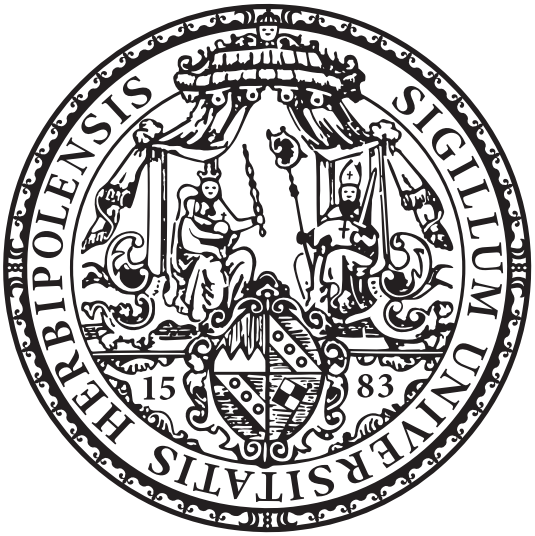
Bachelorarbeit

Hinreichende und notwendige Voraussetzungen für die Realisierbarkeit eines Tangles

Florian Timpf

Abgabedatum: 09. Mai 2022
Betreuer: Prof. Dr. Alexander Wolff
Oksana Firman, M. Sc.



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

Zusammenfassung

In dieser Arbeit untersuchen wir das folgende kombinatorische Problem: Ist eine Menge von n y-monotonen *Kabeln* gegeben, so ist ein *Tangle* eine Folge von Permutationen, sodass sich die Reihenfolge der Kabel zwischen zwei aufeinanderfolgenden *Schichten* des Tangles nur in disjunkten Vertauschungen benachbarter Kabel unterscheidet. Gegeben eine Multimenge L, auch *Liste* genannt, an ungeordneten Paaren der Zahlen 1 bis n, sogenannten *Swaps*, und einer Ausgangspermutation, so *realisiert* ein Tangle L, falls jedes Paar von Kabeln genau so häufig getauscht wird, wie es in L vorkommt. Wir beschränken uns in dieser Arbeit hauptsächlich auf *einfache* Listen, d.h. alle Swaps kommen höchstens einmal vor. Das Ziel ist es für einfache Listen, Tangles von möglichst geringer *Höhe*, also Anzahl an Schichten, zu erzeugen. Wir zeigen, dass es hierfür ausreichend ist *minimal unabhängige* Listen zu betrachten und analysieren für diese Listen die Höhenoptimalität der Tangles eines bereits existierenden Algorithmus. Weiter geben wir zwei Greedy-Algorithmen an, welche wir implementiert haben, und analysieren sie. Schließlich vergleichen wir unsere Algorithmen mit dem bekannten.

Abstract

We study the following combinatorial problem: Given a set of n y-monotone *wires*, a *tangle* is the sequence of permutations such that the order of wires of two consecutive layers differs only in disjoint permutations of neighboring wires. We further define a *swap*, i.e. an unordered pair of numbers between 1 and n. A tangle *realizes* a multiset L of swaps, also called a *list*, if each pair of wires changes its order the exact number of times as specified in L. In this paper, we mostly restrict ourselves to so-called *simple* Lists, containing each swap at most once. The goal of this paper is to find tangles of minimal height for these simple lists. We show that we can restrict the choice of lists to *minimal independent* ones and analyze how often we get tangles of optimal height for an already existent algorithm. We further provide two greedy-algorithms which we implemented and analyze them as well. Finally, we compare our algorithms to the already known.

Inhaltsverzeichnis

1 Einleitung **4**
1.1 Grundbegriffe und Notation 4
1.2 Eigener Beitrag 6

2 Allgemeine Eigenschaften von Listen **7**

3 Odd-Even-Sort **11**
3.1 Verbesserung von Odd-Even-Sort 19
3.1.1 Greedy-Tangler 19
3.1.2 Modified-Odd-Even-Sort 23
3.2 Vergleich der Algorithmen 27

4 Fazit **29**

5 Ausblick **30**

Literaturverzeichnis **32**

1 Einleitung

Wir betrachten eine visuelle Repräsentation von in chaotischen dynamischen Systemen vorkommenden, sogenannten chaotischen Attraktoren. Diese kommen unter anderem in der Meteorologie, Physik und Medizin vor. Ein klassisches Beispiel für ein chaotisches System ist das Doppelpendel, welches bereits bei geringen Abweichungen in den Anfangsbedingungen eine stark abweichende Trajektorie besitzen kann. Hier ist der chaotische Attraktor die Ruhelage des Pendels. Auch etwa die Populationsdynamik von Räuber-Beute-Modellen ist ein chaotisches System, der Attraktor ist hier häufig ein Zyklus.

Schon 1983 erwähnten Birman und Williams [BW83] *Tangles* als Möglichkeit die topologischen Strukturen von chaotischen Attraktoren zu beschreiben. 1990 zeigten Mindlin et al. [MHS$^+$90] wie sich chaotischen Attraktoren mithilfe von ganzzahligen Matrizen, sogenannten *Listen*, beschreiben lassen, deren Einträge die Anzahl von *Swaps* zwischen den Orbits eines Attraktors entsprechen. Olszewski et al. [OMK$^+$18] gaben folgenden Rahmen für die Visualisierung: Ist eine Menge an *Kabeln* gegeben, welche senkrecht in geordneter Form an einer horizontalen Linie hängen, sowie eine Multimenge an Swaps, so ist ein Tangle die Visualisierung dieser Swaps, also gerade die Ordnung in welcher diese Swaps durchgeführt werden, wobei nur benachbarte Kabel geswappt werden und disjunkte Swaps gleichzeitig geschehen dürfen. Für ein Beispiel siehe Abbildung 1.1. Zudem gaben sie einen Exponentialzeit-Algorithmus zum Finden von Tangles zu beliebigen *Listen* an, welche Firman et al. [FKR$^+$19] formalisierten und den Algorithmus verbesserten.

Für den Fall, dass jeder Swap nicht mehr als einmal vorkommt, zeigten Sado und Igarashi [SI87], dass mithilfe einer parallelisierten Variante des Sortieralgorithmus Bubblesort namens Odd-Even-Sort ein Tangle mit höchstens einer Höhe von eins mehr als dem Minimum generiert werden kann.

1.1 Grundbegriffe und Notation

Wir schreiben für die Menge $\{1, \ldots, n\}$ auch kurz $[n]$. Eine Bijektion von der Menge $[n]$ auf sich selber wird *Permutation* genannt. Die symmetrische Gruppe S_n ist die nichtabelsche Gruppe, also eine Gruppe für die das Kommutativgesetz nicht gilt, d.h. im Allgemeinen gilt $f(g(n)) \neq g(f(n))$, der Permutation auf $[n]$, deren Gruppenoperation die Komposition $\circ$ der Permutationen ist und deren neutrales Element die identische Abbildung $\pi_{id} = id$ ist. Für alle $\pi, \sigma \in S_n$ gilt insbesondere $\pi \circ \sigma(i) = \pi(\sigma(i))$ mit $i \in [n]$. Wir notieren $\pi \in S_n$ als die Folge der inversen Permutationen $\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(n)$. Seien also zum Beispiel $\pi(1) = 4, \pi(2) = 2, \pi(3) = 5, \pi(4) = 6, \pi(5) = 3, \pi(6) = 1$, so erhalten wir $\pi = 625134$. Wir schreiben zudem $S_{n,2}$ für die Menge der Permutationen der Ordnung 2. Es ist also $\pi \in S_{n,2}$ genau dann, wenn $\pi \circ \pi = id$.

Die Permutation die i mit j tauscht, $i, j \in [n]$, $i \neq j$, jedoch keine anderen Elemente, nennen wir einen *Swap* oder *Tausch* und schreiben hierfür $\pi(i, j)$ oder (i, j). Eine Menge $S = \{\pi_1, \ldots, \pi_k\}$, $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$, heißt *zulässig*, wenn alle Elemente in $[n]$ höchstens durch einen Swap π_i getauscht werden. Sei S eine zulässige Menge von Swaps. Dann ist das Produkt $\prod S$ unabhängig von der Reihenfolge der Faktoren und liegt in $S_{n,2}$. Zugleich existiert für jedes $\pi \in S_{n,2}$ eine zulässige Menge $S(\pi)$ von paarweise disjunkten Swaps, sodass $\pi = \prod S(\pi)$ ist.

Eine Permutation $\pi \in S_n$ *stützt* eine Permutation $\sigma \in S_{n,2}$, falls i und j in der Folge π für jeden Swap $(i, j) \in S(\sigma)$ benachbart sind.

Wir nennen zwei Permutationen π, σ *benachbart*, wenn eine eine Permutation $\varepsilon \in S_{n,2}$ existiert, sodass ε von π gestützt wird und $\sigma = \pi\varepsilon$. In diesem Fall wird ε ebenfalls von σ gestützt, da mit $\varepsilon \in S_{n,2}$ die Gleichung $\sigma\varepsilon = \pi\varepsilon\varepsilon = \pi$ gilt. Ein *Tangle* der *Höhe* h ist eine Folge $\langle \pi_1, \pi_2, \ldots, \pi_h \rangle$ von Permutationen, sodass je zwei Permutationen π_i und π_{i+1} für $i \in [n]$ benachbart sind. Ein Tangle entspricht also einer Folge von h *Schichten*, welche jeweils zulässige Mengen von Swaps sind. Weiter nennen wir eine Folge $\langle \pi_k, \pi_{k+1}, \ldots, \pi_l \rangle$ $1 \leq k \leq l \leq h$ aufeinanderfolgender Permutationen ein *Teiltangle* von T.

Wir nennen eine y-monotone Kurve ein *Kabel*, welches von einer horizontalen Geraden herabhängt. Gegeben n Kabel, so ist eine *Liste* $L = (l_{ij})$ der Dimension n eine symmetrische $n \times n$-Matrix mit nicht-negativen Einträgen sowie $l_{ii} = 0$, für $i \in [n]$. Ist L eine Liste, so lässt sie sich auch als Menge von Swaps schreiben. Ist $l_{ij} \leq 1$, so nennen wir L *einfach*. Sollten alle Einträge von L gerade sein, so nennen wir L *gerade*[1]. Die *Länge* $|L|$ einer Liste L erhalten wir, indem wir alle Einträge l_{ij} mit $i < j$ aufsummieren, also $|L| = \sum_{i<j} l_{ij}$. Für ein Tangle T definieren wir die symmetrische $n \times n$-Matrix $L(T) = (l_{ij})$ so, dass l_{ij}, mit $1 \leq i < j \leq n$, der Häufigkeit des (i, j)-Swaps entspricht. Offensichtlich gilt $l_{ii} = 0$ für jedes $i = [n]$, da ein Kabel nicht mit sicher selber swappen kann. Zudem sagen wir, dass T die Liste $L(T)$ *realisiert*.

