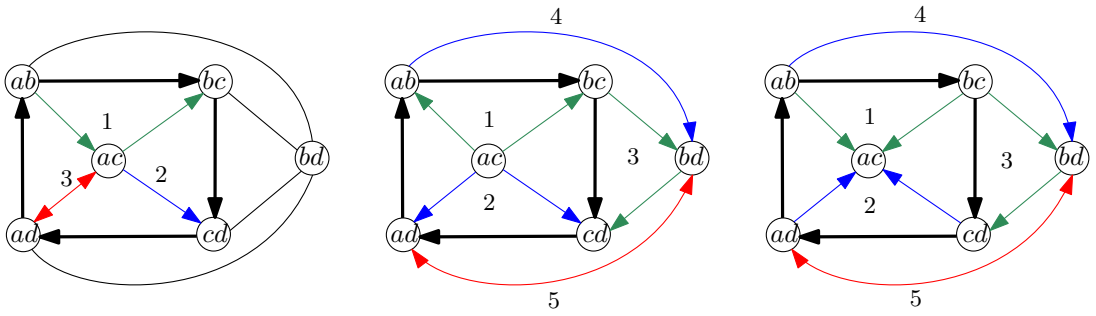

Abbildung 3.5: Die drei möglichen Fälle im Beweis zu Lemma 3.10, $m = 4$. Die Ordnungen von links nach rechts sind $a < b < c$, $a < c < b$ und $b < a < c$. Die gespiegelten Fälle sind jeweils analog. Die Folgerungen sind nach dem Schema im Beweis des gerade verlinkten Lemmas gekennzeichnet und werden hier nicht nochmal beschrieben.

2. Fall $a < c < b$: In der Abbildung 3.5 mittig. Aus der Richtung $ab \to bc$ mit $a < c < b$ folgt aus Regel 2, dass $ac \to ab, ac \to bc$ ist (grün, 1) und um nun keine kürzeren Zyklen (ac, bc, cd) oder dann (ac, cd, ad) zu erzeugen, müssen die Kanten $ac \to cd$ und $ac \to ad$ so gerichtet sein (blau, 2). Nun können wir aus der Orientierung des acd-Tripels ablesen, dass $d < a < c < b$ gelten muss, da entweder $c < a < d$ oder $d < a < c$ gilt, von welchen Ersteres im Widerspruch zu $a < c < b$ steht. Für das bcd-Tripel muss mit dieser Ordnung und Regel 2 also $bc \to bd, bd \to cd$ gelten (grün, 3). Damit kein kleinerer Zyklus (ab, bc, bd) entsteht, muss $ab \to bd$ sein (blau, 4). Nun kann die Kante (ad, bd) nicht gerichtet werden, ohne einen kürzeren Zyklus (ab, bd, cd) oder (cd, ad, bd) zu erzeugen, was wieder im Widerspruch zur Minimalität von Z steht (rot).

3. Fall $b < a < c$: In der Abbildung 3.5 rechts. Wieder kann auf Grund der Ordnung $b < a < c$ und $ab \to bc$ mit Regel 2 $ab \to ac, bc \to ac$ gefolgert werden (grün, 1). Außerdem kann wegen potentiell kürzeren Zyklen erst $ad \to ac$, wegen des (ab, ac, ad)-Zyklus und dann $cd \to ac$, wegen des (ac, cd, ad)-Zyklus gerichtet werden (blau, 2). Nun ergibt sich wegen des acd-Tripels die Ordnung $b < a < c < d$, sodass dann im bcd-Tripel $bc \to bd$ und $bd \to cd$ (grün, 3) und wegen des kürzeren Zyklus (ab, bc, bd), $ab \to bd$ gelten muss (blau, 4). Wieder kann die Kante $ad \to bd$ nicht gerichtet werden, ohne einen Zyklus (ad, bd, cd) oder (ab, bd, ab) zu erzeugen (rot), was einen Widerspruch zur Minimalität von Z ergibt.

Da auf Grund der Spiegelsymmetrie alle Möglichkeiten betrachtet wurden, ist die Behauptung für den Fall $m = 4$ gezeigt.
Wir widmen uns nun dem allgemeinen Fall $m > 4$ und betrachten dieselbe Fallunterscheidung wie zuvor für $m = 4$. Der Zyklus ist dieses mal von der Form $Z = (ab, bc, cd, de, \ldots, ax)$, wobei für den Strang e im Knoten de gelte $e = x$ oder $e \neq x$, je nach Größe des Zyklus. Dabei seien $ac, bd \in V$ enthalten, also die beiden Swap-Tripel abc und bcd. In den Abbildungen wird der neue Swap de und der Restzyklus durch die drei roten Kreise dargestellt.

1. Fall $a < b < c$: Vergleiche Abbildung 3.6 links. Auf Grund der Ordnung im abc-Tripel und $ab \to bc$ folgt aus der Regel 2, dass $ab \to ac \to bc$ (grün) gilt. Nun muss wegen der Minimalität von Z die Kante $ac \to cd$ sein (blau), sonst ist (ac, bc, cd) ein kürzerer Zyklus. Es ist entweder (ab, ac, ax) oder $(ax, ac, cd, ..., ax)$ ein kürzerer Zyklus als Z (rot, auch der zweite hat offensichtlich ein Element weniger).

2. Fall $a < c < b$: Vergleiche Abbildung 3.6 mittig. Es ergibt sich aus der Ordnung, dass alle Kanten von ac weggehen, also $ac \to ab, ac \to bc, ac \to cd, ac \to ax$ (die

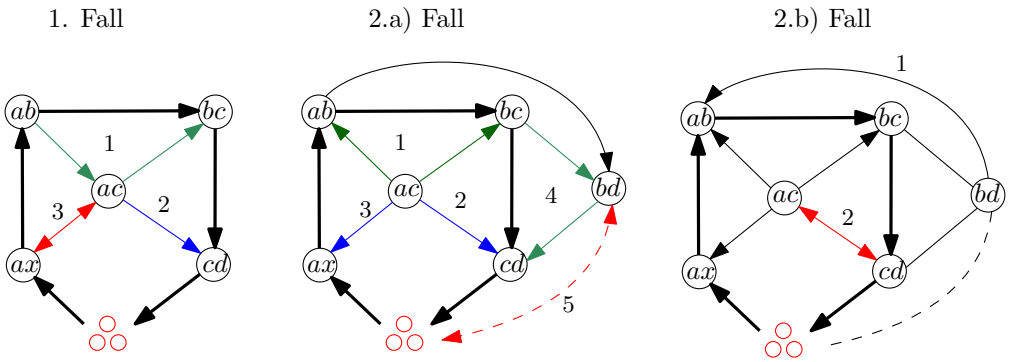

Abbildung 3.6: Die ersten drei Fälle des allgemeinen Beweises für das Lemma 3.10. Die drei roten Kreise stehen für einen oder mehr weitere Swaps im Zyklus. Die vier schwarzen, dünnen Pfeile im rechten Bild sind genau wie im Mittigen zu begründen. Die gestrichelte Linie symbolisiert die Kante (bd, de), wobei womöglich $e = x$ gilt. Eine Erklärung zu den verwendeten Farben und Zahlen steht im Anfang des Beweises und genauer begründet im jeweils zugehörigen Beweisabschnitt.

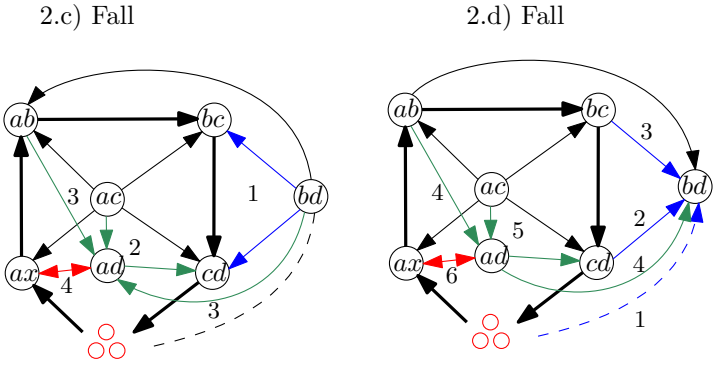

Abbildung 3.7: Die letzten zwei Unterfälle 2.c und 2.d des 2. Falls für den Beweis des Lemmas 3.10. Der obere schwarz gerichtete Pfeil in beiden Graphen ist jeweils nach Annahme so gerichtet. Die aus ac ausgehenden Kanten folgen genau wie im Bild des Falles 2.a. Die Begründungen finden sich wieder im Farbschema am Anfang des Beweises und im jeweiligen Beweisabschnitt.

ersten zwei in grün wegen des abc-Tripels und, da $ab \to bc$ gilt; die letzteren zwei im Uhrzeigersinn nacheinander in blau um keinen kürzeren Zyklus zu erzeugen). Nun unterscheiden wir weiter folgende Unterfälle:

a) Fall $ad \notin V \wedge ab \to bd$: Vergleiche Abbildung 3.6 mittig. Dann kann wegen Regel 3 der Strang d in der Ordnung nicht kleiner als a sein, denn sonst müsste ad enthalten sein. Auf Grund der starken Orientierung $ab \to bd$ muss nun $d < a < c < b$ und damit im bcd-Tripel $bc \to bd, bd \to cd$ gelten (blau). Die letzte gestrichelte Kante (rot) kann dann nicht korrekt gerichtet werden, denn in jedem Fall entsteht ein kürzerer Zyklus als Z.

b) Fall $ad \notin V \wedge bd \to ab$: Vergleiche Abbildung 3.6 rechts. Aus der starken Orientierung $bd \to ab$ folgt $a < d < b$. Betrachte die Richtung $ac \to cd$, die nun unmöglich ist, denn um so zu verlaufen müsste $d < a < c$ gelten, was mit $a < d < b$ unvereinbar ist. Also müsste Knoten ad doch existieren.

c) Fall $ad \in V \wedge bd \to ab$: Betrachte Abbildung 3.7 links. Um keinen kürzeren Zyklus (ab, bc, bd) zu erzeugen, muss $bd \to bc$ sein und wegen des Zyklus (bd, bc, cd)

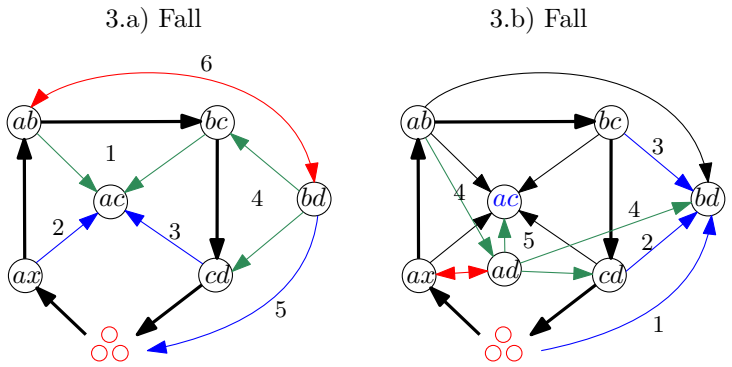

Abbildung 3.8: Die Fälle 3.a und 3.b des Lemmas 3.10. Die schwarzen, zu ac gerichteten Pfeile rechts folgen genau wie links. Das Farbschema und die Begründung stehen genauer im Beweis.

