Fakultät für Mathematik und Informatik
Lehrgebiet Diskrete Mathematik und Optimierung

BACHELORARBEIT

Das Teilwortproblem für einige Heuristiken für das Binary-Paint-Shop-Problem

---

Christiane Brunner

Bearbeitungsbeginn: 15. Juli 2016

Betreuer: Prof. Dr. Winfried Hochstättler

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Automobilindustrie hat Interesse daran eine möglichst effiziente Fahrzeugherstellung zu betreiben. Wird die Produktion der einzelnen Fahrzeuge allein durch die Bestellungen der Kunden getriggert kann eine suboptimale Reihenfolge der einzelnen Prozessschritte entstehen. Wir betrachten dazu den Fertigungsabschnitt Lackiererei, den sogenannten Paint-Shop. *Th. Epping, W. Hochstättler* und *P. Oertel* formulierten in diesem Zusammenhang im Jahr 2004 das Paint-Shop-Problem und im Jahr 2006 den Spezialfall, Binary-Paint-Shop-Problem. Diese mathematischen Modelle beschreiben die Herausforderungen, die vom Kunden bestellte Lackfarbe den verschiedenen Karosserie-Typen so zuzuordnen, dass die Farbe in der Lackiererei so wenig wie möglich gewechselt werden muss, wobei die Reihenfolge der Karosserien nicht geändert wird.

In dieser Arbeit betrachten wir drei Heuristiken für das Binary-Paint-Shop-Problem. Heuristiken sind effiziente Methoden, die innerhalb kurzer Zeit und mit möglichst geringem Aufwand eine zulässige Lösung finden, wobei dies nicht die optimale Lösung sein muss. Nachdem wir die Vorgehensweise der einzelnen Algorithmen besprochen haben, werden wir untersuchen für welche Fälle, d.h. welche Reihenfolgen von Fahrzeugkarosserien, eine optimale Lösung erzeugt wird. Diese Sequenzen von Karosserien bezeichnen wir in der mathematischen Betrachtung als Wort, wobei die Buchstaben die Fahrzeugkarosserien darstellen. Wir werden untersuchen für welche Wörter und Teilwörter keine optimalen Lösungen durch die vorgestellten Heuristiken erzeugt werden und Voraussetzungen beschreiben, unter denen die Optimalität gegeben ist. Die zentrale Frage dieser Arbeit lautet: Für welche zulässigen Wörter des Binary-Paint-Shop-Problems ist die Lösung der betrachteten Heuristiken in allen Teilwörtern optimal?

Im Kapitel 2 führen wir das Paint-Shop-Problem und das Binary-Paint-Shop-Problem ein. Wir werden zunächst die zugrundeliegende Problematik in der Praxis vorstellen, um dann die formalen Definitionen der mathematischen Modelle anzugeben. Anschließend definieren wir einige Begrifflichkeiten, die zum Verständnis dieser Arbeit benötigt werden.

Im Kapitel 3 zeigen wir drei Heuristiken für das Binary-Paint-Shop-Problem, den Greedy-Algorithmus, den Red-First-Algorithmus und den rekursiven Greedy-Algorithmus. Wir erläutern und veranschaulichen deren Vorgehensweisen und gehen anschließend auf ihre Charakteristika ein. *F. Meunier, A. Sebő* und *H. Amini et al.* zeigten 2009 Bedingungen auf, unter denen der

Greedy-Algorithmus eine optimale Lösung liefert [4]. *D. Rautenbach* und *Z. Szigeti* erweiterten im Jahr 2012 diese Ergebnisse und charakterisierten den Greedy-Algorithmus für das Binary-Paint-Shop-Problem in [2]. Wir werden die Ergebnisse dieser Arbeiten in Abschnitt 3.1.2 vorstellen. In Abschnitt 3.2.2 charakterisieren wir den Red-First-Algorithmus, der im Jahr 2011 von *S.D. Andres* und *W. Hochstättler* veröffentlicht wurde [1]. In Abschnitt 3.3.3 gehen wir auf den rekursiven Greedy-Algorithmus ein, der ebenfalls von *S.D. Andres* und *W. Hochstättler* entwickelt wurde [1]. Wir zeigen die bisherigen Ergebnisse der Eigenschaften des rekursiven Greedy-Algorithmus, die vermuten lassen, dass dieser um einige Klassen besser ist als die beiden anderen. Die weitere Arbeit widmet sich der Untersuchung der Optimalität der rekursiven Greedy-Färbungen von zulässigen Wörtern des Binary-Paint-Shop-Problems. Wir werden die Färbungen aller Wörter bestimmter Längen analysieren und bestimmen für welche Wörter beziehungsweise Teilwörter keine optimale Lösung erzeugt wird.