Wie in Abbildung 1.1 zu sehen ist, können für zwei verschiedene Tangles T_1 und T_2 die Listen $L(T_1)$ sowie $L(T_2)$ gleich sein. Die Abbildung zeigt auch, dass beide Tangles die Liste L realisieren.

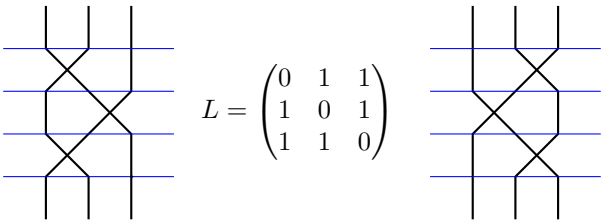


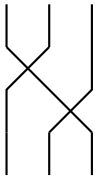
Abb. 1.1: Beispiel für eine realisierbare Liste mit zwei verschiedenen Tangles

Wir nennen eine Liste L *π-realisierbar*, wenn sie durch ein Tangle beginnend mit der Permutation π realisiert wird. Gilt $\pi = \pi_{id}$, so nennen wir L *realisierbar*. Existiert

[1] Der Vollständigkeit halber möchten wir darauf hinweisen, dass es auch ungerade Listen gibt, das sind solche deren Nicht-Nulleinträge ungerade sind, allerdings sind sie in dieser Arbeit nicht größer von Bedeutung

kein Tangle zu einer Liste, durch welches diese realisiert wird, so nennen wir sie *nicht realisierbar*. Ein Beispiel für eine realisierbare und eine nicht realisierbare Liste ist in Abbildung 1.2 zu sehen. Eine Liste L kann auch als Multimenge betrachtet werden, in der Swap (i, j) genau l_{ij} mal vorkommt. Ein Tangle heißt *einfach*, wenn alle Permutationen verschieden sind. Die Höhe $h(L)$ einer realisierbaren Liste L ist die minimale Höhe eines Tangles, welches L realisiert. Ein Tangle T heißt *optimal*, falls $h(T) = h(L(T))$ gilt und wir sagen in diesem Fall, dass es Höhe OPT besitzt.

$$L_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$



$$L_2 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Abb. 1.2: Die Liste L_1 ist realisierbar; Da die Liste L_2 trennbar ist, ist sie nicht realisierbar.

Die Liste L_2 in Abbildung 1.2 verletzt eine wichtige Voraussetzung, die für Realisierbarkeit notwendig ist; die sogenannte *Untrennbarkeit*. Wir sagen, eine Liste ist *untrennbar* oder auch *nicht trennbar*, wenn es für alle Tripel i, j, k mit $i < j < k$ und einem (i, k)-Swap auch einen (i, j)- oder (j, k)-Swap gibt. Weiter nennen wir eine Liste *konsistent*, wenn die finale Konfiguration der Kabel eine Permutation von $[n]$ bilden. Für jedes Kabel i ist dessen finale Position die ursprüngliche Position weniger eins für jedes Kabel $j < i$ mit dem i Swappt und sowie plus eins für jedes Kabel rechts von i, mit dem i Swappt. Firman et al. [FKR$^+$19] haben gezeigt, dass Konsistenz und Untrennbarkeit hinreichend für die Realisierbarkeit einer einfachen Liste sind.

1.2 Eigener Beitrag

Wir behandeln zunächst Eigenschaften von allgemeinen Listen und finden eine natürliche Einschränkung der betrachteten Listen, die sogenannten *minimal unabhängigen Listen*, siehe Kapitel 2. Diese machen wir uns zunächst in Kapitel 3 zu Nutze, indem wir zeigen, dass es ausreichend ist, den schon erwähnten Algorithmus Odd-Even-Sort zum Erzeugen eines Tangles eingeschränkt auf minimal unabhängige Listen zu betrachten. Weiter zeigen wir in demselben Kapitel, dass es Listen gibt, beidenen keine der beiden Varianten von Odd-Even-Sort ein optimales Tangle ausgibt. Daraufhin stellen wir in Abschnitt 3.1 zwei Greedy-Algorithmen vor, namentlich Greedy-Tangler und Modified-Odd-Even-Sort, welche wir ebenfalls implementiert und getestet haben. In Abschnitt 3.2, vergleichen wir unsere beiden Algorithmen mit Odd-Even-Sort und zeigen, dass zumindest Modified-Odd-Even-Sort bei gleicher asympotitischer Laufzeit von $\mathcal{O}(n^2)$ in mehr Fällen ein Tangle von optimaler Höhe generiert, wobei n die Anzahl der Kabel ist.

2 Allgemeine Eigenschaften von Listen

Wir möchten zuerst herausfinden, ob es neben den bisher bekannten Eigenschaften von Listen noch weitere gibt. Wir können Listen zunächst in zwei Mengen einteilen: Solche, bei denen alle Kabel mindestens einmal Swappen und solche, bei denen mindestens ein Kabel existiert, welches an keinem Swap beteiligt ist. Offensichtlich ist der Schnitt zwischen diesen Mengen leer.

Falls wir eine Liste L haben zu der ein Kabel w_j ohne Swap gehört, also für welches $\sum_{i=1}^{n} l_{ij} = 0$ gilt, so nennen wir die Listen, welche aus den Swaps der Kabeln links von w_j, also jenen Kabel w_i mit $i < j$, beziehungsweise rechts von w_j besteht, die *Teillisten* von L und schreiben bei Eindeutigkeit L_ℓ bzw. L_r. Allerdings muss es nicht exakt solche Listen geben. Es kann zum Beispiel auch das erste Kabel an keinen Swaps beteiligt sein, dann hat die Liste allerdings nur eine rechte Teilliste. Wir möchten nun den Leser mithilfe eines Beispiels ein wenig mit dem Begriff der Teilliste vertraut machen.

Beispiel. Sei

$$L = \begin{pmatrix} 0 & 2 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.1}$$

Dann sind

$$L_\ell = \begin{pmatrix} 0 & 2 \\ 2 & 0 \end{pmatrix} \qquad L_\mathrm{r} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.2}$$

Dieses Beispiel zeigt auch eine Eigenschaft die wir sofort aufzeigen werden, welche auch intuitiv Sinn macht. Ist L untrennbar und existieren zwei Teillisten L_ℓ und L_r von L die realisierbar sind, so ist auch L realisierbar.

Lemma 2.1. *Sei L eine untrennbare Liste mit n Kabeln, w_j ein Kabel in L für $j \in [n]$ welches mit keinem anderen Kabel tauscht und L_ℓ,L_r seien die Teillisten deren Kabel links beziehungsweise rechts von w_j liegen, es sei also $i < j < k$ für alle Kabel $w_i \in L_\ell$, $w_k \in L_\mathrm{r}$.*

Dann ist L genau dann realisierbar, wenn die Teillisten L_ℓ und L_r realisierbar sind.

Beweis. Da L untrennbar ist, existieren keine Kabel $w_i \in L_\ell$, $w_\mathrm{k} \in L_\mathrm{r}$ die tauschen, da diese auch mit w_j vertauscht werden müssten(*).

Sei zunächst L durch ein Tangle T realisiert. Dann tauscht in T das Kabel w_j mit keinem anderen Kabel, wir können also zwei Tangles so definieren, dass sie nur die Kabel aus L_ℓ respektive L_r enthalten beziehungsweise die jeweiligen Listen visualisieren. Somit haben wir für je L_ℓ und L_r ein Tangle gefunden welches die Listen realisiert.

Sind nun L_ℓ,L_r realisierbar, so existieren nach allen durchzuführenden Swaps aus den Teillisten keine weiteren aufgrund von (*), womit L bereits realisiert ist, wir können also L realisieren indem wir die Tangles von L_ℓ und L_r so zusammenfügen, dass das Kabel w_j zwischen ihnen liegt. Eventuell muss noch eines der Tangles so lange durch identische Permutationen erweitert werden, bis die Höhen gleich sind. □

Handelt es sich bei der Liste L um eine gerade Liste, also eine Liste mit $l_{ij} \bmod 2 \equiv 0$ für alle $l_{ij} \in L$, so stellt sich die Frage ob man Kabel und dazugehörige Swaps bei der Frage nach der Realisierbarkeit außen vor lassen kann. Offensichtlich gilt dies für solche Kabel, die nicht geswappt werden, solange hierdurch die Untrennbarkeit beeinflusst wird. Aber auch für solche die geswappt werden ist dies unter bestimmten Umständen möglich.

Lemma 2.2. *Ist L eine untrennbare, gerade Liste und tauscht das Kabel w_i mit allen Kabeln links von sich, d.h. für die Liste L ist $l_{ij} > 0, \forall 1 \leq j < i$, und keinen weiteren, so lässt sich w_i an die Position am weitesten links tauschen ohne die Realisierbarkeit von L zu beeinflussen. Äquivalentes gilt auch im Fall, dass w_i mit allen Kabeln rechts von sich und sonst keinen tauscht.*

Beweis. Da L gerade ist, ist die schlussendliche Permutation gleich der ursprünglichen, was wir uns zunutze machen.

Zunächst lässt sich w_i nach Voraussetzung durch $(w_{i-1}, w_i), (w_{i-2}, w_i), \ldots, (w_0, w_i)$ auf die Position 1 tauschen. Sei L' die Liste, die aus L hervorgeht, wenn man das Kabel w_i aus L streicht, also

$$L' = \begin{pmatrix} \ell_{11} & \ldots & \ell_{1,i-1} & \ell_{1,i+1} & \ldots & \ell_{1n} \\ \vdots & \ddots & \vdots & \vdots & \iddots & \vdots \\ \ell_{i-1,1} & \ldots & \ell_{i-1,i-1} & \ell_{i-1,i+1} & \ldots & \ell_{i-1,n} \\ \ell_{i+1,1} & \ldots & \ell_{i+1,i-1} & \ell_{i+1,i+1} & \ldots & \ell_{i+1,n} \\ \vdots & \iddots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ldots & \ell_{n,i-1} & \ell_{n,i+1} & \ldots & \ell_{nn} \end{pmatrix} \tag{2.3}$$

Ist L' nun realisierbar, so ist auch L realisierbar, da nach Durchführung von L' nur die Tauschoperationen für w_i verbleiben. Da L' auch gerade ist nach Voraussetzung, ist die Permutation von L' vor dem Tauschen gleich derjenigen nach dem Tauschen. Da jedes Kabel w_j, $j < i$ mindestens einmal, aber nie gerade oft mit w_i tauscht, lässt sich nun die ursprüngliche Permutation von L wiederherstellen.