auch $bd \to cd$ (blau). Nun haben wir hier ein bcd-Tripel, sodass $a < c < d < b$ gelten muss (zusammen mit Anfangsbedingung $a < c < b$). Nun können wir das acd- und das abd-Tripel richten (grün), was uns zu einem kürzeren Zyklus (ax, ab, ad) oder $(ad, cd, \ldots, ax)$ bei der roten Kante zwischen ax und ad führt.

d) Fall $ad \in V \wedge ab \to bd$: Betrachte Abbildung 3.7 rechts. Zunächst können wir die blauen Pfeile richten, um keinen kürzeren Zyklus zu bekommen, zuerst den gestrichelten $de \to bd$ wegen des Zyklus $(ab, bd, ..., ax, ab)$, dann die im bcd-Tripel auch wegen kürzeren Zyklen (bd, cd, de) für $cd \to bd$ und (bc, cd, bd) für $bc \to bd$. Aus dem bcd-Tripel ergibt sich dann die Anordnung $a < c < b < d$ ($d < b < c$ kann auf Grund der Annahme $c < b$ ausgeschlossen werden). Nun müssen nach Regel 2 im abd-Tripel die Richtungen $ab \to ad$ und $ad \to bd$ und im acd-Tripel $ac \to ad$ und $ad \to cd$ gelten (grün). Damit ergibt sich in jedem Fall bei der roten Kante (ax, ad) ein kürzerer Zyklus $(ad, cd, \ldots, ax)$ oder (ab, ad, ax).

3. Fall $b < a < c$: Vergleiche Abbildung 3.8 links (Schritte 1,2 und 3). Wie für $m = 4$ müssen alle Kanten zu ac hinzeigen, zuerst folgen $ab \to ac$ und $bc \to ac$ wegen des abc-Tripels und der Kante $ab \to bc$. Um dann kürzere Zyklen zu vermeiden, folgen dann auch $ax \to ac$, wegen (ab, ac, ax), und $cd \to ac$, wegen $(cd, \ldots, ax, ac)$. Nun unterscheiden wir weiter folgende Fälle:

a) Fall $ad \notin V$: Vergleiche Abbildung 3.8 links. Damit ist also $cd \to ac$ stark gerichtet und es gilt $b < a < d < c$ ($c < d < a$ lässt sich nach Annahme $a < c$ ausschließen). Durch diese Ordnung und $bc \to cd$ lässt sich das bcd-Tripel richten (grün) mit $bd \to bc, bd \to cd$. Um dann keinen kürzeren Zyklus $(bd, cd, ..., bd)$ zu erhalten muss bd in den Restzyklus hineingerichtet sein (blau) $bd \to de$. Jetzt lässt sich die Kante zwischen ab und bd nicht korrekt orientieren, da entweder der $(ab, bd, ..., xa)$-Zyklus entsteht, der kürzer ist oder ad existieren müsste, um trotz der Ordnung $b < a < d < c$ die Kante $bd \to ab$ zu richten (rot).

b) Fall $ad \in V \wedge ab \to bd$: Vergleiche Abbildung 3.8 rechts. Zuerst wird klar, dass der Restzyklus unten nach bd gerichtet werden muss $de \to bd$, da sonst bereits ein geringerer Zyklus $(ab, bd, ..., xa)$ entsteht. Um weiter die Zyklen (cd, de, bd) und (bc, cd, bd) zu vermeiden werden die beiden anderen blauen Kanten im bcd-Tripel nach bd gerichtet $cd \to bd, bc \to bd$, was uns durch das bcd-Tripel die Anordnung $d < b < a < c$ liefert. Nun können wir das abd- und das acd-Tripel orientieren und erhalten die Richtungen $ab \to ad, ad \to bd$ und $ad \to ac, ad \to cd$ (grün) und kommen zur Erkenntnis, dass (ax, ad) nicht korrekt orientiert werden

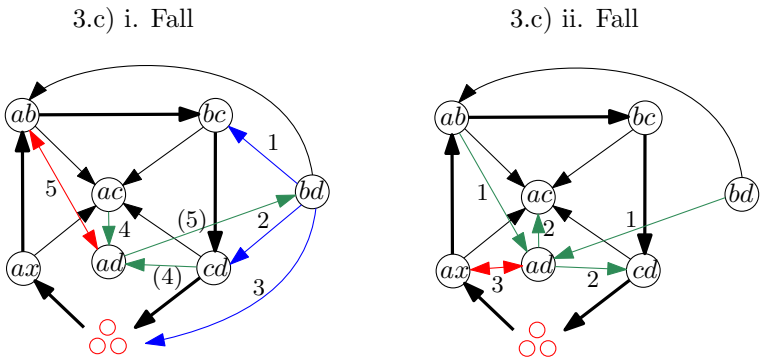

Abbildung 3.9: Die letzten zwei Unterfälle 3.c.i und 3.c.ii des Beweises im Lemma 3.10. Die zu ac hinzeigenden schwarzen Pfeile ergeben sich wie in Fall 3.a weiter oben.

kann (rot), ohne einen kürzeren Zyklus $(ad, cd, \ldots, ax)$ oder (ab, ad, ax) zu erzeugen.

c) Fall $ad \in V, bd \to ab$: Vergleiche Abbildung 3.9 links. Die drei blauen Richtungen $bd \to bc, bd \to cd$ und $bd \to de$, ergeben sich von oben nach unten sofort, um keinen kürzeren Zyklus zu erschaffen. Nun können wir aus dem bcd-Tripel folgern, dass $b < d < c$ gilt (wobei $c < d < b$ wegen der Annahme $b < c$ ausgeschlossen werden kann), was wiederum zwei letzte Fälle ergibt:

   i. $b < a < d < c$: Aus dieser Anordnung lassen sich für die abd- und acd-Tripel die grünen Richtungen $ad \to ab$ und $ac \to ad$ folgern, was dann einen kürzeren Zyklus (ab, ac, ad) ergibt.

   ii. $b < d < a < c$: Vergleiche Abbildung 3.9 rechts. Wieder lassen sich durch die Ordnung das abd- und das acd-Tripel mit $ab \to ad, bd \to ad$ und $ad \to ac, ad \to cd$ wie grün gekennzeichnet anordnen. In rot sieht man wieder, dass die Kante zwischen ax und ad nicht gerichtet werden kann, ohne dass ein kürzerer Zyklus $(cd, \ldots, ax, ad)$ oder (ab, ad, ax) entsteht.

Wie man erkennen kann, werden alle Fälle abgedeckt, zum einen in den drei Fällen mit ihren symmetrischen Pendants die sechs möglichen Anordnungen von a, b, c und zum anderen die Unterfälle auch jeweils vollständig und damit ist die Behauptung gezeigt. □

Nun schränken wir weiter ein, wie viele stark gerichtete Kanten nacheinander existieren können.

Lemma 3.11. *Jeder Zyklus in einer geordneten Orientierung in einem Swapgraphen G auf einer lösbaren Liste L besitzt mindestens eine schwach gerichtete Kante.*

Beweis. Haben wir also eine beliebige, aber lösbare Liste L, so gibt es einen Tangle T, der diese Liste löst; wir erzeugen etwa einen mit Wangs parallelen Bubblesort-Algorithmus [Wan91]. Nun bilden wir die Orientierung $O(T)$ zum Tangle auf dem Swapgraphen G, welche nach Lemma 3.6 azyklisch und geordnet ist. Es sei nun O' eine beliebige zyklische und geordnete Orientierung auf G, die einen Zyklus enthält, der nur aus stark gerichteten Kanten besteht. Weil in jeder geordneten Orientierung auf G die stark gerichteten Kanten dieselben sind, da diese nur von der Existenz eines Knotens abhängen, wie in Definition 3.4 zu sehen ist, muss dieser Zyklus auch in $O(T)$ existieren. Damit ist diese nicht azyklisch, was einen Widerspruch zu Lemma 3.6 ergibt und die Behauptung ist gezeigt. □

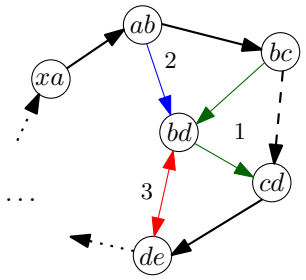

Abbildung 3.10: Der erste Fall des Beweises von Lemma 3.12. Die dicken durchgezogenen Kanten sind stark, die Gestrichelten schwach gerichtet und die Gepunkteten können beides sein. Wir können auf Grund des vorigen Lemmas 3.12 annehmen, dass eine Kante schwach gerichtet ist. Die grüne Kante lässt sich wegen des Swap-Tripels richten, die Blaue, um keinen kleineren Zyklus zu erzeugen und bei der Roten entsteht immer ein kleinerer Zyklus.

Lemma 3.12. *Sei G ein Swapgraph mit Orientierung O_G und sei Z ein minimaler Zyklus der Länge $m \geq 4$. Dann enthält Z keine zwei aufeinanderfolgenden Kanten, die stark gerichtet sind.*

Beweis. Wir betrachten erst den Sonderfall $m = 4$. Sei also $Z = (ab, bc, cd, ad)$ ein minimaler Zyklus mit $ac, bd \notin V$. Damit sind bereits alle Kanten des Zyklus stark gerichtet, was ein Widerspruch zum vorherigen Lemma 3.11 ist.
Nun nehmen wir $m > 4$ an. Unser Zyklus ist also von der Form $Z = (ab, bc, cd, de, \ldots, xa)$ mit $|Z| \geq 5$. Weiter seien $xb, ac \notin V$, also die Kanten $xa \to ab, ab \to bc$ stark gerichtet. Wie wir im vorhergehenden Lemma 3.11 gesehen haben, können wir annehmen, dass eine Kante schwach gerichtet ist. Es sei also die Kante $bc \to cd$ schwach gerichtet mit $bd \in V$. Nach Lemma 3.10 können auch keine zwei Kanten nacheinander schwach gerichtet sein, weswegen $cd \to de$ wieder stark gerichtet sein muss mit $ce \notin V$. Wir erhalten also insgesamt den Graphen wie in Abbildung 3.10, wobei die farbigen Kanten noch ungerichtet sind. Wir betrachten die Kante $ab \to bc$ und erhalten auf Grund der starken Richtung entweder $b < a < c$ oder $c < b < a$, wobei wir ohne Beschränkung der Allgemeinheit wieder ersteren Fall betrachten – der Zweite ist wegen der Spiegelsymmetrie analog. Mit der stark gerichteten Kante $xa \to ab$ erhalten wir $b < x < a < c$. Für die letzte stark gerichtete Kante $cd \to de$ erhalten wir zwei Möglichkeiten: $b < x < a < c < d, e < c$ und $b < x < a < c < e, d < c$, die wir separat behandeln wollen.