Im Kapitel 4 beschreiben wir das Programm, das wir für die Analyse der rekursiven Greedy-Färbungen entwickelt haben. Außerdem zeigen wir die Vorgehensweise der Nachbewertung der Programmausgabe und stellen abschließend die Ergebnisse vor.

Im Kapitel 5 analysieren wir die Ergebnisse aus Kapitel 4 und untersuchen den rekursiven Greedy-Algorithmus. Die ursprüngliche Vermutung, dass wir die Beweisstruktur der Charakterisierung des Greedy-Algorithmus auf den rekursiven Greedy-Algorithmus adaptieren können, um diesen zu charakterisieren, werden wir widerlegen. Dazu untersuchen wir verschiedene Definitionen von Hypergraphen bezüglich einer rekursiven Greedy-Färbung. Wir werden in diesem Kapitel die Ergebnisse, die zu diesem Schluss führen dokumentieren. Eine solche Dokumentation der Resultate kann wichtig sein für die weitere Arbeit der Charakterisierung des rekursiven Greedy-Algorithmus. Abschließend werden wir die Wörter vorstellen, die diese Vermutung widerlegen. Für diese Wörter gibt es keinen Hypergraphen bezüglich der rekursiven Greedy-Färbung, der die Optimalität der rekursiven Greedy-Färbung beweist.

Im Kapitel 6 führen wir den Begriff des Odd-Row-Sum-Packings ein. Wir werden zeigen, dass die geradzahlig laminaren Familien von Hypergraphen ein Spezialfall eines Odd-Row-Sum-Packings sind und alle Wörter aus Kapitel 5 ein Odd-Row-Sum-Packing besitzen. Dies führt uns zur Hypothese, dass man für jedes in allen Teilwörtern optimal gefärbte Wort ein Odd-Row-Sum-Packing, abhängig von der rekursiven Greedy-Färbung, finden kann. Wir werden die Idee zur Erzeugung solcher Odd-Row-Sum-Packings präzisieren und ein Odd-Row-Sum-Packing abhängig von der rekursiven Greedy-Färbung für alle Wörter aus Kapitel 5 finden. Diese Ergebnisse bilden die

Grundlage zur Weiterverfolgung der Vermutung die Optimalität einer rekursiven Greedy-Färbung mit Hilfe von Odd-Row-Sum-Packings zu beweisen und damit den rekursiven Greedy-Algorithmus zu charakterisieren.

2 Das Binary-Paint-Shop-Problem

In diesem Kapitel beschreiben wir das Paint-Shop-Problem sowie das daraus abgeleitete Binary-Paint-Shop-Problem. Wir werden zunächst auf die Problematik in der Praxis eingehen, die zur Formulierung dieses mathematischen Modells geführt hat und anschließend die formalen Definitionen angeben. Außerdem werden weitere Begrifflichkeiten eingeführt, die für diese Arbeit wichtig sind. Die folgenden Ausführungen und Definitionen orientieren sich, soweit nicht anders angegeben an [3].

Die Motivation zur Formulierung des Paint-Shop- und des Binary-Paint-Shop-Problems finden wir in der Optimierung der Fertigungsabläufe in der Automobilindustrie.

Am Automobilmarkt steigt die Nachfrage nach individuell ausgestatteten Fahrzeugvarianten. Um diesem Bedarf nachzukommen bieten Hersteller eine immer größere Auswahl von Austattungs- und Karosserievarianten an. Da eine Vorproduktion dieser unterschiedlichsten Typen hohe Lagerkosten verursachen würde, werden Fahrzeuge nur nach Bestellung gefertigt, um den individuellen Wünschen des Kunden gerecht zu werden. Die Bestellungen der Kunden bestimmen die Reihenfolge der zu produzierenden Fahrzeugkarosserien, wodurch eine sequenzielle Produktionsplanung nicht möglich ist. Dies hat zur Folge, dass Fahrzeugkarosserien gleichen Typs nicht in regelmäßigen Abständen gefertigt werden können [12]. Solch eine sequenzielle Abarbeitung hätte einen positiven Einfluss auf Qualität sowie Kosten der zu durchlaufenden Schritte einer Fertigung, wie zum Beispiel das Presswerk, den Karosseriebau und die Fahrzeugmontage.