Den Fall, dass w_i mit allen Kabeln w_j, $j > i$ und sonst keinen tauscht, zeigt man äquivalent. □

Der Begriff der Teilliste ist abhängig von einem Kabel, welches keinen Swap durchführt. Offensichtlich gibt es Listen L mit Dimension n welche man aufspalten kann in Listen L_1 und L_2 der Dimensionen m und $m-n$, sodass die Konsistenz, Untrennbarkeit sowie Realisierbarkeit der Liste L auf die Listen L_1 sowie L_2 übertragen werden. Daher möchten wir den Begriff der Teilliste verallgemeinern.

Definition 2.3. *Sei L eine konsistente und untrennbare Liste. Dann nennen wir eine Liste U* unabhängige Teilliste *(von L) wenn*

- *U konsistent und untrennbar ist*
- *die Liste $L\backslash U := \begin{pmatrix} * & 0 & * \\ 0 & U & 0 \\ * & 0 & * \end{pmatrix} \backslash U := \begin{pmatrix} * & 0 & * \\ 0 & 0 & 0 \\ * & 0 & * \end{pmatrix}$, mit 0 als Nullmatrix beziehungsweise der Liste ohne Swaps, konsistent und untrennbar ist. Hierbei darf U auch die Kabel 1 und n enthalten, es darf also $L = \begin{pmatrix} U & 0 \\ 0 & * \end{pmatrix}$ sowie $L = \begin{pmatrix} * & 0 \\ 0 & U \end{pmatrix}$ gelten.*

Eine unabhängige Teilliste ist also eine Liste U an Swaps, welche man aus der ursprünglichen Liste L entfernen kann, ohne die resultierende Liste inkonsistent und trennbar zu machen. Es ist zu beachten, dass die Existenz von unabhängigen Teillisten keinesfalls die Realisierbarkeit impliziert. Es gelten die folgenden Eigenschaften.

- jede konsistente und untrennbare Liste L ist eine unabhängige Teilliste ihrer selbst, da L bereits konsistent und untrennbar ist und $L\backslash U = 0$ konsistent und untrennbar ist.
- L ist von der Form $\begin{pmatrix} U_1 & 0 & 0 \\ 0 & U_2 & 0 \\ 0 & 0 & U_3 \end{pmatrix}$, da sonst Kabel aus U_1 mit Kabeln aus U_3 Swappen müssten, jedoch ohne mit jenen aus U_2 zu tauschen, was offensichtlich nicht möglich ist, da keine Swaps (i_1, i_2) bzw (i_2, i_3), $i_1 \in U_1$, $i_2 \in U_2$, $i_3 \in U_3$ existieren.
- Enthält L mindestens eine unabhängige Teilliste die ungleich L ist, so besteht die Liste aus mindestens zwei unabhängigen Teillisten U_1 sowie U_2, wobei oBdA nach eventueller Umbenennung $L = \begin{pmatrix} U_1 & 0 \\ 0 & U_2 \end{pmatrix}$ gilt, da $L\backslash U_2 = \begin{pmatrix} U_1 & 0 \\ 0 & 0 \end{pmatrix}$ konsistent und untrennbar ist.
- L ist realisierbar genau dann, wenn auch $L\backslash U$ und U realisierbar sind, da $L\backslash U$ und U offensichtlich realisierbar sind, wenn L realisierbar ist, da dann jede Teilliste realisierbar ist und falls alle Teillisten von L realisierbar sind, so ist durch ihr gemeinsames Tangle auch L bereits realisiert.

Insbesondere die letzte Eigenschaft ermöglicht es uns die Frage nach Realisierbarkeit einzuschränken auf die Realisierbarkeit der unabhängigen Teillisten. Um möglichst kleine Listen zu erhalten, benötigen wir allerdings noch zwei weitere Definitionen.

Definition 2.4. *Wir nennen eine Liste* zusammenhängend*, wenn der Graph bestehend aus den Kabeln als Knoten und Swaps als Kanten zusammenhängend ist.*

Definition 2.5. *Sei U eine unabhängige Teilliste einer Liste L, die zudem noch zusammenhängend ist. Dann nennen wir U* minimal unabhängige Teilliste *von L. Betrachten wir U ohne L oder ist $L{=}U$, so nennen wir U* minimal unabhängige Liste*.*

Minimal unabhängige Listen, zu denen wir auch kurz *m.u. Listen* sagen, bieten uns eine hervorragende Grundlage zur Betrachtung des Problems der Höhenminimierung, da rein intuitiv betrachtet die Höhe eines Tangles zu einer Liste, welche aus minimal unabhängigen Teillisten $U_1, U_2, \ldots, U_k$ besteht die Höhe des höchsten Tangles der U_i sein muss. Diese Intuition wird sich später zumindest für einen Spezialfall als korrekt herausstellen. Zudem ist es eine natürliche Einschränkung, da bei jeder Liste, welche aus minimal unabhängigen Teillisten besteht, die einzelnen m.u. Teillisten einzeln betrachtet werden können und ihre Eigenschaften auf die ursprüngliche Liste übertragen werden können, was insbesondere bei Algorithmen zu verbesserten Laufzeiten führen kann.

3 Odd-Even-Sort

Wir möchten uns nun mit der Minimierung der Höhe von Tangles befassen. Sowohl Wang [Wan91] als auch Firman et al. [FKR$^+$19] befassten sich damit bereits. Letztere lieferten einen exakten Algorithmus zur Realisierung von Listen, welcher ebenfalls die Höhe eines Tangles minimiert.

Wang zeigte insbesondere, dass eine parallele Variante von Bubblesort bereits ein Tangle der Höhe OPT oder OPT+1 generiert, wobei OPT hierbei die optimale Höhe bezeichnet. Diese Variante wird Odd-Even-Sort genannt, läuft in $\mathcal{O}(n^2)$, wie Sado und Igarashi [SI87] zeigten, und geht wie folgt vor.

Zunächst werden jeweils alle Positionen mit ungeradem Index betrachtet und falls sie mit dem Kabel rechts von dieser Position tauschen, so wird dieser Swap durchgeführt und der Swap aus der Liste gestrichen. Hiernach werden diejenigen Positionen betrachtet, die geraden Index besitzen und die gleiche Operation durchgeführt. Dieser Vorgang wird nun solange durchgeführt, bis die Liste abgearbeitet ist, ein Beispiel für ein so generiertes Tangle ist in Abbildung 3.1 zu sehen.

Eine äquivalente Variante bei der zunächst die geraden und dann die ungeraden Indizes betrachtet werden, bezeichnen wir mit Even-Odd-Sort oder einfach nur Even-First, falls der Kontext klar ist. Entsprechend diesem Schema bezeichnen wir die erste Variante als Odd-Even-Sort oder Odd-First.

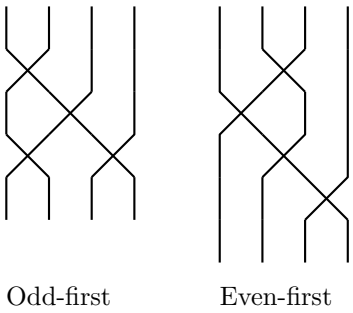


Abb. 3.1: Beispiel für zwei von Odd-Even-Sort generierte Tangles

In Abbildung 3.1 sehen wir bereits ein Beispiel bei dem Odd-Even-Sort und Even-Odd-Sort Tangles von verschiedener Höhe erstellen.

Algorithmus 1 : Odd-Even-Sort(List L)

Eingabe : konsistente und untrennbare Liste L
Ausgabe : Tangle T zu L

```
1  π = π_id
2  k = 0
3  T = ⟨π⟩
4  while |L| > 0 do
5      k = k + 1
6      π_k = ∅
7      for i = 1 to dim(L) do
8          j = π^{-1}(i)
9          l = π^{-1}(i + 1)
10         if l_{j,l} == 1 then
11             π_k = π_k ∘ {(l, j)}        // Wir fügen zu π_k = π_k(π^{-1}(1, ..., i − 1))
12             l_{j,l} = l_{l,j} = 0       //   die Permutation π^{-1}(i + 1, i) hinzu
13         else
14             π_k = π_k ∘ π_id(j, l)
15         i = i + 2
16     k = k + 1
17     π_k = π
18     for i = 2 to dim(L) do
19         j = π^{-1}(i)
20         l = π^{-1}(i + 1)
21         if l_{j,l} == 1 then
22             π_k = π_k ∘ {(l, j)}
23             l_{j,l} = l_{l,j} = 0
24         else
25             π_k = π_k ∘ π_id(j, l)
26         i = i + 2
27     if π ≠ π_k then
28         T = T.append(π_{k−1})           // erweitere das Tangle um
29         π = π_{k−1}                     // die aktuelle Permutation
30         if π ≠ π_k then
31             if π ≠ π_{k−1} then
32                 T = T.append(π_{k−1})
33                 π = π_{k−1}
34     else
35         if π ≠ π_k then
36             T = T.append(π_k)
37             π = π_k
38 return T
```

Wir möchten im folgenden betrachten wie sich Tangles der Höhe OPT+1 zwischen solchen von Odd-Even-Sort und Even-Odd-Sort verteilen, wobei wir die betrachteten Listen auf minimal unabhängige beschränken. Um dies durchzuführen erstellen wir zu jeder Permutation von $[n]$ eine Liste L, für welche wir zunächst überprüfen ob sie minimal unabhängig ist. Sollte das der Fall sein, wenden wir Odd-Even-Sort bzw Even-Odd-Sort auf L an. Weiter vergleichen wir das erhaltene Tangle mit dem durch den Algorithmus von Firman et al.[FKR$^+$19] entstandene, vergleiche Algorithmus 2[1]. Hierdurch erhalten wir alle Listen bzw Tangles deren Höhe größer als OPT ist. Aufgrund der Laufzeit von $\mathcal{O}\left((2|L|/n^2+1)^{n^2/2} \cdot n^3 \cdot \log|L|\right)$ des von Firman et al. entworfenen Algorithmus sowie dem Umstand, das jede Permutation von $[n]$ getestet wird, beläuft sich der gesamte Zeitaufwand für den Algorithmus auf $\mathcal{O}\left(n! \cdot (2|L|/n^2+1)^{n^2/2} \cdot n^5 \cdot \log|L|\right)$. Wir erhalten so das folgende Ergebnis:

		Anzahl der Tangles mit Höhe OPT+1	
n	Anzahl m.u. Listen	Odd-First	Even-First
2	1	0	1
3	3	1	1
4	13	2	8
5	71	25	25
6	461	92	254
7	3447	1253	1253
8	29094	6779	15400

Tab. 3.1: Übersicht wie häufig Odd-First bzw. Even-First Tangles der Höhe OPT+1 ausgeben

Wir sehen, dass zu den getesteten ungeraden Dimension die Anzahl der Tangles mit Höhe OPT+1 für sowohl Odd- als auch Even-First gleich ist. Folglich ist es nur natürlich, die Frage zu stellen ob, und falls ja, wieso dies der Fall für alle ungeraden n ist. Wir sehen auch, dass für die untersuchten Listendimensionen Odd-First seltener Tangles der Höhe OPT+1 erstellt als Even-First. Dies führt uns auf zwei Vermutungen:

Vermutung 1. Odd-first gibt für alle Dimensionen $n \in \mathbb{N}$ höchstens so viele Tangle der Höhe OPT+1 aus wie Even-First.