1. Fall $b < x < a < c < d, e < c$, vergleiche Abbildung 3.10: Da wir nun die Ordnung des bcd-Tripels kennen, können wir mit der Richtung $bc \to cd$ die Kanten $bc \to bd$ und $bd \to cd$ folgern. Um keinen (ab, bc, bd)-Zyklus zu erzeugen richten wir $ab \to bd$ und können die Kante (bd, de) nicht richten, ohne einen kürzeren Zyklus $(de, \ldots, xa, ab, bd)$ oder (bd, cd, de) zu bilden, sodass wir diesen Fall ausschließen können.

2. Fall $b < x < a < c < e, d < c$: Da wir hier für den Strang d keine genauere Aussage machen können, als $d < c$ unterscheiden wir weiter folgende zwei Fälle:

   a) Fall $d < b$, vergleiche Abbildung 3.11 links: Da nun die Ordnung der Stränge im bcd-Tripel bekannt ist, können wir damit und der Kante $bc \to cd$ das Tripel mit $bc \to bd$ und $cd \to bd$ richten. Die Kanten $ab \to bd$ und $de \to bd$ können nun gerichtet werden, da in die andere Richtung kürzere Zyklen (ab, bc, bd) oder $(ab, bd, de, \ldots, ax)$ entstehen würden. Weiter muss nach Lemma 3.3 wegen der Ordnung $d < b < a$ und $ab, bd \in V$ der Swap ad enthalten sein und die Kanten in diesem Swap-Tripel können damit auch gleich orientiert werden.

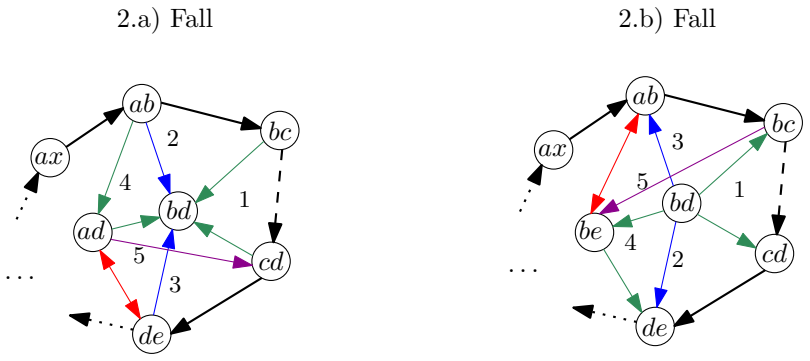

Abbildung 3.11: Hier sehen wir die zwei Unterfälle 2.a und 2.b des Beweises des Lemmas 3.12. Die dicken durchgezogenen Kanten sind stark gerichtet, die Gestrichelte schwach gerichtet und die Gepunkteten können beides sein. Das Farbschema entspricht wieder dem im Beweis des Lemmas 3.10. Die Begründung für die Richtungen sind im Beweis.

Wir erhalten $ab \to ad$ und $ad \to bd$. Da der Swap ac nach Voraussetzung nicht existiert, ist die Kante $ad \to cd$ stark gerichtet. Nun existiert wegen der Kante (de, ad) entweder der (cd, de, ad)- oder der $(ad, de, \ldots, xa, ab)$-Zyklus, die beide kleiner als Z sind, was der Annahme widerspricht, dass der Zyklus Z minimal ist.

b) Fall $b < d$, vergleiche Abbildung 3.11 rechts: Wieder ist hier das bcd-Tripel mit der Kante $bc \to cd$ und der Ordnung $b < d < c$ eindeutig festgelegt und wir erhalten $bd \to bc$ und $bd \to cd$. Um keinen (bd, cd, de)-Zyklus zu erzeugen muss $bd \to de$ gelten, und um keinen $(bd, de, \ldots, ax, ab)$-Zyklus zu erzeugen $bd \to ab$. Es kann die Kante $bd \to de$ nicht stark gerichtet sein, denn $b < d < e$ gilt. Der Swap be muss also existieren. Wir richten wegen $bd \to de$ das bde-Tripel mit $bd \to be$ und $be \to de$. Da der Swap ce nach Voraussetzung nicht existiert, muss die Kante $bc \to be$ stark gerichtet sein. Nun entsteht durch die Kante (ab, be) immer ein kürzerer Zyklus $(ab, bc, de, \ldots, ax)$ oder (ab, bc, be).

Wir sehen also, dass für alle Ordnungen ein kleinerer Zyklus entsteht, wodurch die Behauptung gezeigt wurde. □

Wir haben nun genügend Einschränkungen für die Struktur von minimalen Zyklen, dass wir folgenden Satz beweisen können.

Satz 3.13. *In jeder geordneten zyklischen Orientierung O_G zu einem Swapgraphen G gibt es einen minimalen Zyklus der Größe 3.*

Beweis. Es sei wie auch vorher angenommen, dass in der geordneten und azyklischen Orientierung O_G kein kleinster Zyklus der Größe 3 existiert. Wir betrachten zuerst den Fall $m = 4$. Sei also $Z = (ab, bc, cd, ad)$ der minimale Zyklus. Nach den Lemmata 3.12, 3.10 können zwei Kanten nacheinander weder beide stark noch schwach gerichtet sein. Seien also $ad \to ab$ und $bc \to cd$ stark und $ab \to bc$ und $cd \to ad$ schwach gerichtet. Es gilt damit $ac \in V, bd \notin V$. Betrachten wir nun die Kante $bc \to cd$, dann können wir auf Grund ihrer starken Richtung $c < b < d$ oder $d < b < c$ folgern. Wir beweisen wegen der Spiegelsymmetrie nur den Fall $c < b < d$, der Beweis für $d < b < c$ geht analog. Mit der starken Orientierung $ad \to ab$ folgt dann, dass die Ordnung $c < b < d < a$ gilt. Nun können wir die beiden abc- und acd-Tripel orientieren mit $ab \to ac \to bc$ und $cd \to ac \to ad$ und erhalten sofort zwei kleinere Zyklen (ab, ac, ad) und (bc, cd, ad) (vergleiche Abbildung 3.12),

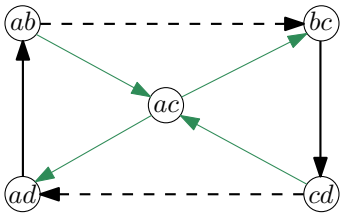

Abbildung 3.12: Der Zyklus für den Beweis des Satzes 3.13 für einen Zyklus der Länge $m = 4$. Die dicken Kanten stellen den Zyklus dar, gestrichelt sind schwach und durchgezogen stark gerichtete Kanten. Die grünen Kanten entstehen auf Grund der Ordnung $c < b < d < a$, die aus den starken Kanten abgeleitet werden kann. Wir erhalten sofort zwei Zyklen der Größe 3: (ab, ac, ad) und (bc, cd, ad).

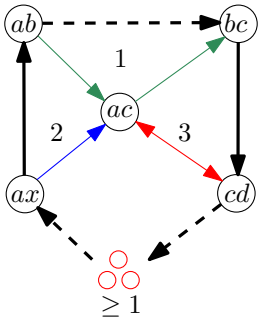

Abbildung 3.13: Der erste Fall des Beweises des Satzes 3.13. Es gilt die Ordnung $a < b < c$. Das Farbschema ist wieder das Selbe wie im Beweis des Lemmas 3.10.

was im Widerspruch zur Minimalität des Zyklus Z steht.
Nehmen wir nun an, es gäbe einen minimalen Zyklus $Z := (ab, bc, cd, \ldots, xa)$ der Größe $m > 4$. Wie in den vorigen Lemmata gezeigt, ist also genau jede zweite Kante in einem Swap-Tripel. Hier sei $ac \in V$ und somit das abc-Tripel enthalten, und damit $xb, bd \notin V$. Wir unterscheiden nun folgende drei Fälle für die Ordnung der Stränge a, b und c:

1. Fall $a < b < c$: Betrachte auch die Abbildung 3.13. Die grünen Kanten $ab \to ac$ und $ac \to bc$ folgen nach Regel 2 aus der Ordnung $a < b < c$ und der Kante $ab \to bc$. Um dann keinen (ab, ac, ax)-Zyklus zu erhalten, muss $xa \to ac$ enthalten sein (blau). Nun kann die Kante zwischen ac und cd nicht gerichtet werden, ohne einen kürzeren Zyklus zu erzeugen. Betrachte dazu die Zyklen $(\ldots, xa, ac, cd, \ldots)$ und (ac, bc, cd).

2. Fall $a < c < b$: Diesen Fall sehen wir links in der Abbildung 3.14. Wieder liefert die Anordnung $a < c < b$ mit der Kante $ab \to bc$ bereits die eindeutigen Richtun-

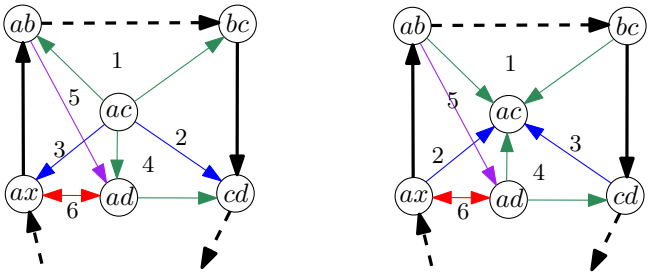

Abbildung 3.14: Der 2. und 3. Fall des Beweises des Satzes 3.13. Die Begründungen stehen im Beweis und das Farbschema ist wieder das Selbe wie im Beweis des Lemmas 3.10.

gen $ac \to ab$ und $ac \to bc$ mit Regel 2 (grün,1). Um kürzere Zyklen zu vermeiden muss nun $ac \to cd$ (sonst ist (ac, bc, cd) ein Zyklus) und dann $ac \to ax$ gelten (sonst ist $(\dots, ax, ac, cd, \dots)$ ein kürzerer Zyklus; blau). Durch die stark gerichtete Kante $bc \to cd$ muss nun nach Regel 1 und $c < b$ die Ordnung $d < c < b$ und damit wegen Transitivität $a < c < b < d$ gelten. Mit Lemma 3.3 und $ac, cd \in V, a < c < d$ folgt, dass ad auch im Graphen enthalten ist. Weiter können wir Regel 2 für das acd-Tripel anwenden und erhalten wegen $ac \to cd$, dass $ac \to ad, ad \to cd$ sein müssen (grün,4). Die Kante $ab \to ad$ ist so gerichtet, da nach Voraussetzung $bd \notin O_G$ ist und damit die Kante nach Regel 1 und der oben gesehenen Ordnung stark gerichtet sein muss (violett). Nun können wir die Kante (ax, ad) nicht mehr orientieren, sonst erhalten wir entweder den kürzeren Zyklus (ab, ad, ax) oder $(\dots, ax, ad, cd, \dots)$, was einen Widerspruch zur Minimalität des Zyklus Z bildet.