Im Folgenden gehen wir auf den Fertigungsabschnitt der Lackiererei, im Englischen Paint-Shop, ein. In diesem Schritt passieren die Fahrzeugkarosserien unterschiedliche Kammern zur Reinigung, Grundierung und Lackierung der Endfarbe [6]. Ein Farbwechsel tritt auf, wenn zwei aufeinanderfolgende Karosserien mit unterschiedlicher Endfarbe den Paint-Shop durchlaufen. Solche Farbwechsel erfordern eine Reinigung der Lackierroboter und führen zu einem erhöhten Maß an verunreinigtem Abwasser, zu Umrüstzeiten und somit zu zusätzlichen Kosten. Es liegt im hohen Interesse der Automobilindustrie die Anzahl der Farbwechsel zu reduzieren, um dadurch die Produktionskosten zu senken und die Nachhaltigkeit der Fertigungsschritte zu verbessern.

Eine Möglichkeit die Anzahl der Farbwechsel zu reduzieren bietet die Anwendung von Sortier-Heuristiken mit Zwischenlager-Systemen. Dieses System gruppiert die Karosserien nach der Lackfarbe. Ein Beispiel hierfür ist das Line-Storage-System [5], welches eine Sortierung vornimmt, indem es eine Eingangssequenz von Karosserien auf eine bestimmte Anzahl von Sor-

tierbändern der Farbe nach verteilt. Anschließend werden die sortierten Karosserien zu einer der Endfarbe nach gruppierten Ausgangssequenz vereinigt.

Es sprechen mehrere Gründe dafür diesen Ansatz von Farb-Lager-Systemen zu verwerfen und die Sequenz der Fahrzeugkarosserien als einen externen, unveränderbaren Parameter zu betrachten. Einer dieser Gründe ist eine suboptimale Reihenfolge der Karossieren für vorgehende und nachfolgende Prozessschritte verursacht durch eine optimale Sortierung nach der Lackfarbe. Ein weiterer Grund ist die Gewinnung von mehr Flexibilität in der Produktion, indem die Karosserie und deren Lackfarbe voneinander getrennt betrachtet werden. Dies kann mit Hilfe von Mikrochips realisiert werden, die auf den Karosserien angebracht sind und deren Eigenschaften online steuern und dokumentieren.

Aus diesen vielfältigen Anforderungen für eine optimale Fertigungsreihenfolge von Fahrzeugkarosserien ergibt sich ein kombinatorisches Sequenzierungsproblem, das wir durch das Paint-Shop-Problem und das Binary-Paint-Shop-Problem beschreiben.

Wir geben nachfolgend die formale Definition an. Ziel ist es eine Farbsequenz einer Menge von Fertigungsaufträgen zu ermitteln, sodass die Anzahl der Farbwechsel innerhalb dieser Sequenz minimal ist.

Wir nehmen an, dass vorab eine beliebige Sequenz von Fertigungsaufträgen festgelegt wurde. Jede Fahrzeugkarosserie wird durch einen Buchstaben eines abzählbar unendlichen Alphabets Σ dargestellt und eine Sequenz von Karosserien entsprechend mit einem Wort w. Alle im Folgenden betrachteten Wörter w sind endlich. Die Sequenz der Farben, also die Färbung des Wortes w ist durch einen Vektor f der Länge des Wortes w gegeben, wobei f_i die Farbe von w_i für alle $i \in \mathbb{N}$ bezeichnet. Ein Farbwechsel tritt auf, wenn $f_i \neq f_{i+1}$ gilt. Das Paint-Shop-Problem besteht darin eine Permutation der Färbung zu finden, welche die Anzahl der Farbwechsel in f minimiert, wobei die Anordnung von w unverändert bleibt.