Vermutung 2. Für ungerade Anzahl an Kabeln geben odd- und Even-First genau gleich viele Tangle mit Höhe OPT+1 aus.

Vergleicht man nun die Ausgabe von Odd-Even-Sort und Even-Odd-Sort bei gleicher Liste, so könnte man zunächst vermuten, dass mindestens eines der ausgegebenen Tangles jeweils gleich ist. Diese These wird dadurch unterstützt, dass sie auf alle Listen mit Dimension höchstens 5 zutrifft.

[1] Wir verzichten in der hier erklärten Variante im Vergleich zum aufgeschriebenen Pseudocode des Algorithmus auf das Erstellen eines zweiten Tangles sowie den Höhenvergleich zwischen den beiden durch den zu testenden Algorithmus erstellten

Um sie weiter zu überprüfen, verwenden wir wieder Algorithmus 2. Dieser unterscheidet sich im Vergleich zum vorigen Algorithmus in dem Punkt, dass zunächst die Höhe der beiden Tangles zur gleichen Liste verglichen wird und nur, falls diese gleich ist der weitere Vergleich mit der optimalen Höhe stattfindet. Da wir somit nur eine weitere Iteration von Odd-Even-Sort (bzw. Even-Odd-Sort) erhalten, verändert sich die asymptotische Laufzeit nicht.

Algorithmus 2 : testingAlgorithm(natürliche Zahl n, zu verwendender Algorithmus Algo)

Eingabe : natürliche Zahl n, Algorithmus Algo
Ausgabe : Alle Listen zu denen beide Varianten des Algorithmus Tangles von nicht-optimaler Höhe ausgeben

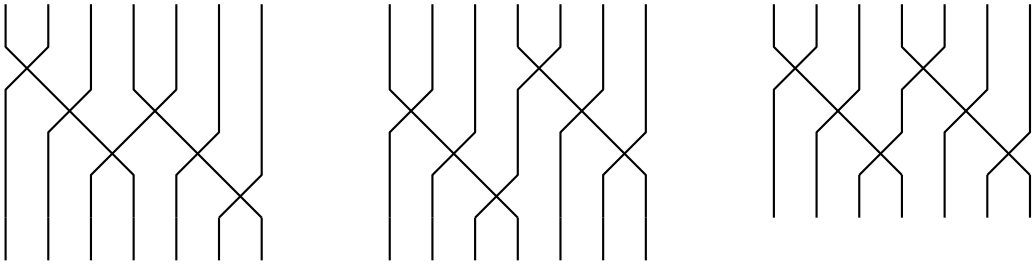
```
1  A = {1, ..., n}
2  B = A
3  TangleSet
4  permute(A)
5  while A ≠ B do
6      B = A
7      L =List.Assemble(A)
8      if L.isMinimallyIndependent then
9          T1 = Algo.firstVariant(L)
10         T2 = Algo.secondVariant(L)
           // Sind die Höhen der Tangles gleich, so muss entweder
           // Höhe=OPT oder Höhe>OPT gelten
11         if T1.height == T2.height then
12             if T1.height > Zink.computeMinHeightRealization.height then
13                 TangleSet.add(L)
14     permute(A)
15 return T
```

Hierbei ist permute(A) jene Methode, die uns die nächste Permutation von A in lexikografischer Ordnung übergibt und firstVariant() sowie secondVariant() entsprechen der jeweils ersten beziehungsweise zweiten vorgestellten Variante des jeweils zu überprüfenden Algorithmus, bei Odd-Even-Sort dementsprechend Odd-First bzw. Even-First. Zink.computeMinHeightRealization ist hierbei jene Methode, die den Algorithmus zur Minimierung der Höhe von Firman et al. implementiert[2].

Als eines der acht Beispiele zur Dimension $n = 7$ bei denen sowohl Odd-First als auch Even-First ein Tangle der Höhe OPT+1 ausgeben, erhalten wir dasjenige in Abbildung 3.2. Wir sehen hier sehr gut, dass Odd-Even-Sort in beide Varianten daran scheitert ein Tangle der Höhe OPT+1 auszugeben. Die optimale Höhe ist offensichtlich

[2]Wir möchten an dieser Stelle Philipp Kindermann und Johannes Zink [KZ22] danken die uns ihren Java-Code zur Verfügung gestellt haben

3, was gleich der minimal notwendigen Höhe gleicht, da die größte Anzahl Swaps die ein Kabel durchführt drei ist und dementsprechend viele Schichten mindestens benötigt werden. Da jedoch die beiden Kabel 1 und 4 von verschiedener Parität sind, aber beide an jeweils drei Swaps beteiligt sind, jedoch kein (1,4)-Swap existiert können Odd-First und Even-First nur Tangles der Höhe mindestens vier erzeugen. Für $n = 8$ existieren 48 Beispiele bei denen Odd-First und Even-First ein Tangle der Höhe OPT+1 ausgeben.



Odd-First Even-First optimal

Abb. 3.2: Gegenbeispiel zur These, dass die minimale Höhe der Ausgaben von Odd-First und Even-First immer optimal ist.

		Anzahl der Tangles mit Höhe OPT+1 bei		
n	#m.u. Listen	Odd-First	Even-First	odd-&Even-First
2	1	0	1	0
3	3	1	1	0
4	13	2	8	0
5	71	25	25	0
6	461	92	254	0
7	3447	1253	1253	8
8	29094	6779	15400	48

Tab. 3.2: Verteilung der Tangles mit Höhe OPT+1 bei Odd-Even-Sort.

Um die auffallende Gleichmäßigkeit der Verteilung von Tangles mit Höhe OPT+1 unter den getesteten Listen mit ungerader Dimension zu untersuchen, wollen wir zunächst einen neuen Begriff einführen, der es uns insbesondere ermöglichen wird einen Teil von Vermutung 2 zu beweisen.

Definition 3.1. *Sei L eine beliebige Liste. Die Liste*

$$M := (m_{i,j}) := (\ell_{n-j+1,n-i+1})$$

heißt gespiegelte *Liste von L.*

M ist offensichtlich genau jene Liste die man erhält, wenn man L an der Gegendiagonale spiegelt. Offensichtlich existiert zu jeder Liste L eine gespiegelte Liste und zudem

lassen sich einige Eigenschaften von L auf M übertragen, was wir später nutzen werden.

Satz 3.2. *Sei L eine Liste mit gespiegelter Liste M. Dann werden folgende Eigenschaften auf M übertragen:*

- *Konsistenz*
- *Untrennbarkeit*
- *Realisierbarkeit*
- *Minimale Unabhängigkeit*

Beweis. Seien L und M wie vorgegeben.

Zur Konsistenz:
Da M die an der Gegendiagonalen gespiegelte Liste L ist, ist

$$\sum_{i<j} \ell'_{i,j} - \sum_{i>j} \ell'_{i,j} = \sum_{i>j} m'_{i,j} - \sum_{i<j} m'_{i,j}$$

wobei $\ell' \equiv \ell \mod 2$ sowie $m' \equiv m \mod 2$ ist, d.h. für jedes Kabel i zu dem die neue Position $n-i+1$ ist, wird um jeweils den gleichen Betrag verschoben, ist also $\pi_L(i) := i + \sum_{i<j} l'_{i,j} - \sum_{i>j} l'_{i,j}$, so ist $\pi_M(n-i+1) = n-i+1+\sum_{i>j} m'_{i,j} - \sum_{i<j} m'_{i,j}$ und es ist $\pi_M(n-i+1) \neq \pi_M(n-j+1) \forall i \neq j$, wodurch die Konsistenz gewährt ist.

Zur Untrennbarkeit:
Da in L zu jedem Swap (i,j) entweder ein (i,k)-Swap oder ein (k,j)-Swap existiert oder i und j bereits benachbart sind, existiert nach Definition zu jedem $(n-j+1, n-i+1)$-Swap ein $(n-j+1, n-k+1)$- oder ein $(n-k+1, n-i+1)$-Swap existiert beziehungsweise $n-j+1$ und $n-i+1$ benachbart sind.

Realisierbarkeit:
Da L realisierbar ist, existiert ein realisierendes Tangle T. Sei T' das Tangle, welches entsteht wenn man T an der Achse, die entlang der Position $\frac{n-1}{2}$ vertikal gelegen ist, spiegelt. Dann realisiert T' die Liste M, da wir für jeden (i,j)-Swap in T einen $(n-j+1, n-i+1)$-Swap in T' machen.