3. Fall $b < a < c$: Diesen Fall sehen wir rechts in der Abbildung 3.14. Regel 2 mit der Anordnung $b < a < c$ im abc-Tripel und $ab \to bc$ liefert $ab \to ac, bc \to ac$ (grün,1). Damit kein kürzerer Zyklus (ab, ac, ax) entsteht muss $ax \to ac$ sein und dann wegen des Zyklus $(\dots, ax, ac, cd, \dots)$ auch $cd \to ac$ sein (blau,2,3). Da $bc \to cd$ stark gerichtet ist, muss $d < b < a < c$ gelten nach Regel 1. Da $cd \to ac$ gilt, muss ad existieren, da sonst die Kante wegen der Ordnung $d < a < c$ und Regel 1 in die andere Richtung stark gerichtet sein müsste. Nun können wir das acd-Tripel mit dieser Ordnung und $cd \to ac$ richten und wir erhalten $ad \to ac$ und $ad \to cd$. Weiter muss nach Regel 3 $ab \to ad$ stark gerichtet sein (violett), da bd nicht existiert und $d < b < a$ gilt. Nun erhalten wir wieder einen kürzeren Zyklus, egal wie wir die Kante zwischen ax und ad richten (rot): entweder $(\dots, ax, ad, cd, \dots)$ oder (ab, ad, ax), was wieder zu einem Widerspruch zur Minimalität des Zyklus führt.

Aufgrund der Symmetrie sind damit alle Fälle behandelt worden und der Beweis ist vollständig. □

Korollar 3.14. *Jede Orientierung O_G, die keinen Zyklus der Länge 3 besitzt, ist zyklenfrei.*

Damit können wir eine geordnete azyklische Orientierung zu einem Swapgraphen erzeugen, indem wir darauf achten, jedes Tripel zyklenfrei zu halten. Man beachte auch, dass nur Tripel in Singleton-Cliquen zyklenfrei sein müssen, solche in Swap-Tripeln werden nie zyklisch, wenn sie nach Regel 2 gerichtet sind.

4. Praktische Erzeugung minimaler Tangles

Die bisher erhaltenen Ergebnisse reichen offenbar nicht aus, um tatsächlich in polynomieller Zeit einen minimalen Tangle zu finden. Es bleibt also die Frage, wie wir in der Praxis Tangles minimieren. Dazu testen wir verschiedene Varianten und vergleichen deren Laufzeit und Korrektheit.
Alle Vergleiche und Tests wurden am selben PC durchgeführt. Genutzt wurde ein Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz Prozessor mit 16GB RAM. Wir benennen den von Firman et al. vorgestellten Algorithmus [FKR$^+$19] als Firman und Wangs parallelen Bubblesort-Algorithmus [Wan91] als Wang. Zum Vergleich der Algorithmen lassen wir jeden auf einer vorher generierten Liste an Permutationen laufen, welche jeweils 200 Beispiele für Stränge der Länge $n = 4, 5, 6, 7, \ldots$ umfasst.

4.1 Ein ILP basierend auf Swapgraphen

Wir übersetzen den Swapgraphen aus dem vorhergehenden Kapitel in ein ganzzahliges lineares Gleichungssystem (kurz ILP aus dem Englischen *integer linear program*) und lassen es vom Programm Gurobi [GO20] lösen. Nachdem nun einige interessante Ergebnisse für die Swapgraphen vorliegen und ILP ein gut untersuchtes Problem sind, besteht durchaus die Möglichkeit, dass diese ILPs schnell gelöst werden können. Wir stellen die Reduktion vor und vergleichen dann die Laufzeit mit Firman.

4.1.1 Die Reduktion

Tragen wir nochmal kurz zusammen, was für Eigenschaften der Swapgraph erfüllen soll. Zuerst haben wir einen Graphen $G = (V, E)$ mit Swaps als Knoten und Kanten zwischen Swaps mit gleichen Strängen. Für diesen wollen wir eine Orientierung O_G nach den Regeln in Definition 3.4 finden, die azyklisch ist und in der der längste Pfad minimal ist. Alle diese Eigenschaften wollen wir in ein ILP übersetzen und beispielsweise vom Programm Gurobi [GO20] lösen lassen. Wir definieren zuerst folgende Variablen:

1. Zu jeder Kante $(v, w) \in E$ definieren wir zwei binäre Variablen $x_{(v,w)}, x_{(w,v)}$, die uns die Richtung der Kante beschreiben.
2. Zu jedem Knoten $v \in V$ definieren wir eine Variable $h(v) \in \mathbb{R}_{\geq 0}$, die die Länge des längsten Pfades bis zum Knoten v angibt.

3. Und wir erzeugen eine Variable $h \in \mathbb{N}_0$, welche die Länge des längsten Pfades repräsentiert.

Nun übersetzen wir die Eigenschaften auf das Gleichungssystem. Die meisten Schritte sind relativ klar. Am interessantesten ist wohl, wie wir im 5ten Schritt die Höhe jedes Knoten berechnen. Die Höhe $h(v)$ für $v \in V$ ist der längste Pfad zu diesem. Die Idee ist, dass wir für jede eingehende Kante zum Knoten v die Höhe um mindestens 1 größer bestimmen wollen als die Höhe des Knoten, von welchem die Kante kommt. Ist die Kante jedoch nicht zum Knoten hin gerichtet, so wird der Höhe keine tatsächliche Einschränkung nach unten geben.

1. Zuerst wollen wir für jede Kante genau eine Richtung erhalten. Wir sagen eine Kante ist genau dann von Swap v nach w gerichtet, wenn $x_{(v,w)} = 1$ ist. Also soll für jede Kante $(v, w) \in E$ die Gleichung $x_{(v,w)} + x_{(w,v)} = 1$ gelten.
2. Für jede stark gerichtete Kante können wir die Orientierung direkt angeben. Für alle $ab, ac \in V, bc \notin V, a < b < c$ soll also $x_{(ab,ac)} = 1$ sein.
3. Für die schwach gerichteten Kanten soll eine der beiden Möglichkeiten aus Regel 2 in der Definition 3.4 erfüllt sein. Für $ab, ac, bc \in V$ mit $a < b < c$ soll also entweder $(ab, ac), (ab, bc), (ac, bc) \in O_G$ sein oder genau die andere Richtung. Wir schreiben also $x_{(ab,ac)} = x_{(ab,bc)} = x_{(ac,bc)}$.
4. Weiter soll unser Graph azyklisch sein; für diese Eigenschaft nutzen wir den Satz 3.13 und betrachten alle Tripel im Graphen, die keine Swap-Tripel sind – also alle der Form $(ab, ac, ad) \in V$. Diese drei Knoten sollen keinen Zyklus bilden, also bestimmen wir, dass $1 \leq x_{(ab,ac)} + x_{(ac,ad)} + x_{(ad,ab)} \leq 2$ gelten muss. Die einzigen zwei Möglichkeiten, dass drei Elemente einen Zyklus bilden, werden damit ausgeschlossen.
5. Für zwei Knoten $v, w \in V$ mit $(v, w) \in E$ bestimmen wir die Höhe mit $h(w) \geq h(v) + 1 + n \cdot (x_{(v,w)} - 1)$, wobei n größer als jeder mögliche längste Pfad sein sollte, also etwa $n = |E| + 1$. Ist nun also v nach w gerichtet, also $x_{(v,w)} = 1$, so fällt der hintere Term mit n weg und $h(w) \geq h(v) + 1$ gilt. Ist die Kante anders herum von w nach v gerichtet, ist $x_{(v,w)} = 0$ und damit gilt $h(w) \geq h(v) + 1 - n$. Da wir n so hoch gewählt haben gilt $n \geq h(v) + 1$ und damit im schlimmsten Fall $h(w) \geq 0$, was keine Einschränkung ist.
6. Zu guter Letzt wollen wir die Länge des längsten Pfades minimieren, dafür nutzen wir $h \geq h(v)$ für alle $v \in V$ und setzen als Ziel des ILP: minimiere h.

Dieses Verfahren liefert uns nun eine geordnete und azyklische Orientierung des Swapgraphen mit minimalem längsten Pfad, indem wir für alle $x_{(v,w)} = 1$ in der Lösung des ILP's die Kante $(v, w) \in O_G$ setzen.

4.1.2 Beispiel

Um das Verfahren etwas deutlicher zu machen betrachten wir folgendes einfaches Beispiel. Für die Zielpermutation 4231 erhalten wir die Liste $L = \{12, 13, 14, 24\}$. Wir bilden dazu den Swapgraphen, wie in Abbildung 4.1 zu sehen. Wir erstellen die drei Arten von Variablen, einmal zu jeder Kante $(a, b) \in E : x_{(a,b)}$ und zu jedem Knoten $v \in V : h(v)$, sowie die zu minimierende Höhe h. Nun wollen wir schrittweise alle Gleichungen aufstellen.

1. Für jede Kante gibt es nur eine Richtung, also zum Beispiel für die Kante $(12, 13) \in E$ erhalten wir die Gleichung $x_{12,13} + x_{13,12} = 1$. Dies führen wir für alle Kanten fort und erhalten 5 Gleichungen.
2. Wir haben zwei stark gerichtete Kanten $(12, 13)$ und $(12, 14)$ (schwarz in der Abbildung). Da diese nur so gerichtet werden können, definieren wir $x_{(12,13)} = 1$ und $x_{(12,14)} = 1$.

$$L := \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

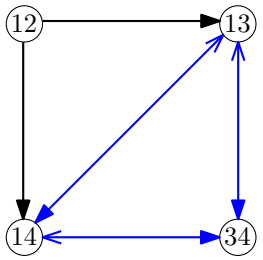

Abbildung 4.1: Hier sehen wir die Swap-Liste, sowie den Graphen für den das ILP erzeugt wird. In schwarz haben wir zwei stark gerichtete Kanten und in blau ist ein Swap-Tripel markiert. Die verschiedenen Pfeilspitzen deuten die beiden möglichen Orientierungen des Swap-Tripels an.

3. Nun haben wir ein Swap-Tripel $(13, 14, 34)$ im Graphen, was uns zu der Gleichung $x_{(13,14)} = x_{(13,34)} = x_{(14,34)}$ führt.
4. Wir haben genau ein Tripel, das kein Swap-Tripel ist, nämlich $(12, 13, 14)$. Damit erhalten wir die Ungleichung $1 \leq x_{(12,13)} + x_{(13,34)} + x_{(34,12)} \leq 2$.
5. Wir definieren n wie oben beschrieben $n = |E|+1 = 5+1 = 6$ und erhalten wir für jedes der zehn $x_{(a,b)}$ eine Ungleichung. Exemplarisch wollen wir zwei für die Kante $(13, 34)$ mit Hin- und Rückrichtung aufstellen. Wir erhalten $h(34) \geq h(13)+1+n\cdot(x_{(13,34)}-1)$ und $h(13) \geq h(34) + 1 + n \cdot (x_{(34,13)} - 1)$.
6. Natürlich müssen wir auch noch die Höhe mit $h \geq h(12), h \geq h(13), h \geq h(14), h \geq h(34)$ festlegen und setzen als Ziel die Minimierung von h.