Definition 2.1. *(Paint-Shop-Problem für Wörter) [3]. Gegeben sei ein endliches Alphabet Σ, ein Wort $w = (w_1, ..., w_n) \in \Sigma^*$, eine endliche Menge von Farben F und eine Färbung $f = (f_1, ..., f_n)$ des Wortes w mit $f_i \in F$ für $i = 1, ..., n$. Finde eine Permutation $\sigma : 1, ..., n \to 1, ..., n$, sodass $w_{\sigma(i)} = w_i$ für $i = 1, ..., n$ und die Anzahl der Farbwechsel in $\sigma(f) = (f_{\sigma(1)}, ..., f_{\sigma(n)})$ minimal ist.*

Aus dem allgemeinen Fall des Paint-Shop-Problems leiten wir das Binary-Paint-Shop-Problem ab [4]. In diesem Fall kommt jede Fahrzeugkarosserie genau zweimal innerhalb einer Sequenz vor und wird mit zwei unterschiedlichen Farben lackiert. Auch hierbei ist das Ziel die Anzahl der Farbwechsel zu

minimieren, indem wir die Farben den beiden Vorkommen der Karosserien optimal zuordnen. Für die mathematische Betrachtung bezeichnen wir die Farben mit 0 und 1. Die folgenden Definitionen orientieren sich, soweit nicht anders angegeben an [1].

Definition 2.2. *(Zulässiges Wort). Sei $n \in \mathbb{N}$ und $n > 0$. Ein Wort w, in dem jeder Buchstabe eines abzählbar unendlichen Alphabets Σ entweder genau zweimal oder gar nicht auftritt heißt zulässiges Wort der Länge $2n$ aus n Buchstaben.*

Wir betrachten im folgenden zulässige Wörter, deren Länge $2n$ endlich ist.

Definition 2.3. *(0/1-Färbung). Eine Färbung $f = (f_1, ..., f_{2n})$ eines Wortes $w = (w_1, ..., w_{2n})$ mit 0 und 1 heißt 0/1-Färbung.*

Definition 2.4. *(Farbwechsel). Sei $f = (f_1, ..., f_{2n}) \in \{0,1\}^{2n}$ eine 0/1-Färbung eines Wortes $w = (w_1, ..., w_{2n})$. Ein Farbwechsel tritt auf, wenn zwei aufeinanderfolgende Positionen der Färbung ungleich sind. Es gilt $f_i \neq f_{i+1}$ für $1 \leq i < 2n$. Wir sagen, es gibt einen Farbwechsel zwischen i und $i+1$ an Stelle $(i + \frac{1}{2})$.*

Definition 2.5. *(Gute Färbung). Eine 0/1-Färbung $f = (f_1, ..., f_{2n}) \in \{0,1\}^{2n}$ eines zulässigen Wortes w heißt gut, wenn beide Vorkommen eines Buchstabens eines Wortes $w = (w_1, ..., w_{2n})$ unterschiedlich gefärbt sind, d.h. wenn $w_i = w_j$ mit $i \neq j$ für alle $i, j \in \{1, ..., 2n\}$ impliziert $f_i \neq f_j$.*

Das Binary-Paint-Shop-Problem beschreibt die Aufgabe, eine gute 0/1-Färbung für ein zulässiges Wort w zu finden, sodass die Anzahl der Farbwechsel minimal ist.

Definition 2.6. *(Optimale Färbung). Eine gute 0/1-Färbung $f = (f_1, ..., f_{2n}) \in \{0,1\}^{2n}$ eines zulässigen Wortes w heißt optimal, wenn die Anzahl der Farbwechsel k mit $k \in \mathbb{N}$ minimal ist.*

Definition 2.7. *(Binary-Paint-Shop-Problem) [3]. Gegeben ist ein endliches Alphabet Σ der Kardinalität n, dessen Elemente Buchstaben heißen, ein Wort $w = (w_1, ..., w_{2n}) \in \Sigma 2n$ wobei jeder Buchstabe im Wort w genau zweimal vorkommt. Finde eine 0/1-Färbung $f = (f_1, ..., f_{2n}) \in \{0,1\}^{2n}$ von w, die beide Vorkommen jedes Buchstabens unterschiedlich färbt, d.h. $w_i = w_j$ und $i \neq j$ impliziert $f_i \neq f_j$ für alle $i, j \in \{1, ..., 2n\}$, sodass die Anzahl der Farbwechsel in w minimal ist.*

Die folgende Abbildung veranschaulicht die Färbung eines Wortes. Wir nutzen die Farben rot beziehungsweise blau zur besseren Sichtbarkeit einer Färbung mit 0 beziehungsweise 1.