Zur minimalen Unabhängigkeit:
Die sieht man sofort ein, wenn man die Definition der minimalen Unabhängigkeit mit der Definition der gespiegelten Liste kombiniert. □

Satz 3.3. *Sei L eine einfache Liste von ungerader Dimension und gespiegelter Liste M. Dann ist Tangle, welches von Odd-Even-Sort(L) ausgegeben wird gleich dem an der Position $\frac{n-1}{2}$ gespiegelten Tangle von Even-Odd-Sort(M). Äquivalentes gilt für die Tangle von Even-Odd-Sort(L) und Odd-Even-Sort(M).*

Beweis. Wir können die Vorgehensweise von Even-Odd-Sort umschreiben, so dass es wie Odd-Even-Sort funktioniert, jedoch den Swap $(\pi^{-1}(i-1), \pi^{-1}(i))$ betrachtet anstelle des Swaps $(\pi^{-1}(i), \pi^{-1}(i+1))$ (solange $i \neq 1$ gilt) und nennen diese Version Odd-Even-Sort-BW. Dann ist das Tangle, welches Odd-Even-Sort-BW(M) erzeugt genau dasjenige, welches Odd-Even-Sort(L) erzeugt und an der Achse, die durch die Position $\frac{n-1}{2}$ geht, gespiegelt wird, denn die Parität von i ist gleich der von $n-i+1$, da $2 \mid n+1$ und somit $2 \mid n-i+1 \Leftrightarrow 2 \mid i$ und somit werden nach Konstruktion von Odd-Even-Sort-BW die Swaps $(i, i+1)$ in der selben Iteration von Odd-Even-Sort ausgeführt wie die Swaps $(n-i, n-i+1)$, wodurch die gespiegelte Tangles erstellt werden. □

Zu jeder Liste L existiert eine zugehörige gespiegelte Liste M, insbesondere ist also die zu M gehörende gespiegelte Liste L. Daher gibt es eine gerade Anzahl an gespiegelten Listen mit L≠M. Die Anzahl der Listen mit $L = M$ und Dimension n ist

$$\left(\sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} n - 2k - 1\right)! = \begin{cases} \left(\frac{n^2}{4} - 1\right)! & 2 \mid n \\ \frac{n^2-1}{4}! & 2 \nmid n \end{cases} \tag{3.1}$$

da für jede Zeile der Liste L oberhalb der Zeile $\lfloor \frac{n}{2} \rfloor$ durch die Spiegelung an der Gegendiagonalen sowie der Symmetrie der Liste genau $n-2k-1$ Kabel existieren die unbestimmt sind. Dies ist in Abbildung 3.3 schematisch dargestellt. Man beachte insbesondere, dass $\frac{n^2-1}{4}!$ für ungerade n wohldefiniert ist, da für beliebiges n der Form $2m+1$ oder $2m+3$ die Gleichungen $n^2 - 1 = 4m^2 + 4m + 1 - 1 = 4m^2 + 4m = 4(m^2 + m)$ und $n^2 - 1 = 4m^2 + 12m + 9 - 1 = 4m^2 + 12m + 8 = 4(m^2 + 3m + 2)$ gelten. Alle übrigen Einträge abseits der l_{ii} sind durch die Wahl der $\ell_{ij}, i < j < n - i + 1$ bereits festgelegt. Über die Anzahl der minimal unabhängigen Listen für die sowohl Odd-Even-Sort(L_i)

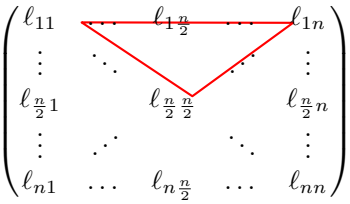


Abb. 3.3: Schema der veränderbaren Einträge der Liste L

als auch Even-Odd-Sort(L_i) ein Tangle der Höhe OPT+1 ausgeben, können wir nun folgende Aussage treffen:

Korollar 3.4. *Für eine ungerade Anzahl an Kabeln existiert eine gerade Anzahl an Listen $L_1, L_2, \ldots L_{2k}$, zu denen je eine gespiegelte Liste M_i existiert die ungleich L_i ist, für die sowohl Odd-Even-Sort(L_i) als auch Even-Odd-Sort(L_i) ein Tangle der Höhe OPT+1 zurückgeben.*

Wir wollen noch ein Ergebnis zeigen, welches die Höhen von Tangles zu minimal unabhängige Teillisten mit solchen von allgemeinen Listen verbindet. Insbesondere weißt uns

dieser Satz darauf hin, wie sich die Höhe einer allgemeinen konsistenten, untrennbaren und einfachen Liste zusammensetzt.

Satz 3.5. *Sei L eine konstistente, untrennbare und einfache Liste. Dann ist die Höhe des von Odd-Even-Sort ausgegebenen Tangles angewandt auf L ausschließlich abhängig von den Höhen aller ausgegebenen Tangles von Odd-Even-Sort mit gleicher relativer Parität angewandt auf die minimal unabhängigen Teillisten U_i.*

Es gilt sogar

$$Height(odd{-}even{-}Sort(L)) = \max_{i \in \mathbb{N}}(Height(odd{-}even{-}Sort(U_i)))$$

Gleiches gilt auch für Even-Odd-Sort.

Beweis. Es seien $U_1, U_2, \ldots, U_k$ minimal unabhängige Teillisten einer konsistenten, untrennbaren und einfachen Liste L wobei die Kabel ihre Nummerierung beibehalten, sodass $\max\{i \mid i \in U_k\} < \min\{i \mid i \in U_{k+1}\} \forall k = 1, \ldots, n-1$. Wir nehmen an, dass das Maximum der Höhen der von Odd-Even-Sort erstellten Tangles zu den U_i für $i \in [k]$ ungleich der Höhe des durch den Algorithmus erhaltenen Tangle von L sei.

Da Odd-Even-Sort bei gleicher relativer Parität jeweils das gleiche Tangle erstellt können wir die so erhaltenen Tangles der U_i jeweils nebeneinander zeichnen und erhalten so ein Tangle zu L. Allerdings erhalten wir, wenn wir Odd-Even-Sort auf L anwenden das gleiche Tangle, wobei wir offensichtlich die gleiche maximale Höhe erhalten müssen.

Für den Beweis zu Even-Odd-Sort gehen wir äquivalent vor und erhalten so das gleiche Ergebnis. □

3.1 Verbesserung von Odd-Even-Sort

Wir wollen nun versuchen Odd-Even-Sort so zu verändern, dass wir im Schnitt häufiger Tangles von der Höhe OPT erhalten. Diesen Versuch werden wir mithilfe zweier, an Odd-Even-Sort angelehnte, Algorithmen tätigen. Siehe Abschnitt 3.1.1 sowie Abschnitt 3.1.2.

3.1.1 Greedy-Tangler

Wir wollen zunächst den Algorithmus Greedy-Tangler betrachten und analysieren. Dieser funktioniert folgendermaßen:

Wir nehmen eine konsistente, untrennbare und einfache Liste entgegen und Swappen so lange Kabel entsprechend den Anweisungen der Liste, bis keine Swaps mehr durchzuführen sind. Beginnend mit Kabel 1 und Endend mit Kabel $n-1$ betrachten wir zum Kabel an der Position i immer ob ein Swap mit dem Kabel an $i+1$ in L existiert und führen diesen aus. Existiert kein Swap, so überprüfen wir als nächstes das Kabel an der Position $i+1$, im anderen Fall, also jenem in dem wir einen Swap durchgeführt haben, betrachten wir das Kabel an Position $i+2$. Sobald i mindestens gleich $n-1$ ist haben wir eine Schicht erstellt und können mit der nächsten Iteration fortfahren, vergleiche hierzu Abbildung 3.4 für ein Beispiel.

Diese unterscheidet sich gegenüber der bereits erklärten in dem Punkt, dass wir von der Position n beginnend bis $i = 2$ iterieren und jeweils betrachten ob der Swap $(\pi(i-1), \pi(i))$ existiert und diesen bei Existenz durchführen. Die Schrittweiten der Iterationen innerhalb der Schicht ist äquivalent zu den obigen. Diese Variante nennen wir Left-First, während wir die Variante, bei der wir mit der Iteration von n bis 2 beginnen mit Right-First bezeichnen.

Greedy-Tangler erzeugt immer ein Tangle da in jeder Iteration nur disjunkte Permutationen hinzugefügt die am Ende der vergangenen Iteration benachbart waren. Somit sind alle Permutationen adjazent. Da zudem so lange Swaps hinzugefügt werden, bis alle Swaps durchgeführt werden, da dies die Abbruchbedingung der Schleife ist, erhalten wir immer ein Tangle zur eingegebenen Liste.

Wir wollen nun betrachten wie schnell Greedy-Tangler ist. Offensichtlich überprüfen wir in jeder Iteration höchstens $n-1$ mal ob es einen Swap gibt, somit ist die asymptotische Laufzeit der Iteration $\mathcal{O}(n)$. Zudem brechen wir nach h Iterationen ab, d.h. die asymptotische Laufzeit beträgt $\mathcal{O}(hn)$.

Da h höchstens gleich $\max\{|L| \mid L \text{ einfach und realisierbar}\} = (n^2-n)/2$ ist, erhalten wir mit $(n^2-n)/2$ eine obere Schranke für die Höhe. Somit ist die Laufzeit höchstens $\mathcal{O}(n^3-n^2)$ $\in \mathcal{O}(n^3) \in \mathcal{O}(n^3)$.

Abb. 3.4: Darstellung der Funktionsweise von Greedy-Tangler. Die Kabel in Magenta werden zum ersten mal in dieser Iteration überprüft, diejenigen in Orange sind solche die bereits von links überprüft wurden

Algorithmus 3 : Greedy-Tangler(List L)

Eingabe : (realisierbare) Liste L
Ausgabe : Tangle T zu L

```
1  π = π_id
2  k=0
3  T = ⟨π⟩
4  while |L| > 0 do
5      k = k+1
6      π_k = π
7      for i = 1 to dimension(L) − 1 do
8          a = π^{-1}(i)
9          b = π^{-1}(i + 1)
10         if l_{a,b} = 1 then
11             Swap(a,b)
12             π_k = π_k ∘ {(b, a)}        // Wir fügen zu π_k = π_k(π^{-1}(1, ..., i − 1))
13             l_{a,b} = l_{b,a} = 0          // die Permutation π^{-1}(i + 1, i) hinzu
14             i = i+1
15         else
16             π_k = π_k ∘ π_id(a, b)
17         i = i+1
18         π = π_k
19     k = k+1
20     π_k = π_{k−1}
21     for i = dimension(L) downto 2 do
22         a = π^{-1}(i − 1)
23         b = π^{-1}(i)
24         if l_{a,b} = 1 then
25             Swap(a,b)
26             π_k = π_k ∘ {(b, a)}
27             l_{a,b} = l_{b,a} = 0
28             i = i-1
29         else
30             π_k = π_k ∘ π_id(a, b)
31         i = i-1
32         π = π_k
33     if π ≠ π_k then
34         T = Tangle.extend(π_{k−1})
35         T = Tangle.extend(π_k)
36     else
37         if π ≠ π_{k−1} then
38             T = Tangle.extend(π_{k−1})
39 \return T
```

Um nun die Effizienz des Algorithmus zu testen, verwenden wir wieder den Test-Algorithmus. Hier erhalten wir bereits für Listen mit Dimension $n = 6$ Beispiele bei denen beide Varianten des Algorithmus ein suboptimales Tangle erzeugen, siehe Abbildung 3.5. Ebenfalls finden wir in Abbildung 3.6 drei Beispiele für Tangles mit Höhe OPT+2.