Insgesamt erhalten wir also 15 Variablen und 26 Gleichungen:

$$\begin{array}{lll} \textit{minimiere } h & & \\ \textit{sodass} & x_{12,13} + x_{13,12} & = 1 \\ & x_{14,34} + x_{34,14} & = 1 \\ & \dots & \\ & x_{12,13} & = 1 \\ & x_{12,14} & = 1 \\ & x_{13,14} & = x_{13,34} \\ & x_{13,14} & = x_{14,34} \\ & x_{12,13} + x_{13,34} + x_{34,12} & \geq 1 \\ & x_{12,13} + x_{13,34} + x_{34,12} & \leq 2 \\ & h(13) + 1 + n \cdot (x_{(13,34)} - 1) & \leq h(34) \\ & h(34) + 1 + n \cdot (x_{(34,13)} - 1) & \leq h(13) \\ & \dots & \\ & h(12) & \leq h \\ & \dots & \leq h \end{array}$$

Es werden nicht alle 26 Gleichungen aufgeführt, die Fehlenden sind aber angedeutet.

4.1.3 Vergleich

Wir bezeichnen diesen Algorithmus als Swapgraph. Es stellt sich heraus, dass die Laufzeit zwar besser ist, als die von Max Swapmengen, der im nächsten Kapitel Vorgestellte ist

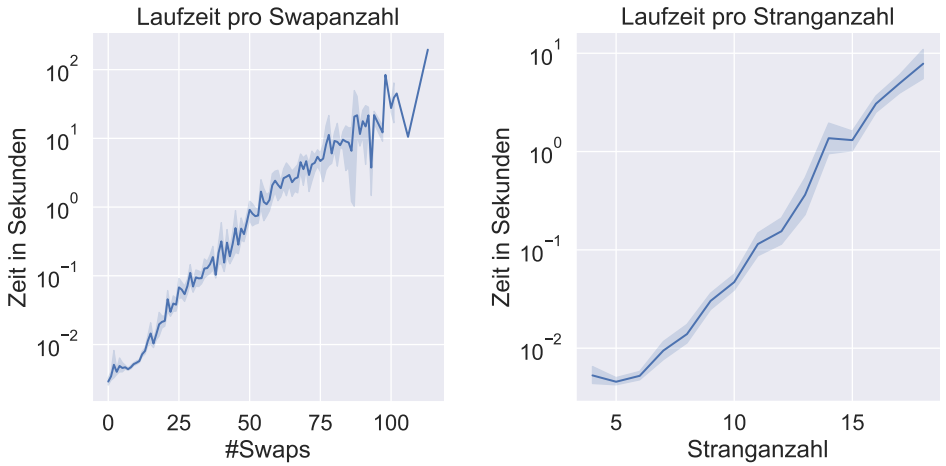

Abbildung 4.2: Hier wird die Laufzeit des SWAPGRAPH Algorithmus einmal der Anzahl an Swaps und einmal der Anzahl an Strängen gegenübergestellt. Man sieht klar, dass die Laufzeit exponentiell steigt. Es ist auch zu erwarten, dass sich beide Graphen ähnlich verhalten, da die Anzahl an Swaps von der Anzahl an Strängen abhängt.

allerdings nochmal deutlich schneller. Wie wir später noch sehen werden, schafft FIRMAN Eingaben bis zu 8 Strängen, in etwa der selben Zeit in der dieser Eingaben mit etwa 18 Strängen schafft, wie man der Abbildung 4.2 entnehmen kann. Nachdem im nächsten Abschnitt ein deutlich besserer Algorithmus gezeigt wird, ist dieser Algorithmus auch auf Grund seiner hohen Komplexität nicht wirklich empfehlenswert. Wir wollen dennoch das Verhalten für verschiedene Eingabegrößen untersuchen. Nach 60 Minuten mit je 200 zufälligen Permutationen für Stränge der Länge 4,5,6,... konnte das Programm Lösungen bis 18 Stränge berechnen (es wurden dieselben Permutationen wie für die anderen später vorgestellten Algorithmen verwendet). Wie in der Abbildung 4.2 zu sehen ist, wächst die Laufzeit sowohl mit der Anzahl an Strängen n als auch mit der Anzahl an Swaps sehr genau exponentiell. Es ist auch zu erwarten, dass sich die beiden Kurven ähnlich verhalten, da mit der Anzahl an Strängen auch die mögliche Anzahl an Swaps steigt. Der Ausschlag zu Beginn ist auf Grund der kurzen Laufzeit wohl nicht ungewöhnlich und wahrscheinlich auf das Aufsetzen der Gurobi-Umgebung, das Bilden des Graphen und diverse Java-spezifische Eigenheiten zurückzuführen.

4.2 Verbesserung des Algorithmus von Firman et al.

Firman et al. haben in ihrem Artikel [FKR+19] bereits einen Algorithmus beschrieben, der immer korrekte Tangles zu einfachen Listen bildet. Ihr Algorithmus baut im Wesentlichen schrittweise einen azyklisch orientierten Graphen, beginnend bei der Startpermutation, bei dem ein Knoten zum nächsten gerichtet wird, wenn die nächste Permutation adjazent ist und Swaps zum ersten mal durchführt. Dieses Verfahren wird an jedem Knoten weitergeführt, bis schlussendlich die Zielpermutation erreicht ist. Daraufhin wird mit einer einfachen Breitensuche der kürzeste Weg von der Start- zu der Zielpermutation gesucht und ausgegeben. Es wird also jede mögliche Zwischenpermutation berechnet, was bis zu $\mathcal{O}(n!)$ viele sind und zu jedem Knoten den maximalen Ausgangsgrad $F(n)$ liefert, also der n-ten Fibonacci-zahl, wie sie gezeigt haben. Dabei ist $F(n) \in \mathcal{O}(\phi^n)$ mit $\phi \approx 1,618$ dem goldenen Schnitt. In der Abbildung 4.3 sieht man den Graphen für die Permutation 4231.

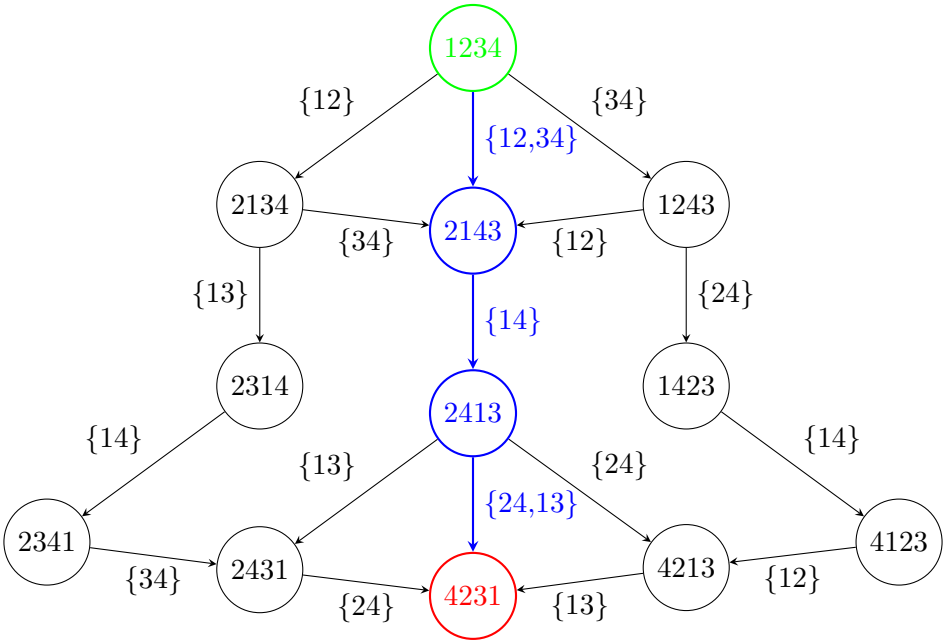

Abbildung 4.3: Der Graph, wie er vom FIRMAN-Algorithmus gebaut wird. In grün und rot sind die Start- und die Zielpermutation markiert. In der Mitte verläuft von oben nach unten der kürzeste Pfad in blau. Die ausgeführten Swaps sind an den Kanten angeschrieben. Alle Kanten, die keine maximalen Swapmengen symbolisieren, führen zu längeren Pfaden. Der verbesserte Algorithmus, der nur noch maximale Mengen verwendet, bildet genau nur den blauen Pfad, was offensichtlich nicht nur weniger Kanten, sondern auch deutlich weniger Permutationen sind.

4.2.1 Verbesserter Algorithmus

In der Problemstellung im 2.Kapitel haben wir bereits im Satz 2.4 gesehen, dass man in jeder Ebene eine maximale Anzahl an Swaps verwenden kann und dennoch sicher einen minimalen Tangle erhält. Wir sagen eine Swapmenge S zu einer Permutation π und Swapliste L ist *maximal*, wenn es keinen weiteren in π benachbarten Swap $s \in L$ gibt, der disjunkt zu allen Elementen in S ist. Anstatt also zu jeder Zwischenpermutation jede adjazente Permutation zu berechnen, berechnen wir nur noch *maximale* adjazente Permutationen (also solche bei der die Swapmenge maximal ist). Dies ergibt eine exponentielle Verbesserung des Algorithmus, wie wir gleich sehen werden.

Lemma 4.1. *Zu einer Permutation π und der Liste L der Länge $n \geq 2$ gibt es maximal $\mathcal{O}(\psi^n)$ maximale Swapmengen, mit* $\psi = \frac{\sqrt[3]{9-\sqrt{69}}+\sqrt[3]{9+\sqrt{69}}}{\sqrt[3]{18}} \approx 1.3245$.

Beweis. Wir zeigen zunächst über vollständige Induktion, dass die maximale Anzahl an ausgehenden Kanten für eine Permutation der Länge $n \geq 2$ mit der Rekursionsgleichung $D(n) \leq D(n-2) + D(n-3)$ beschrieben werden kann. Ohne Beschränkung der Allgemeinheit soll die betrachtete Permutation id_n sein. Sprechen wir vom Element i, ist das im Allgemeinen das Element an der Stelle i. Für unsere Abschätzungen beweisen wir zusätzlich $D(n) \leq D(n-1)$. Beachte, dass die Anzahl an ausgehenden Kanten genau der an maximalen Swapmengen entspricht.
Induktionsanfang $n = 2, 3, 4, 5, 6$:
Wir definieren $D(2) = 1, D(3) = 2, D(4) = 2$ genau so, dass sie dem größten Ausgangsgrad, also der größten Anzahl an maximalen Swapmengen entsprechen und $D(5), D(6)$

n	Permutation	max. Swapmengen	Ausgangsgrad	$D(n)$
2	12	{12}	1	1
3	123	{12},{23}	2	2
4	1234	{12,34}, {23}	2	2
5	12345	{12,34}, {23,45}, {12,45}	3	1+2 = 3
6	123456	{12,34,56}, {23,45}, {12,45}, {23,56}	4	2+2 = 4

Tabelle 4.1: Für den Induktionsbeginn des Lemmas 4.1 sehen wir hier, dass $D(n)$ der maximalen Anzahl an Swapmengen entspricht.

ergeben sich dann richtig, wie in Tabelle 4.1 zu sehen ist. Weiter gilt offensichtlich $D(2) \leq D(3) \leq D(4) \leq D(5) \leq D(6)$.
Induktionsvoraussetzungen
Für $2 \leq k \leq n$ und $n \geq 6$ beliebig aber fest gilt (I) $D(k) \leq D(k-2) + D(k-3)$ und (II) $D(k-1) \leq D(k)$.
Schritt $n \to n+1$:
Wir betrachten nun eine Permutation der Länge $n+1$.