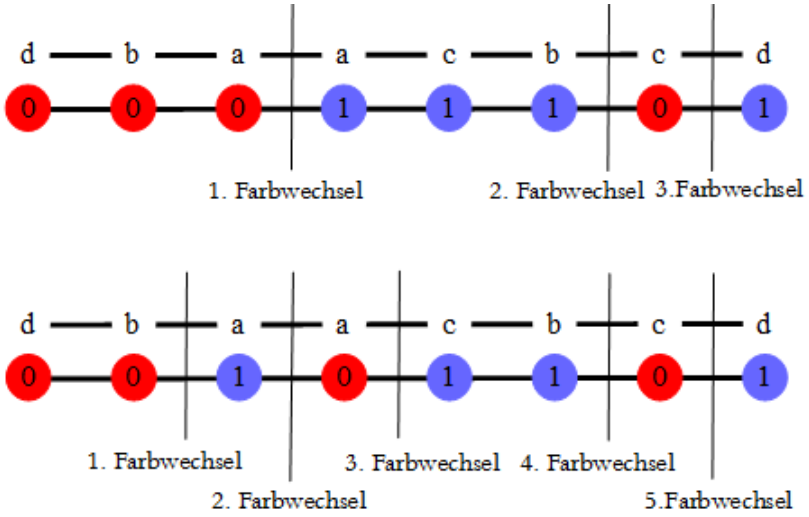

Abbildung 2.1: Die obere Darstellung zeigt eine optimale Färbung des Wortes $w = (d, b, a, a, c, b, c, d)$ mit drei Farbwechsel. Im unteren Teil sehen wir die suboptimale Färbung des Wortes mit fünf Farbwechsel.

3 Heuristiken für das Binary-Paint-Shop-Problem

Zur Lösung von Minimierungsproblemen, wie dem Binary-Paint-Shop-Problem, gibt es sogenannte ρ-Approximationsalgorithmen. Diese berechnen für jeden Fall des Problems mit optimaler Lösung k, eine Lösung ρk, wobei $\rho = (1+\epsilon)$ für jedes $\epsilon > 0$ ist. Wir nennen einen Algorithmus ein Polynomial Time Approximation Scheme, kurz PTAS [10], wenn dieser für ein Minimierungsproblem polynominal in n und $\frac{1}{\epsilon}$ für jedes $\epsilon > 0$ ist. $\mathcal{APX}$ bezeichnet die Klasse der Probleme, für die ein Polynomialzeit-ρ-Approximationsalgorithmus für ein $\rho > 1$ existiert. Ein Problem heißt $\mathcal{APX}$-schwer, wenn die Existenz eines PTAS für dieses Problem impliziert, dass für jedes Problem in $\mathcal{APX}$ ein PTAS existiert [7].

Im Binary-Paint-Shop-Problem ist das Auftreten eines Buchstabens und die Anzahl der Farben auf zwei beschränkt. Dieses Problem ist $\mathcal{NP}$-vollständig [3] und $\mathcal{APX}$-schwer [7]. Für das Binary-Paint-Shop-Problem gibt es, unter Beachtung der Unique-Games-Conjecture [11], keine Approximation mit einem konstanten Faktor, wenn $\mathcal{P} \neq \mathcal{NP}$ gilt.

Wir beschreiben in diesem Kapitel drei Heuristiken zur Lösung des Binary-Paint-Shop-Problems und deren Charakteristika. Diese Heuristiken färben die Wörter des Binary-Paint-Shop-Problems auf unterschiedliche Weise. Dabei gibt es Wörter beziehungsweise Teilwörter deren Färbung durch die jeweilige Heuristik nicht optimal ist. Wir nennen diese Wörter für den jeweiligen Algorithmus verbotene Wörter.