Das gefundene erlaubt uns nun eine verbesserte Einschätzung der Laufzeit zu geben. Da die bisher höchste gefundene Höhe OPT+2 ist, können wir eine Laufzeit von $\mathcal{O}(n^2)$ annehmen.

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

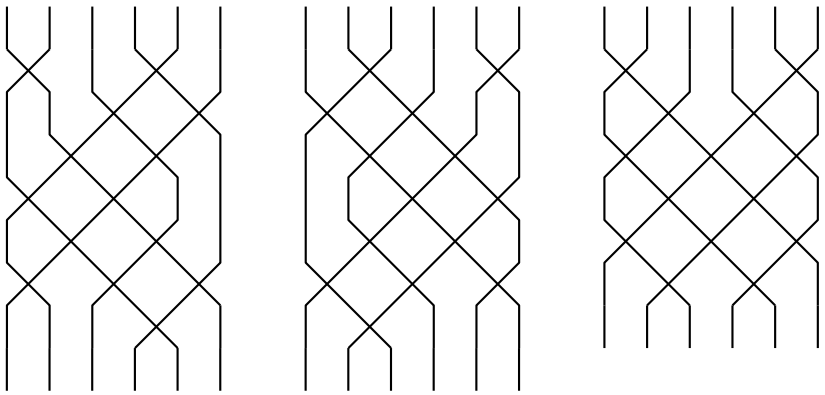


Abb. 3.5: Beispiel für ein Tangle mit Höhe OPT+1, generiert durch Greedy-Tangler

n	#m.u. Listen	Anzahl der Tangles mit Höhe größer OPT bei Odd-Even-Sort	Greedy-Tangler
2	1	0	0
3	3	0	0
4	13	0	0
5	71	0	0
6	461	0	17
7	3447	8	167
8	29094	48	2354

Tab. 3.3: Verteilung der Tangles mit Höhe größer OPT zwischen Odd-Even-Sort und Greedy-Tangler

Dieses Ergebnis impliziert, dass Greedy-Tangler keine Verbesserung von Odd-Even-Sort ist, sondern sogar eine Verschlechterung. Gerade die Beispiele in Abbildung 3.6 zeigen dies, da Odd-Even-Sort keine Tangles der Höhe OPT+2 ausgibt, Greedy-Tangler allerdings schon. Auch gibt Greedy-Tangler bereits bei geringerer Dimension Tangles mit Höhe größer OPT aus. Weiterhin generiert für die gleiche Dimension Greedy-Tangler häufiger Tangles der Höhe größer OPT als Odd-Even-Sort.

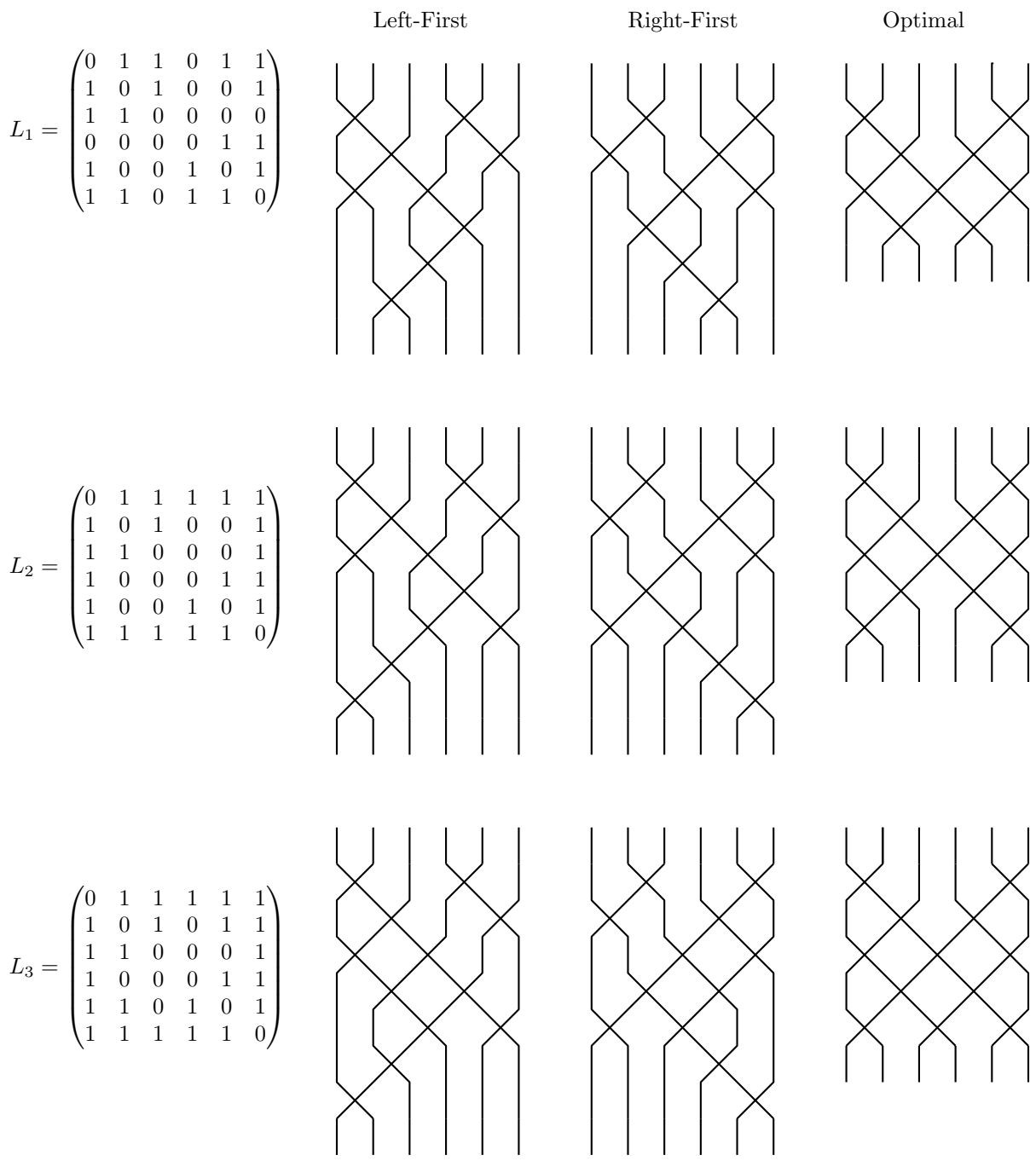


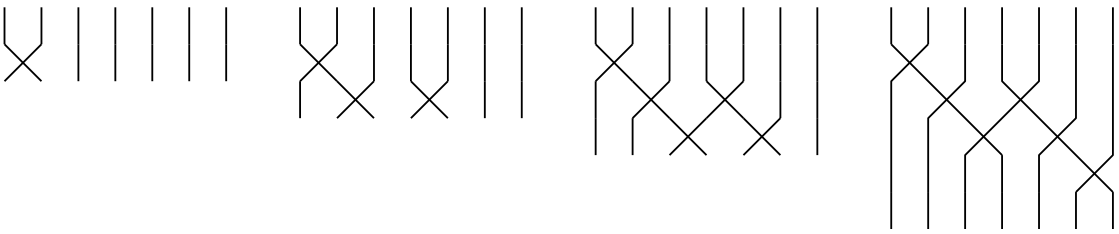
Abb. 3.6: Darstellung der drei von Greedy-Tangler generierten Tangles mit Höhe OPT+2

3.1.2 Modified-Odd-Even-Sort

Wir möchten einen weiteren Versuch unternehmen, Odd-Even-Sort zu verbessern. Hierfür ändern wir die Strategie: Anstatt Swaps durchzuführen an denen ein in der aktuellen Iteration bisher nicht betrachtetes Kabel beteiligt ist, möchten wir nur solche betrachten bei denen bisher nur einer der beiden potentiell möglichen Swaps berücksichtigt wurde.

Somit gehen wir zunächst wie bei Odd-Even-Sort vor und betrachten in der ersten Iteration zunächst Kabel an Positionen mit geradem Index. Falls wir jeweils mit dem Kabel rechts hiervon tauschen können, so machen wir dies. Andernfalls betrachten wir die Möglichkeit eines Swaps mit dem Kabel links des Kabels an der Position mit geradem Index und führen diesen gegebenenfalls durch, siehe Abbildung 3.7 für ein vergleichendes Beispiel zur Liste

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

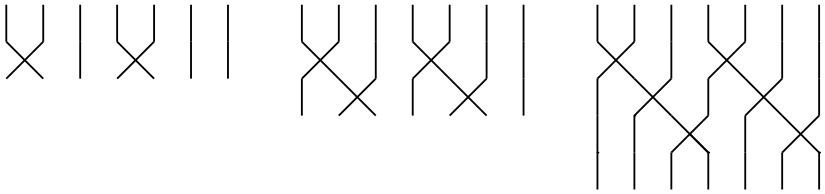


$\pi_1 = (2\,1\,3\,4\,5\,6\,7)$

$\pi_1 = (2\,1\,3\,4\,5\,6\,7)$
$\pi_2 = (2\,3\,1\,5\,4\,6\,7)$

$\pi_1 = (2\,1\,3\,4\,5\,6\,7)$
$\pi_2 = (2\,3\,1\,5\,4\,6\,7)$
$\pi_3 = (2\,3\,5\,1\,6\,4\,7)$

$\pi_1 = (2\,1\,3\,4\,5\,6\,7)$
$\pi_2 = (2\,3\,1\,5\,4\,6\,7)$
$\pi_3 = (2\,3\,5\,1\,6\,4\,7)$
$\pi_4 = (2\,3\,5\,1\,6\,7\,4)$



$\pi_1 = (2\,1\,3\,5\,4\,6\,7)$

$\pi_1 = (2\,1\,3\,5\,4\,6\,7)$
$\pi_2 = (2\,3\,1\,5\,6\,4\,7)$

$\pi_1 = (2\,1\,3\,5\,4\,6\,7)$
$\pi_2 = (2\,3\,1\,5\,6\,4\,7)$
$\pi_3 = (2\,3\,5\,1\,6\,7\,4)$

Abb. 3.7: Vergleichendes Beispiel bezüglich der Vorgehensweise von Modified-Odd-Even-Sort. Oben: Odd-First, Unten: Mod-Odd-First

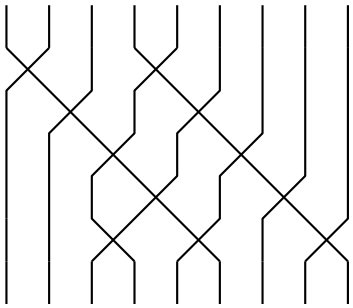
Für den exakten Algorithmus verweisen wir auf Algorithmus 4. Hier gibt es wieder zwei Varianten, genannt Mod-Odd-Even-Sort und Mod-Even-Odd-Sort, beziehungsweise Mod-Odd-First und Mod-Even-First, die sich wie bei Odd-First und Even-First in dem ersten betrachteten Kabel unterscheiden. Wie bei Greedy-Tangler fügt Modified-Odd-Even-Sort in jeder Iteration nur disjunkte Permutationen dem Tangle hinzu und bricht erst ab, sobald alle Swaps durchgeführt wurden, produziert also immer ein Tangle.