1. Fall $(n, n+1) \notin L$: Damit wird in dieser Permutation $n+1$ nicht vertauscht und der maximale Ausgangsgrad entspricht dem der Permutation id_n und wir erhalten $D(n+1) = D(n)$. Wir erhalten dann die Umformung

$$D(n+1) = D(n) \leq^{\mathrm{I}} D(n-2) + D(n-3) \leq^{\mathrm{II}} D(n-1) + D(n-2).$$

   Sodass beide Induktionsvoraussetzungen nun auch für $n+1$ erfüllt sind.

2. Fall $(n, n+1) \in L, (n-1, n) \notin L$: Wir nehmen also die maximale Anzahl an Swaps der Permutation bis $n-1$, was nach Induktionsvoraussetzung $D(n-1)$ viele sind und hängen den Swap $(n, n+1)$ dran, sodass wir $D(n+1) = D(n-1)$ erhalten. Mit I erhalten wir

$$D(n+1) = D(n-1) \leq^{\mathrm{I}} D(n-3) + D(n-4) \leq^{\mathrm{II}} D(n-1) + D(n-2).$$

   Es gilt außerdem $D(n) = D(n-1) = D(n+1)$, da in diesem Fall nach Voraussetzung $(n-1, n) \notin L$ gilt und somit in der Permutation der Länge n die maximalen Swapmengen dieselben wie für die Permutation der Länge n sind.

3. Fall $(n-1, n), (n, n+1) \in L$: Wir können also zwei Arten von maximalen Swapmengen bilden. Einmal swappen wir $(n, n+1)$ und fügen diesen Swap zu jeder der maximalen Swapmengen von 1 bis $n-1$ hinzu. Und einmal swappen wir den Strang $n+1$ nicht, sondern $(n-1, n)$ und kombinieren diesen Swap wiederum mit jeder maximalen Swapmenge aus 1 bis $n-2$. Damit erhalten wir also insgesamt $D(n+1) = D(n-1) + D(n-2)$. Weiter gilt dann

$$D(n) \leq^{\mathrm{I}} D(n-2) + D(n-3) \leq^{\mathrm{II}} D(n-1) + D(n-2) = D(n+1).$$

Offensichtlich bleiben beide Induktionsvoraussetzungen in jedem Schritt gültig, womit die Behauptung gezeigt ist. Beachte auch, dass alle verwendeten Funktionswerte von D definiert sind, da $n \geq 6$ gilt.
Wir vermuten nun, dass sich $D(n)$ wie auch die Fibonacci Reihe als x^n für ein $x \in \mathbb{R}^+$ schreiben lässt. Wir setzen also $D(n) = x^n$. Durch Einsetzen in die Rekursionsgleichung erhalten wir

$$D(n) = D(n-2) + D(n-3) \Leftrightarrow x^n = x^{n-2} + x^{n-3} \Leftrightarrow x^3 = x + 1 \Leftrightarrow x^3 - x - 1 = 0.$$

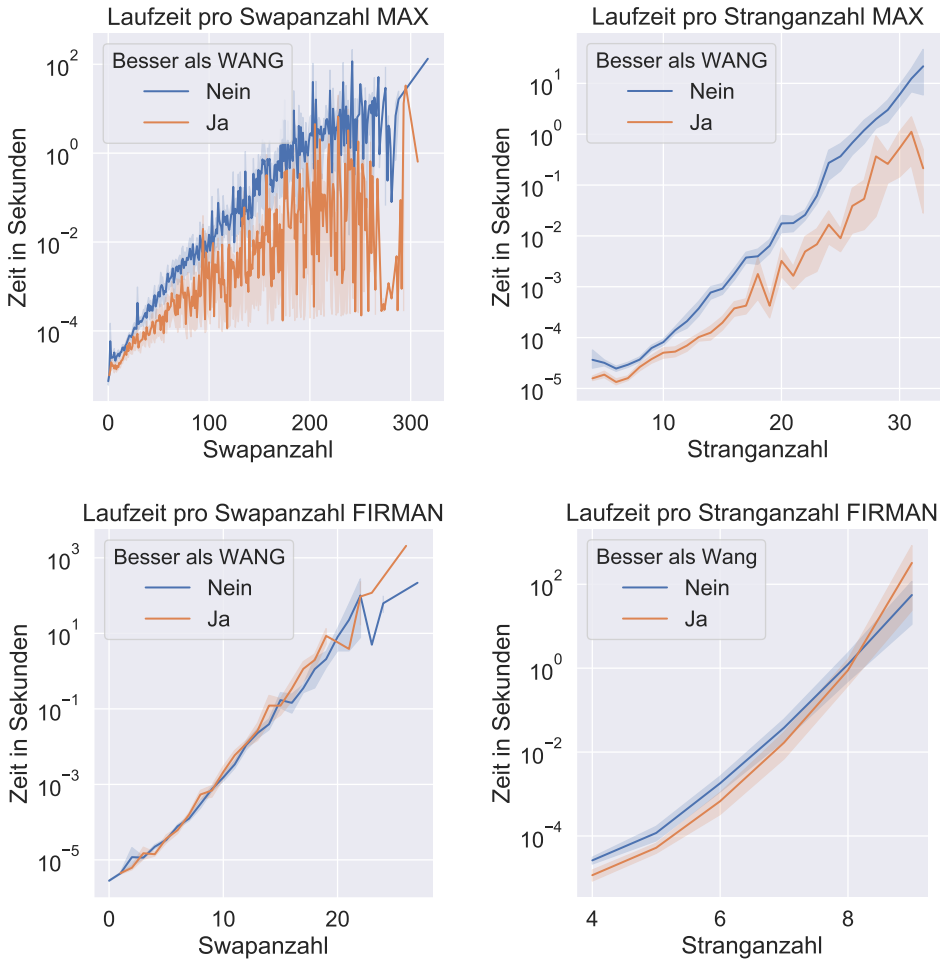

Abbildung 4.4: Wir sehen die Laufzeiten des (oben) und des originalen Firman Algorithmus (unten) in Sekunden. Beide Varianten nutzen die Schranke von Wang und bilden den Baum zweigweise. Links ist die Laufzeit jeweils der Anzahl an Swaps gegenübergestellt, rechts der Anzahl an Strängen. In blau verläuft die Kurve, wenn Wangs Schranke bereits ein optimales Ergebnis berechnet hat, in orange, falls Wangs Schranke zu hoch war.

Wir finden mit Wolfram Alpha [1] die Nullstellen des Polynoms und erhalten zwei komplexe und eine reelle Nullstelle

$$\psi := x_3 = \frac{\sqrt[3]{9-\sqrt{69}}+\sqrt[3]{9+\sqrt{69}}}{\sqrt[3]{18}} \approx 1,3247.$$

Offensichtlich ist nur die reelle Nullstelle für unsere Berechnungen sinnvoll und die Behauptung ist gezeigt. □

In der Abbildung 4.3 sieht man in blau auch zum Vergleich, wie der verbesserte Graph aussieht. Man erkennt bereits, dass in diesem Beispiel deutlich weniger Kanten und Knoten erzeugt werden.

4.2.2 Vergleich

Wir bezeichnen den gerade vorgestellten Algorithmus als Max Swapmengen. Die Algorithmen Max Swapmengen und Firman wurden auf zwei verschiedene Arten implementiert,

[1] (https://www.wolframalpha.com/)

einmal wird der komplette Baum schichtweise gebaut (nur mit maximalen adjazenten Permutationen) und abgebrochen, sobald eine Lösung gefunden wurde, sodass man sich gelegentlich Pfade sparen kann, wenn eine Permutation bereits in derselben oder einer darüberliegenden Schicht erreicht wurde. Dies belastet allerdings den Arbeitsspeicher schnell, weswegen auch eine Variante implementiert wurde, in der immer einzelne Zweige bis zu WANGs Schranke oder einer gültigen Endpermutation verfolgt werden. Hierbei können manche Pfade nicht weiter ausgeschlossen werden, wenn sie bereits vorher gefunden wurden, dafür ist der Arbeitsspeicher nicht überlastet. Außerdem gibt es hier einen deutlichen Vorteil: Wenn WANG nicht optimal ist, können sowohl FIRMAN als auch MAX SWAPMENGEN abbrechen und sofort das Ergebnis zurückgeben, sobald die Endpermutation erreicht wurde und die Tanglehöhe kleiner als in WANGs Lösung ist (da WANG ja maximal um 1 falsch ist). Mit Glück müssen also nur wenige Pfade abgesucht werden. Empirisch gesehen ist WANG mit 37% relativ häufig falsch, wie man der Tabelle 4.2 entnehmen kann. Deswegen haben wir wahrscheinlich in 37% der Fälle (unter der Annahme, dass die Fehlerrate für höhere m gleich bleibt) eine deutlich kürzere Laufzeit im Vergleich dazu wenn der ganze Baum gebaut wird (Faktor 100-1000 bei Beispielen mit 30 Strängen), jedoch eine um etwa Faktor 5 langsamere sonst (da wir bei Gleichheit den kompletten Baum bauen müssen). Die Laufzeit ist also am besten, wenn WANG falsch liegt.
Um nun den alten FIRMAN und den neuen MAX SWAPMENGEN zu vergleichen geben wir beiden 60 Minuten Zeit so große Permutationen wie möglich zu lösen. Wir beginnen bei $n = 4, 5, 6, \ldots$ Strängen und lassen die Algorithmen jeweils 200 zufällig generierte Permutationen abarbeiten. Wir verwenden für die Berechnungen dieselben Permutationen. Betrachte für die Ergebnisse die Abbildung 4.4. Die Algorithmen sind sehr gut vergleichbar, der einzige Unterschied liegt darin, dass MAX SWAPMENGEN einen deutlich kleineren Graphen baut. Wie man erkennen kann, ist für FIRMAN bei $n = 9$ die Laufzeit bereits sehr hoch, während MAX SWAPMENGEN im Bruchteil einer Sekunde fertig wird. MAX SWAPMENGEN schafft alle Beispiele bis $n = 31$ und noch ein Paar für $n = 32$ Stränge in 60 Minuten, was eine sehr deutliche Verbesserung ist. Es scheint, dass die maximale Anzahl an Knoten im Graphen von MAX SWAPMENGEN mit $\mathcal{O}(n!)$ zu hoch abgeschätzt ist, denn die Kurve verläuft relativ schön exponentiell, während für den FIRMAN die Kurve anscheinend leicht überexponentiell steigt, dies wäre eventuell noch genauer zu untersuchen. Für MAX SWAPMENGEN sieht man klar, dass durch WANGs Schranke eine deutliche Verbesserung zu verzeichnen ist, was bei FIRMAN nicht der Fall ist. Dies liegt einfach an der Implementierung, denn die ausgehenden Kanten werden der Reihe nach abgearbeitet und es wird immer von kleinen zu großen Swapmengen durch iteriert, sodass beinahe immer fast der komplette Baum gebaut wird. Bei MAX SWAPMENGEN kann dieses Problem nicht auftreten, da nur maximale Swapmengen untersucht werden. Vielleicht ergibt sich für MAX SWAPMENGEN jedoch noch eine Verbesserung, wenn wir die ersten Baumabstiege mit den größten maximalen Swapmenge beginnen. Im unteren rechten Graph überholt FIRMAN für Werte bei denen WANG falsch lag sogar den anderen, was wohl nur ein Zufall ist, da für 9 Stränge nur noch wenige Beispiele berechnet wurden und die Laufzeit für einzelne sich stark unterscheiden können.