Definition 3.1. *(Verbotenes Wort). Wird ein zulässiges Wort w des Binary-Paint-Shop-Problems durch einen Algorithmus nicht in allen Teilwörtern optimal gefärbt nennen wir dieses Wort für diesen Algorithmus ein verbotenes Wort.*

3.1 Der Greedy-Algorithmus

3.1.1 Die Vorgehensweise des Algorithmus

Um ein zulässiges Wort w des Binary-Paint-Shop-Problems zu färben, existiert ein natürlicher Greedy-Algorithmus [4]. Ein Greedy-Algorithmus generiert eine Lösung, indem er die bestmögliche Wahl, für die in diesem Schritt betrachtete Situation, trifft und diese Entscheidung auch in späteren Schritten nicht mehr revidiert. Der Begriff Greedy, im Deutschen gierig, ist in diesem Zusammenhang ein feststehender Begriff. Zur Färbung eines zulässigen Wortes $w = (w_1, ..., w_{2n})$ mit Hilfe des Greedy-Algorithmus wird das Wort von links nach rechts durchlaufen und der erste Buchstabe mit 0 gefärbt. Anschließend wird das erste Vorkommen jedes Buchstabens mit der Farbe seines Vorgängers gefärbt und das zweite Vorkommen des Buchstabens

entsprechend in der anderen Farbe. Es tritt also nur dann ein Farbwechsel auf, wenn das zweite Vorkommen eines Buchstabens in einer anderen Farbe als sein Vorgänger gefärbt werden muss. Durch dieses Prinzip wird sichergestellt, dass beide Vorkommen eines Buchstabens unterschiedlich gefärbt sind. Es gilt somit folgende Aussage:

Proposition 3.2. *[2]. Die Färbung des Greedy-Algorithmus ist für jedes zulässige Wort w gut.*

Eine Färbung, die durch den Greedy-Algorithmus für ein zulässiges Wort $w = (w_1, ..., w_{2n})$ erzeugt wird, nennen wir Greedy-Färbung $g = (g_1, ..., g_{2n}) \in \{0,1\}^{2n}$. Nachfolgend geben wir einen Pseudocode an.

Algorithm 1 Greedy-Algorithmus

```
1: g_1 = 0;
2: i = 2;
3: while i ≤ 2n do
4:     if w_i ist das erste Vorkommen des Buchstaben then
5:         setze g_i = g_{i-1}
6:     else
7:         setze g_i ≠ g_j für w_i = w_j mit i ≠ j
8:     end if
9:     i = i + 1;
10: end while
```

Zur Veranschaulichung zeigen wir in der folgenden Darstellung, am Beispiel des Wortes $w = (d, a, a, b, c, b, c, d)$, die Vorgehensweise des Greedy-Algorithmus . Es wird in drei Zeilen jeweils die Färbung bis zu einem Farbwechsel dargestellt.

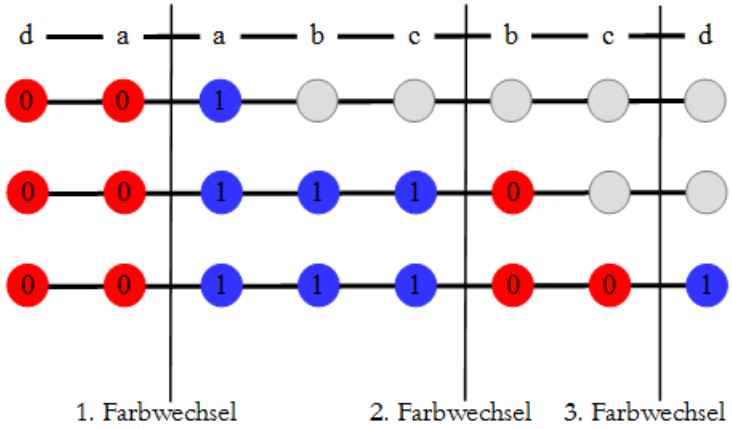

Abbildung 3.2: Die Abbildung zeigt die optimale Färbung des Wortes $w = (d, a, a, b, c, b, c, d)$ mit 0 (rot) und 1 (blau) durch den Greedy-Algorithmus.