Da wir im schlechtesten Fall innerhalb einer Iteration alle möglichen Kombinationen an Swaps, von denen es $n-1$ gibt, durchprobiert werden, haben wir wieder eine asymptotische Laufzeit von $\mathcal{O}(hn)$. Es gilt nun, h zu bestimmen oder zumindest zu beschränken.

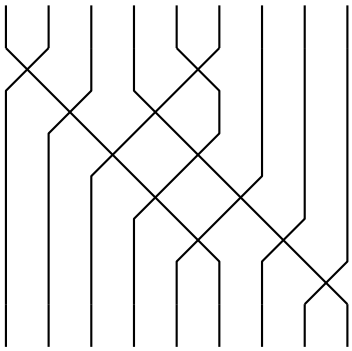
Hierfür können wir uns überlegen wie Modified-Odd-Even-Sort im Vergleich zu Odd-Even-Sort die Tangles erstellt. Modified-Odd-Even-Sort geht zunächst wie Odd-Even-Sort vor, Swappt aber auch jene Kabel zwischen denen es einen Swap gibt, die jedoch nicht in der aktuellen Iteration durch Odd-Even-Sort geswappt werden. Daher ist die Summe der bisher geswappten Kabel in jeder Iteration von Modified-Odd-Even-Sort immer mindestens gleich der Gesamtanzahl der bisher geswappten Kabel in der gleichen Iteration von Odd-Even-Sort. Insbesondere erhalten wir einen Swap in Modified-Odd-Even-Sort immer höchstens eine Iteration bevor er in Odd-Even-Sort durchgeführt werden würde. Hiermit erhalten wir als obere Schranke für die gesamte Laufzeit $\mathcal{O}(n^2)$.

Überprüfen wir diese Variante wieder mit Algorithmus 2, so sehen wir, dass bis Dimension $n = 8$ keine Liste existiert, für die Modified-Odd-Even-Sort Tangles der Höhe OPT+1 in beiden Varianten ausgibt. Allerdings gibt es für Dimension $n = 9$ insgesamt 54 Instanzen bei denen dies der Fall ist, eine dieser Listen ist inklusive der ausgegeben Tangles und eines optimalen Tangles in Abbildung 3.8 abgebildet.

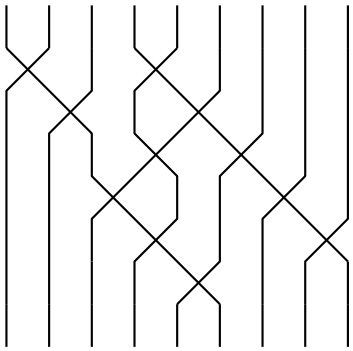
$$L = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 &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



optimal



Odd-First



Even-First

Abb. 3.8: Beispiel für eine Liste bei der Modified-Odd-Even-Sort sowohl bei Mod-Odd-First als auch Mod-Even-First ein Tangle der Höhe OPT+1 produzieren

Algorithmus 4 : Modified-Odd-Even-Sort(Liste L)

Eingabe : (realisierbare) Liste L

1 **Ausgabe :** Tangle T

$\pi = \pi_{\mathrm{id}}$

$k = 0$

$T = \langle \pi \rangle$

while $|L| > 0$ **do**

  $k = k + 1$

  $\pi_k = \pi$

  for $i = 1$ **to** $\dim(L)$ **do**

    $j = \pi^{-1}(i)$

    $m = \pi^{-1}(i + 1)$

    if $l_{j,m} == 1$ **then**

      $\pi_k = \pi_k \circ (m, j)$

      $l_{j,m} = l_{m,j} = 0$

    else

      if $i > 1$**and** $\pi_k^{-1}(i - 1) == \pi^{-1}(i - 1)$ **and** $l_{\pi^{-1}(i-1),j} == 1$ **then**

        $\pi_l = \pi_k \circ (j, \pi^{-1}(i - 1))$

        $l_{\pi^{-1}(i-1),j} = l_{j,\pi^{-1}(i-1)} = 0$

    $i = i + 2$

  $k = k + 1$

  $\pi_k = \pi_{k-1}$

  for $i = 2$ **to** $\dim(L)$ **do**

    $j = \pi^{-1}(i)$

    $m = \pi^{-1}(i + 1)$

    if $\pi_k^{-1}(i - 1) == \pi_{k-1}^{-1}(i - 1)$ **and** $l_{j,m} == 1$ **then**

      $\pi_k = \pi_k \circ (m, j)$

      $l_{j,m} = l_{m,j} = 0$

    else

      if $l_{\pi^{-1}(i-1),j} == 1$ **then**

        $\pi_l = \pi_k \circ (j, \pi^{-1}(i - 1))$

        $l_{\pi^{-1}(i-1),j} = l_{j,\pi^{-1}(i-1)} = 0$

    $i = i + 2$

  if $\pi_{k-1} \neq \pi_k$ **and** $\pi_{k-1} \neq \pi$ **then**

    $T = T.\mathrm{append}(\pi_{k-1})$

    $T = T.\mathrm{append}(\pi_{k-1})$

    $\pi = \pi_k$

  else

    if $\pi \neq \pi_{k-1}$ **then**

      $T = T.\mathrm{append}(\pi_{k-1})$

      $\pi = \pi_{k-1}$

return T

3.2 Vergleich der Algorithmen

Abschließend zu diesem Kapitel möchten wir noch einmal die Algorithmen miteinander vergleichen, insbesondere in den Punkten Laufzeit und Ausgabe.

Hierfür müssen wir wieder die einzelnen Laufzeiten betrachten. Sicher sind Odd-Even-Sort und Modified-Odd-Even-Sort in $\mathcal{O}(n^2)$, da jedoch Modified-Odd-Even-Sort im schlechtesten Fall jedes Paar benachbarter Kabel betrachtet, beträgt hier die Laufzeit $(n-1)(n+1) = n^2-1$, während Odd-Even-Sort in jeder Iteration höchstens $\lfloor (n-1)/2 \rfloor$ Vergleiche tätigt bei gleicher maximaler Höhe. Daher ist Odd-Even-Sort um einen Faktor 2 schneller als Modified-Odd-Even-Sort. Gleichzeitig ist die heuristische Laufzeit von Greedy-Tangler ebenfalls bei $\mathcal{O}(n^2)$ und ist genauso wie Modified-Odd-Even-Sort um einen Faktor zwei langsamer als Odd-Even-Sort, da auch hier bis zu $n-1$ Vergleiche in jeder Iteration gemacht werden, allerdings liegt dieser Algorithmus im schlechtesten Fall in $\mathcal{O}(n^3)$.

Wir wollen uns noch einmal der Optimalität der ausgegebenen Tangles zuwenden. Vergleichen wir also zunächst die Anzahl der Tangles die der jeweilige Algorithmus ausgibt und welche Höhe größer OPT für beide Varianten haben. Hierfür ziehen wir Tabelle 3.4 heran. Bis $n = 5$ gibt jeweils mindestens eine der Varianten ein Tangle der Höhe OPT aus, während ab $n = 6$ Greedy-Tangler bereits 17 Tangles mit suboptimaler Höhe erzeugt, einige hiervon sogar mit Höhe OPT+2, siehe Abbildung 3.6. Bei $n = 7$ erzeugt nun auch Odd-Even-Sort die ersten Tangles mit Höhe OPT+1, während Modified-Odd-Even-Sort dies erst zu $n = 9$ anfängt. Zudem liegt der Prozentsatz der von Greedy-Tangler erzeugten Tangles bei $n = 8$ bereits bei circa 8,1% und scheint auf die 10% zuzugehen. Demgegenüber stehen die Werte von Odd-Even-Sort von ca. 0,23% bei $n = 7$ und 0.16% bei $n = 8$ sowie 0,02% von Modified-Odd-Even-Sort bei $n = 9$.

		Anzahl der Tangles mit Höhe größer OPT bei		
n	#m.u. Listen	Odd-Even-Sort	Greedy-Tangler	Modified-Odd-Even-Sort
2	1	0	0	0
3	3	0	0	0
4	13	0	0	0
5	71	0	0	0
6	461	0	17	0
7	3447	8	167	0
8	29094	48	2354	0
9	273365	—	—	54

Tab. 3.4: Verteilung der Tangles mit Höhe größer OPT zwischen Odd-Even-Sort und Greedy-Tangler. Für $n = 9$ wurden aus Zeitgründen weder Odd-Even-Sort noch Greedy-Tangler getestet

Da Greedy-Tangler höchstens so gut ist wie jeder der beiden anderen Algorithmen ist es wenig sinnvoll ihn weiter zu untersuchen. Daher werden wir uns auf das Vergleichen von Odd-Even-Sort und Modified-Odd-Even-Sort beschränken.

Mithilfe von Tabelle 3.5 betrachten wir nun wie häufig die einzelnen Varianten von Odd-Even-Sort und bei $n = 2, \ldots, 8$ Kabel Tangles von suboptimaler Höhe generieren. Wir sehen, dass, zumindest für die untersuchten Listendimensionen, beide Varianten von Modified-Odd-Even-First jeweils öfters ein Tangle der Höhe OPT zurückgeben als die entsprechende Variante von Odd-Even-Sort. Demnach ist abzuschätzen, dass dies auch für höhere Dimensionen gilt.