4.3 Ein Greedy-Algorithmus

Wir betrachten Tangles und ihre Höhe nochmals etwas genauer. Wenn wir einen Strang x haben, der insgesamt noch mit drei anderen Strängen swappen muss, so können wir sagen, dass der Tangle mindestens die Höhe 3 haben muss, da der Strang x in jeder Schicht nur maximal einmal vertauscht werden kann. Das führt uns zu folgender Beobachtung.

Beobachtung 4.2. *Sei l_i für $1 \leq i \leq n$ die Anzahl an Swaps, an denen der Strang i in einem Tangle teilnimmt. Dann gilt für die Höhe h jedes resultierenden Tangles $h \geq l_i$ für alle i.*

Algorithm 4.1: BUILDPRIORITYTANGLE

```
Input: Permutation P
Data: WorkPermutation W
Output: A correct tangle T of P.

// Initialization
1 WorkPermutation W = id_n
2 W.setDestinationPermutation(P)

// Build tangle
3 while W.hasSwaps() do
      // sort all swaps that can be performed on the workPermutation
4     possibleSwaps = W.getPossibleSwapsSorted()

      // fill the layer
5     forall Swaps s ∈ possibleSwaps do
6         layer.addIfDisjoint(s)

      // update the workpermutation
7     W.swapAll(layer)
8     T.add(layer)
```

Wir bezeichnen zu einem Strang $i, 1 \leq i \leq n$ die *Priorität* $p(i) : [n] \to \{0, \ldots, n-1\}$ als die Anzahl an Swaps, an denen der Strang im (restlichen) Tangle noch beteiligt ist. Die Idee des folgenden Algorithmus ist nun möglichst früh die priorisierten Swaps abzuarbeiten.

4.3.1 Der Algorithmus

Die Zahl $n-1$ ist eine obere Schranke zur Priorität, da ein Strang maximal mit jedem anderen Strang swappen kann. Wir bauen nun schichtweise den Tangle wie folgt auf. Wir berechnen zu jedem Strang die Priorität, erzeugen dann eine Liste mit allen möglichen Swaps und sortieren diese absteigend nach der Priorität der Swaps. Dabei zählt zuerst die höchste Priorität, dann die zweithöchste. Nun iterieren wir durch die erzeugte Liste und fügen der Schicht jeden Swap hinzu, der disjunkt zu den bereits eingefügten ist. Damit erhalten wir eine maximale Schicht, die die priorisiertesten Swaps enthält, siehe auch den Algorithmus 4.1.
Da hier viele externe Methoden aufgerufen werden, wollen wir deren Funktionsweise genauer untersuchen – wobei die Namen bereits eine klare Idee vermitteln dürften. Die eingegebene Permutation kann etwa als Integer-Array übergeben werden, während die *WorkPermutation* ein eigens erstellter Datentyp ist. Dieser besitzt eine aktuelle Permutation und eine Zielpermutation (*destinationPermutation*), sodass eine Menge an benachbarten Swaps leicht berechnet werden kann. Die möglichen Swaps werden dann in einer Liste *possibleSwaps* gespeichert und sortiert. Wie oben erwähnt sind die Werte der Priorität eingeschränkt mit $0 \leq p(i) \leq n-1, i = 1 \ldots n$, sodass sich für das Sortieren linearzeit-Algorithmen eignen, wie Counting-Sort oder auch Bucket-Sort [CLRS09]. Die Swaps werden erst nach höchster und bei Gleichheit nach niedrigster Priorität sortiert. In der Schleife in Zeile 5,6 werden also aus allen möglichen Swaps alle höchst priorisierten Swaps hinzugefügt, wenn sie disjunkt zu allen in der Schicht eingefügten Swaps sind. Bevor die Schicht dem Tangle hinzugefügt wird, muss noch die *WorkPermutation* auf den neusten Stand gebracht, also alle Swaps durchgeführt und die Prioritäten geändert werden.

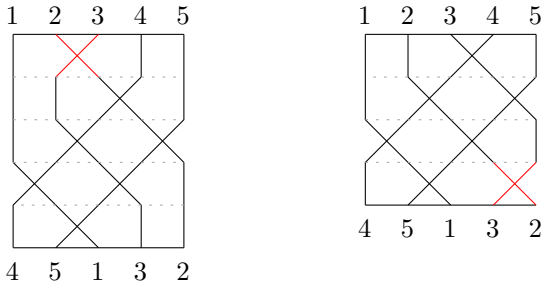

Abbildung 4.5: Links sehen wir die Lösung des greedy Algorithmus. Da die Prioritäten der Swaps 23 und 34 identisch sind, wählt der Algorithmus einfach den ersten, was aber zu einem größeren Tangle führt. In rot ist der verschobene Swap markiert.

4.3.2 Die Laufzeit

Wir bezeichnen diesen Algorithmus als Priority. Wir wollen nun die Laufzeit genau anhand des gerade beschriebenen Algorithmus untersuchen. Das bilden der id_n Permutation geht offensichtlich in $\mathcal{O}(n)$ Zeit und die Zuweisung in der 2. Zeile in konstanter Zeit. Die while-Schleife in Zeile 3 läuft maximal $\mathcal{O}(h)$ mal, mit h als Höhe des resultierenden Tangles, da jedes mal eine Schicht hinzugefügt wird und dann die Höhe des Tangles die Abbruchsbedingung der While-Schleife ist. Das Finden aller möglichen Swaps in Zeile 4 funktioniert in $\mathcal{O}(n)$ Zeit, indem wir eine boolesche Matrix mitführen, die zu jedem Swap speichert, ob dieser noch ausgeführt werden muss und dann alle benachbarten Swaps in der Permutation durchiterieren und hinzufügen, wenn der Swap noch gemacht werden muss. Da wir linearzeit-Sortierverfahren verwenden, funktioniert auch das Sortieren in $\mathcal{O}(n)$ Zeit. Die innere Schleife in Zeile 6 läuft maximal $n-1$ mal, da es nicht mehr benachbarte Swaps in einer Permutation der Länge n geben kann. Die Methode AddIfDisjoint() läuft in konstanter Zeit, wenn wir ein boolesches Array der Länge n mitführen in welchem gespeichert wird, welche Stränge bereits geswapped wurden. Zeile 8 macht eine konstante Vertauschoperation für jedes der maximal n Elemente in der Schicht und ist damit auch in $\mathcal{O}(n)$. Es ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(hn)$. Es scheint sinnvoll zu sein $h \in \mathcal{O}(n)$ anzunehmen. Zum einen hat Wang [Wan91] bereits bewiesen, dass die Höhe eines optimalen Tangles maximal $n+1$ beträgt, was auf Grund der hohen Korrektheit von Priority intuitiv darauf schließen lässt, dass die Abweichung nicht zu hoch sein sollte und auch empirisch ergibt sich für hohe Stranganzahlen, dass die resultierende Höhe nur um wenige Ebenen größer ist, als die Anzahl an Strängen. Weiter ließe sich der Algorithmus auch so anpassen, dass er sicher in $\mathcal{O}(n^2)$ Zeit läuft, indem wir bei $n+1$ Schritten abbrechen und stattdessen Wang laufen lassen.

4.3.3 Korrektheit

Offensichtlich erhalten wir immer einen Tangle, da wir schichtweise nur disjunkte Swaps hinzufügen, die in der Zwischenpermutation benachbart waren. Damit sind alle Zwischenpermutationen adjazent. Außerdem werden alle Swaps hinzugefügt – das ist genau die Bedingung in der While-Schleife.

Interessanter ist nun die Frage, wie klein die Tangles sind. Der Algorithmus liefert nicht immer ein minimales Ergebnis. Betrachte beispielsweise die Permutation 45132. In der ersten Ebene können die Swaps 23 und 34 durchgeführt werden, die nicht disjunkt sind. Die Prioritäten beider sind gleich mit $p(2) = p(3) = p(4) = 3$ und somit wählt der Algorithmus den vorderen Swap 23, was zu einem zu hohen Tangle führt, wie in Abbildung 4.5 zu sehen ist. Für größere Permutationen erhalten wir auch größere Abweichungen als nur eins vom Optimum, zum Beispiel für die Permutation [6, 9, 10, 8, 7, 3, 1, 5, 4, 2].

# Stränge	# Wang falsch	# Prio falsch	% Wang falsch	% Prio falsch
7	81	14	0,405	0,07
9	78	14	0,39	0,07
11	73	31	0,365	0,155
13	82	18	0,41	0,09
15	78	24	0,39	0,12
17	84	23	0,42	0,115
19	81	21	0,405	0,105
21	82	17	0,41	0,085
23	94	18	0,47	0,09
25	85	17	0,425	0,085
27	89	15	0,445	0,075
29	89	10	0,445	0,05
31	86	11	0,43	0,055

Tabelle 4.2: Hier wurden zu jeder Stranganzahl 200 zufällig erzeugte Permutationen mit dem verbesserten Firman-Algorithmus in der Tiefensuche Variante gelöst. Aufgeführt ist die Korrektheit des Algorithmus von Wang und des Prioritäts-Algorithmus für die verschiedenen Eingabegrößen. Es ergab sich eine Korrektheit von etwa 41% bei WANG . Insgesamt waren von 2600 Beispielen bei WANG 1082 fehlerhaft (=41,62%), beim Prioritäts-Algorithmus 233 von 2600 (= 8,96%). Die Wahrscheinlichkeit, dass PRIO falsch ist, gegeben, dass WANG falsch liegt, ist etwa dieselbe mit 8,23% (sogar eher weniger). Die gleiche Beobachtung gilt auch für gerade Stranganzahlen.