3.1.2 Die Charakteristik des Greedy-Algorithmus

Der Greedy-Algorithmus ist optimal, wenn alle Wörter und Teilwörter dem fifo-Prinzip folgen [4]. Diese first in - first out-Regel ist erfüllt, wenn für einen Buchstaben, dessen erstes Vorkommen nach dem ersten Vorkommen eines anderen Buchstabens auftritt und auch dessen zweites Vorkommen nach dem zweiten Vorkommen des anderen Buchstabens auftritt. Betrachtet man nun die Anwendung in der Automobilproduktion ist dieses Prinzip erfüllt, wenn eine Fahrzeugkarosserie, die zu Beginn einer Teilsequenz lackiert wurde auch in der Teilsequenz der zweiten Vorkommen zuerst den Paint-Shop durchläuft. Diese Bedingung lässt sich auch so formulieren, dass ein Wort w kein Teilwort der Form (a, b, b, a) enthält [8]. Dieses Ergebnis wird durch den folgenden Satz auf zwei Wörter der Länge sechs erweitert.

Satz 3.3. *[4]. Die Greedy-Färbung ist für jedes zulässige Wort w des Binary-Paint-Shop-Problems optimal, wenn w kein Teilwort der Form (a, b, a, c, c, b) oder (a, b, b, c, a, c) enthält.*

Beweis. Den vollständigen Beweis finden wir in [4], Kapitel 2. □

D. Rautenbach und *Z. Szigeti* erweitern dieses Ergebnis und charakterisieren den Greedy-Algorithmus für das Binary-Paint-Shop-Problem [2]. Zunächst definieren wir die nötigen Begriffe.

Definition 3.4. *(Hypergraph) (siehe z.B. [9]). Ein Hypergraph $\mathcal{H}$ ist ein Paar $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ mit:*

(i) $\mathcal{V}$ ist eine endliche Menge von .

(ii) $\mathcal{E}$ ist eine endliche Menge von Hyperkanten. Eine Hyperkante ist eine nicht-leere Teilmenge aus $\mathcal{V}$.

Definition 3.5. *(Laminarer Hypergraph) [4]. Ein laminarer Hypergraph ist ein Hypergraph $\mathcal{H}$ der bezüglich Inklusion eine Baumstruktur enthält. Seien $A, B \subseteq \mathcal{H}$, dann gilt eine der beiden Aussagen:*

(i) $A \subseteq B$ oder $B \subseteq A$.

(ii) $A \cap B = \emptyset$.

Definition 3.6. *(Geradzahlig laminarer Hypergraph) [4]. Ein Hypergraph $\mathcal{H}$ heißt geradzahlig laminar, wenn die beiden folgenden Aussagen gelten:*

(i) $\mathcal{H}$ ist laminar,

(ii) für alle $A \in \mathcal{H}$ gibt es eine gerade Anzahl von Teilmengen in $\mathcal{H}$, die in A enthalten und von A unterschiedlich sind.

Proposition 3.7. *Für jede gute Färbung $f = (f_1, ..., f_{2n})$ eines zulässigen Wortes $w = (w_1, ..., w_{2n})$ gilt, dass zwischen den beiden Vorkommen $w_i = w_j$ mit $i \neq j$ und $1 \leq i < j \leq 2n$ eines Buchstabens, die Anzahl der Farbwechsel ungerade ist.*

Beweis. Die Aussage ist evident. □

Seien $i, j \in \mathbb{N}$ mit $1 \leq i < i+1 < \cdots < j-1 < j \leq 2n$ die Positionen der beiden Vorkommen eines Buchstaben in einem zulässigen Wort $w = (w_1, ..., w_{2n})$. Wir geben die Intervalle $I(i, j)$ und die Menge dieser Intervalle $l(w)$ wie folgt an:

$$I(i,j) = \left\{ i + \tfrac{1}{2}, (i+1) + \tfrac{1}{2}, ..., (j-1) + \tfrac{1}{2} \right\}$$

und

$$l(w) = \{I(i,j) | 1 \leq i < j \leq 2n, w_i = w_j \text{ mit } i \neq j\}.$$

In der folgenden Abbildung sehen wir eine grafische Darstellung der Intervalle $I(i, j)$ am Beispiel des zulässigen Wortes $w = (a, b, d, a, d, c, c, b)$.