Insbesondere liefert Mod-Odd-First bis einschließlich $n = 4$ kein Tangle der Höhe OPT+1, während die drei anderen Versionen schon bei geringeren Dimensionen solch suboptimale Tangles erzeugen. Was hier wieder auffällig ist, ist die Verteilung von Tangles mit Höhe OPT+1 bei ungeradem n, wie bereits bei bei Odd-Even-Sort beobachtet. Modified-Odd-Even-Sort erzeugt auch hier gleich häufig Tangles der Höhe OPT+1, allerdings sind es weniger als bei Odd-Even-Sort, nämlich ca. 12,7% im Vergleich zu ca. 35,2% bei $n = 5$ und ca. 22,4% gegenüber ca. 36,4% bei $n = 7$. Weiter ist bei geradem n das Wachstum der Anzahl der Listen mit OPT+1 bei Modified-Odd-Even-Sort größer als bei Odd-Even-First, nämlich ca 7-8% von n zu $n + 2$, bei 0% beginnend über 8% zu 15,2% bei Mod-Odd-First im Vergleich zu 3-5% von 15,4% sowie 20,0% zu 23,3% bei Odd-First und 14%, beziehungsweise 8% bei Mod-Even-First bei dem die Prozentwerte 15,4%, 29,7% und 38,7% sind, im Vergleich zu -7 bis -3% bei Even-First, da hier aus 62,5%, 55,6% beziehungsweise 52,9% der minimal unabhängigen Listen suboptimale Tangles erzeugt wurden[3].

		Anzahl der Tangles mit Höhe OPT+1 bei			
n	# m.u. Listen	Odd-First	Even-First	Mod-Odd-First	Mod-Even-First
2	1	0	1	0	0
3	3	1	1	0	0
4	13	2	8	0	2
5	71	25	25	9	9
6	461	92	254	37	137
7	3447	1253	1253	771	771
8	29094	6779	15400	4418	11270

Tab. 3.5: Verteilung der Tangles mit Höhe OPT+1 bei den Varianten von Odd-Even-Sort und Modified-Odd-Even-Sort

Die absolute Verteilung der Werte lässt uns ebenfalls abschätzen, dass Modified-Odd-Even-Sort im allgemeinen seltener Tangles der Höhe OPT+1 ausgibt und somit effizienter ist als Odd-Even-Sort

Ein Algorithmus welcher Swaps nach Kabelpriorität durchführt wurde Baumann [Bau20] formuliert und ebenfalls mit Odd-Even-Sort verglichen, jedoch ohne die Einschränkung auf minimal unabhängige Listen, wir verweisen bei Interesse daher auf diese Arbeit.

[3]Die jeweiligen Prozentwerte sind mit aufsteigendem n aufgeschrieben, d.h. immer für $n = 4$, $n = 6$, $n = 8$.

4 Fazit

Wir haben zunächst Teillisten eingeführt, welche uns ermöglicht haben ein Kriterium für die Realisierung von Listen mit Teillisten zu finden.

Hiernach haben wir die (minimal) unabhängigen Listen entwickelt, welche sich als eine sinnvolle Einschränkung der zu untersuchenden Listen herausgestellt hat, da sich somit die Frage nach den Eigenschaften Konsistenz, Untrennbarkeit sowie Realisierbarkeit darauf reduziert, ob die jeweiligen unabhängigen Listen die jeweilige Eigenschaften besitzen.

Anhand dieser Listen haben wir gezeigt, dass für Dimension $n \geq 3$ Odd-First in mindestens 15% der Fälle suboptimale Tangles erstellt. Ebenfalls konnten wir abschätzen, dass Odd-First in allen Fällen höchstens so viele Tangles der Höhe OPT+1 erzeugt wie Even-First, was wir bis Dimension $n = 8$ experimentell gezeigt haben. Zudem haben wir gezeigt, dass die Schnittmenge von Tangles mit Höhe OPT+1 von Odd-First und Even-First bis einschließlich Dimension $n = 6$ der leeren Menge gleicht, hiernach jedoch nicht mehr.

Im Anschluss hieran entwickelten wir die Algorithmen Greedy-Tangler sowie Modified-Odd-Even-Sort, wobei wir die asymptotische Laufzeit von Greedy-Tangler auf $\mathcal{O}(n^3)$ bestimmen konnten und sogar aufgrund der experimentellen Ergebnisse auf $\mathcal{O}(n^2)$ beschränkt haben. Wir konnten zudem zeigen, dass Greedy-Tangler im Gegensatz zu Odd-Even-Sort Tangles der Höhe OPT+2 ausgibt. Demgegenüber steht Modified-Odd-Even-Sort, dessen Laufzeit in $\mathcal{O}(n^2)$ liegt und welches bisher nur Tangles der Höhe höchstens OPT+1 erzeugt hat.

Schlussendlich haben wir die Algorithmen miteinander verglichen und zeigen können, dass Modified-Odd-Even-Sort tatsächlich eine Verbesserung von Odd-Even-Sort ist. Hierbei haben wir den prozentualen Anteil der suboptimalen Tangles verglichen und gesehen, dass bei ansteigender Dimension der Listen der Prozentsatz suboptimaler Tangles ebenfalls ansteigt.

5 Ausblick

An dieser Stelle möchten wir noch einige Gedanken und Ideen formulieren, die außerhalb des Rahmens der Arbeit liegen.

Um Vermutung 2 vollständig zu zeigen oder zu widerlegen fehlt ausschließlich die Parität der Anzahl der gespiegelten Listen mit $L = M$ für die sowohl Odd-First und Even-First Tangles der Höhe OPT+1 ausgeben. In diesem Zug ließe sich auch die Frage stellen, ob bei alle einfachen und minimal unabhängigen Listen L mit $L = M$ entweder Odd-First oder Even-First ein Tangle der Höhe OPT ausgeben. Insbesondere würde dies eventuell die Vermutung belegen können, falls dem so wäre.

Auch wenn wir Vermutung 1 als heuristisch korrekt annehmen können fehlt hierfür ein konkreter Beweis. Dieser würde wahrscheinlich darauf eingehen, dass in einem Even-First generierten Tangle die erste Permutation höchstens so viele Swaps beinhalten kann wie die erste Schicht in Odd-First, jedoch gerade bei Listen gerader Dimension dieses Maximum um 1 verringert wird.

Da sich jede Liste als eine $n \times n$-Matrix darstellen lässt wäre es möglich sich zu überlegen, ob die Eigenwerte der Matrizen etwas mit der Lösbarkeit oder der Ausgabe der untersuchten Algorithmen zu tun haben. Da wir symmetrische Matrizen haben ist die Frage nach bestimmten Eigenwerten zu stellen. Betrachtet man die Eigenwerte der Liste

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

welche die Liste zu Abbildung 3.2 ist, so erhält man $\{-2, -\sqrt{2}, 0, 0, 0, \sqrt{2}, 2\}$. Da hier der Eigenwert 0 vorkommt lässt sich die Frage stellen, ob dies ein solcher Wert ist, aufgrund dessen Odd-Even-Sort in beiden Varianten eine Tangle der Höhe OPT+1 ausgibt. Diese Vermutung wird dadurch verstärkt, dass unter anderem die Liste

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

die Eigenwerte $\{-1, -1, -1, -1, 4\}$ besitzt. Die äquivalent aufgebaute Liste mit Dimension $n = 7$, also die Liste mit $\ell_{ij} = 1$ für alle $i \neq j$ besitzt die paarweise verschiedenen

Eigenwerte -1 und 6, auffallenderweise ist -1 hier sechsmal vertreten als Eigenwert. Somit wäre eine weitere Eigenschaft die man zeigen kann, dass die Summe aller Eigenwerte der so aufgestellten Listen immer gleich null ist.

Zuletzt möchten wir noch eine Idee für einen weiteren Algorithmus vorstellen. Zu einer gegebenen Liste L wird in jeder Iteration jedem Kabel eine Priorität in Höhe der Anzahl der noch durchzuführenden Swaps zugewiesen und daraufhin werden Swaps so durchgeführt, dass in wir eine möglichst hohe Anzahl an Swaps haben und gleichzeitig werden die Swaps bevorzugt, an denen Kabel von möglichst hoher Priorität beteiligt sind.

Literaturverzeichnis

[Bau20] Jakob Baumann: Höhenminimierung einfacher Tangles. Bachelorarbeit, Fakultät für Informatik und Mathematik, Universität Passau, 2020.

[BW83] Joan S. Birman und R.F. Williams: Knotted periodic orbits in dynamical systems—I: Lorenz's equation. *Topology*, 22(1):47–82, 1983, https://doi.org/10.1016/0040-9383(83)90045-9.

[FKR+19] Oksana Firman, Philipp Kindermann, Alexander Ravsky, Alexander Wolff und Johannes Zink: Computing Optimal Tangles Faster. *CoRR*, abs/1901.06548, 2019. `http://arxiv.org/abs/1901.06548`.

[KZ22] Philipp Kindermann und Johannes Zink: Java and Python Code for Computing Tangles. Github repository, 2022. `https://github.com/PhKindermann/chaotic-attractors`.

[MHS+90] Gabriel B. Mindlin, Xin Jun Hou, Hernán G. Solari, R. Gilmore und N. B. Tufillaro: Classification of strange attractors by integers. *Phys. Rev. Lett.*, 64:2350–2353, 1990, 10.1103/PhysRevLett.64.2350. `https://link.aps.org/doi/10.1103/PhysRevLett.64.2350`.

[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): *Proc. 26th Int. Symp. Graph Drawing & Network Vis. (GD'18)*, Band 11282, Seiten 106–119, 2018, 10.1007/978-3-030-04414-5_8. `https://arxiv.org/abs/1807.11853`.

[SI87] Kazuhiro Sado und Yoshihide Igarashi: A Function for Evaluating the Computing Time of a Bubbling System. *Theor. Comput. Sci.*, 54:315–324, 1987, 10.1016/0304-3975(87)90136-8.

[Wan91] Deborah C. Wang: Novel routing schemes for IC layout part I: Two-layer channel routing. In: *Proc. 28th ACM/IEEE Design Automation Conf. (DAC'91)*, Seiten 49–53, 1991, 10.1145/127601.127626.

Erklärung

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

Würzburg, den 09. Mai 2022

. . . . . .
Florian Timpf