4.3.4 Vergleich mit Wangs Algorithmus

Wir haben bereits den parallelen Bubblesort-Algorithmus von Wang kennengelernt, der nicht immer ein optimales Ergebnis liefert, aber in $\mathcal{O}(n^2)$ Zeit läuft und Ergebnisse liefert, die maximal um eine Schicht zu groß sind. Wie WANG läuft PRIORITY in $\mathcal{O}(n^2)$ Zeit. Wir wollen nun testen, welcher Algorithmus denn in der Praxis bessere Ergebnisse liefert.
Dafür bilden wir zu jeder Stranganzahl n 200 zufällig erzeugte Permutationen. Wir vergleichen die Größen der Lösungen des Prioritäts-Algorithmus und der von Wang mit den Lösungen des immer optimalen MAX SWAPMENGEN in Bereichen von $n = 6, ..., 31$ Strängen, wie der Tabelle 4.2 zu entnehmen ist (aufgeführt sind nur ungerade Stränge). Die wichtigsten Beobachtungen sind, dass WANG etwa 37 % falsch lag (für gerade Eingaben 31 %, für ungerade 41 %), während der Prioritäts-Algorithmus nur etwa 9 % der Zeit zu hoch war. WANG bietet jedoch den Vorteil, dass die Lösungen maximal um eine Ebene zu groß werden, was bei PRIO nicht der Fall ist (etwa 20 % der falschen Lösungen sind um mehr als eins höher, wobei die Wahrscheinlichkeit für höhere Abweichungen stetig deutlich kleiner wird; bei unseren Tests gab es maximal 3 Ebenen zu viel). Weiter sind für die getesteten Stranganzahlen die Ereignisse eines falschen Ergebnisses von WANG und PRIO anscheinend unabhängig. Führen wir auf einer Permutation also beide Algorithmen einmal aus und nehmen den kleineren Tangle, so haben wir zu etwa 97 % einen optimalen Tangle, der weiter maximal eine Schicht zu viel hat. Die Frage ist natürlich, ob sich die Wahrscheinlichkeiten für größere Permutationen so fortsetzen, da die Daten doch nur für vergleichsweise kleine Permutationen mit einer sicher optimalen Lösung verglichen werden können. Interessant ist dafür, dass man bei WANG eher einen steigenden Trend für höhere Eingabegrößen beobachten kann, bei PRIO sinkt die Wahrscheinlichkeit.
Es bleibt zu erwähnen, dass PRIO im Vergleich zu WANG in der Praxis etwa 200 mal langsamer läuft. Wobei der Vergleich eventuell nicht ganz fair ist. Beide wurden zwar

in Java implementiert, der Prioritäts-Algorithmus jedoch rekursiv und insgesamt wohl auch nicht maximal effizient. Wie man aber am Code 4.1 schon sieht, ist der Algorithmus deutlich komplexer als Wangs, sodass auch für schnellere Implementierungen ein Faktor im Bereich 100 realistisch klingt.

4.3.5 Zusammenfassung und weiterführende Ideen

Der Prio Algorithmus ist nicht immer korrekt, auch wenn er in Kombination mit Wang eine deutliche Verbesserung zum schnellen Finden von guten Lösungen darstellt, wie im vorhergehenden Abschnitt erwähnt. Interessant bleibt die Frage ob der Algorithmus optimale Ergebnisse liefert, wenn man jedes mal, wenn zwei Prioritäten gleich sind, beide Möglichkeiten berechnet. Dies ergäbe einen wesentlich kleineren Baum als bei Firman, da wir sowieso nur maximale Swapmengen betrachten und weiter mit der Priorität einige Pfade ausschließen. Man müsste dann allerdings noch prüfen, ob damit nicht bereits zu viele Pfade ausgeschlossen werden, sodass nicht immer eine Optimallösung gefunden wird. Man könnte auch verschiedene Härtegrade des Abstiegs anwenden: Eine Stellschraube wäre noch ausschließlich für die höchste Priorität mehrere Verzweigungen zu erlauben; wenn niedrigere Prioritäten mehrfach belegt sind, bildet man dennoch nur eine beliebige maximale Swapmenge. Zum Anderen wäre interessant ob es sinnvoll ist auch nach der kleineren Priorität im Swap zu sortieren oder ob man, sobald irgendwelche zwei Prioritäten gleich sind beide Pfade durchrechnet.

5. Zusammenfassung und Ausblick

Inspiriert vom Artikel von Firman et al. [FKR+19] untersuchten wir also, ob es einen polynomiellen Algorithmus zum Finden von höhenminimalen Tangles zu einfachen Listen gibt. Wir untersuchten zuerst grundlegend, wie Tangles aufgebaut sind. Dabei fanden wir im ersten Satz 2.4, dass nur maximale Swapmengen zum Bilden von Tangles betrachtet werden müssen. Diese Einsicht führte schlussendlich im letzten Kapitel auch zum schnellsten korrekten Algorithmus Max Swapmengen, der empirisch Permutationen mit etwa 32 Strängen in akzeptabler Zeit auf dem genutzten Rechner löste (circa 10 Sekunden), wogegen der alte Firman bereits bei neun Strängen teilweise länger als eine Stunde benötigte. Während auch der neue Algorithmus fast denselben Graphen wie Firman baut, wie er ursprünglich in ihrem Artikel [FKR+19] vorgestellt wurde, konnte auch hier die Arbeitsweise etwas verbessert werden, indem wir für den Pfadabstieg im Lösungsbaum mit Wangs Algorithmus eine obere Schranke für den optimalen Tangle definierten, sodass wir die Suche vorzeitig abbrechen konnten, wenn die durch Wang berechnete Lösung falsch ist. Das verringert empirisch in etwa 37 % der Fälle die Laufzeit des Algorithmus um einen Faktor ungefähr zwischen 100 und 1000. Wir konnten im Lemma 4.1 auch zeigen, dass unser Algorithmus Max Swapmengen in $\mathcal{O}(\psi^n n!)$ Laufzeit exponentiell besser ist als Firman, welcher eine Laufzeit in $O(\phi^n n!)$ hat, mit $\phi \approx 1,618, \psi \approx 1,325$. Da auch offensichtlich bei Max Swapmengen für $n \geq 4$ in keinem Fall alle $n!$ Zwischenpermutationen im Graphen erzeugt werden, bleibt die Frage, ob womöglich auch dieser Faktor in der Laufzeit verringert ist.
In einem weiteren Ansatz übertrugen wir das Problem auf den Swapgraphen, welcher Swaps als Knoten hat und zwischen zwei Knoten eine Kante hat, wenn beide einen gleichen Strang haben. Wir fanden heraus, dass jede geordnete und azyklische Orientierung einem Tangle entspricht und insbesondere im Satz 3.13, dass jede zyklische und geordnete Orientierung einen Zyklus der Länge 3 enthält. Betrachten wir das Tangle-Height Minimization Problem mit Swapgraphen, so gilt es, eine azyklische und geordnete Orientierung mit minimalem längsten Pfad zu finden. Aus den Regeln für die geordnete Orientierung können wir ableiten, dass wir zu jedem Swap-Tripel im Graphen eine binäre Entscheidung treffen können, in welche dieses gerichtet ist. Die Anzahl an Swap-Tripel liegt jedoch in $\mathcal{O}(n^2)$, sodass die Anzahl an möglichen Kombinationen in $\mathcal{O}(2^{n^2})$ liegt. Tatsächlich hängen jedoch die meisten Swap-Tripel voneinander ab, sodass durch das Richten von wenigen Tripeln weitere Richtungen schnell determiniert sind – etwa, um einen Zyklus zu vermeiden.
Wir lösten im letzten Kapitel 4.1 das Problem, eine geordnete und azyklische Orientierung mit minimalem längsten Pfad zu finden, indem wir es auf ein ILP überführten und vom externen Programm Gurobi [GO20] lösen ließen. Dieser Ansatz lieferte in der Praxis von der

Laufzeit her ein deutlich besseres Ergebnis als Firman, war aber wieder deutlich schlechter als Max Swapmengen, sodass er auch auf Grund hohen Implementierungsaufwandes nicht wirklich empfehlenswert scheint.
Mit unserem als drittem vorgestellten Algorithmus Prio fanden wir eine gute Heuristik, die uns durch das Priorisieren von Strängen, welche noch viele Swaps durchzuführen haben, in etwa 91 % der Fälle einen korrekten Tangle lieferte. Zusammen mit Wang konnten wir einen kombinierten Algorithmus angeben, der in einer Laufzeit in $\mathcal{O}(n^2)$ in 97 % der Fälle einen korrekten und ansonsten einen maximal um eine Schicht zu hohen Tangle als Ergebnis liefert. Für die Praxis sei allerdings angemerkt, dass dieser Algorithmus etwa 100-mal langsamer ist als der originale Algorithmus von Wang, sodass man hier vielleicht weiter abwägen sollte.
Weiterführend könnte auch untersucht werden, ob es bereits ähnliche gelöste Probleme wie das Finden einer Orientierung mit kürzestem Pfad unter gewissen Einschränkungen gibt. Eventuell ließen sich diese Lösungsverfahren auch hier anwenden. Das Problem, für einen allgemeinen Graphen eine minimale Orientierung zu finden, ist NP-schwer, da daraus – nach dem Satz von Gallai–Hasse–Roy–Vitaver [Xu01] – die chromatische Zahl einfach abgeleitet werden kann, was bekannterweise NP-vollständig ist [GJ78].

Literaturverzeichnis

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest und Clifford Stein: *Introduction to Algorithms, Third Edition*, 2009.

[FFK+20] Oksana Firman, Stefan Felsner, Philipp Kindermann, Alexander Ravsky, Alexander Wolff und Johannes Zink: *The Complexity of Finding Tangles*, 2020. https://arxiv.org/abs/2002.12251.

[FKR+19] Oksana Firman, Philipp Kindermann, Alexander Ravsky, Alexander Wolff und Johannes Zink: *Computing Height-Optimal Tangles Faster*. In: Daniel Archambault und Csaba D. Tóth (Herausgeber): *Graph Drawing and Network Visualization - 27th International Symposium, GD 2019, Proceedings*, Band 11904 der Reihe *Lecture Notes in Computer Science*, Seiten 203–215. Springer, 2019. https://doi.org/10.1007/978-3-030-35802-0_16.

[GJ78] M. Garey und D. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness. GT5*. 1978.

[GO20] LLC Gurobi Optimization: *Gurobi Optimizer Reference Manual*, 2020. http://www.gurobi.com.

[OMK+18] Maya Olszewski, Jeff Meder, Emmanuel Kieffer, Raphaël Bleuse, Martin Rosalie, Grégoire Danoy und Pascal Bouvry: *Visualizing the Template of a Chaotic Attractor*. In: Therese Biedl und Andreas Kerren (Herausgeber): *Graph Drawing and Network Visualization*, Seiten 106–119, Cham, 2018. Springer International Publishing, ISBN 978-3-030-04414-5.

[Wan91] Deborah C. Wang: *Novel Routing Schemes for IC Layout, Part I: Two-Layer Channel Routing*. In: A. Richard Newton (Herausgeber): *Proceedings of the 28th Design Automation Conference*, Seiten 49–53. ACM, 1991. https://doi.org/10.1145/127601.127626.

[Xu01] Junming Xu: *Interconnection Networks and Graphs*, Seiten 129–130. Springer US, Boston, MA, 2001, ISBN 978-1-4757-3387-7. https://doi.org/10.1007/978-1-4757-3387-7_1.