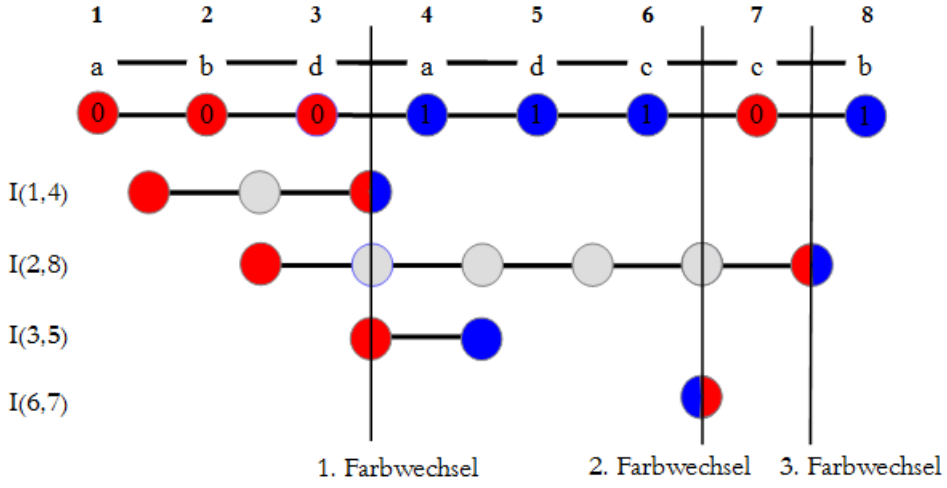

Abbildung 3.3: Die grafische Darstellung der Intervalle $I(i, j)$ eines zulässigen Wortes $w = (a, b, d, a, d, c, c, b)$ mit optimaler Greedy-Färbung.

Sei $\mathcal{G}(w)$ ein Hypergraph, der bezüglich einer guten Färbung des Greedy-Algorithmus definiert ist. Die Hyperkanten von $\mathcal{G}(w)$ sind die Intervalle $I(i, j)$ der Buchstaben eines Wortes w, vor deren zweiten Vorkommen ein Farbwechsel auftritt.

$$\mathcal{G}(w) = \{I(i,j) \in l(w) | g_{j-1} \neq g_j\}$$

Mit diesen Voraussetzungen wird nun der Greedy-Algorithmus mit folgendem Satz charakterisiert.

Satz 3.8. *[2]. Für jedes zulässige Wort w, sind folgende Aussagen äquivalent.*

(i) Die Greedy-Färbung ist für jedes zulässige Teilwort w' von w optimal.

(ii) $G(w')$ ist ein geradzahlig laminarer Hypergraph für jedes zulässige Teilwort w' von w.

(iii) w enthält keines der drei Wörter (a, b, a, c, c, b), (a, d, d, b, c, c, a, b) und (a, d, d, c, b, c, a, b) als ein Teilwort.

Wir beweisen nun den Satz 3.8 mit einem Ringbeweis orientiert an [2].

Beweis. Wir beweisen die drei Implikationen $(i) \Rightarrow (iii)$, $(iii) \Rightarrow (ii)$ und $(ii) \Rightarrow (i)$.

$(i) \Rightarrow (iii)$: Seien die drei verbotenen Wörter $w_1 = (a, b, a, c, c, b)$, $w_2 = (a, d, d, b, c, c, a, b)$ und $w_3 = (a, d, d, c, b, c, a, b)$ gegeben. Die Greedy-Färbungen beziehungsweise die optimalen Färbungen der Wörter w_i bezeichnen wir mit g_i bwz. f_i für $i = 1, 2, 3$. Es gilt.

$g_1 = (0, 0, 1, 1, 0, 1)$, $g_2 = (0, 0, 1, 1, 1, 0, 1, 0)$ und $g_3 = (0, 0, 1, 1, 1, 0, 1, 0)$

$f_1 = (0, 1, 1, 1, 0, 0)$, $f_2 = (0, 1, 0, 0, 0, 1, 1, 1)$ und $f_3 = (0, 1, 0, 0, 0, 1, 1, 1)$

Ein Vergleich der Anzahl der Farbwechsel von Greedy- und optimaler Färbung der Wörter w_1, w_2 und w_3 zeigt, dass diese Wörter durch den Greedy-Algorithmus nicht optimal gefärbt werden.

$(iii) \Rightarrow (ii)$**:** Gegeben sei ein zulässiges Wort $w = (w_1, ..., w_{2n})$ und dessen Greedy-Färbung $g = (g_1, ..., g_{2n})$. Um die Aussage in (ii) zu zeigen, müssen wir beweisen, dass der Hypergraph $\mathcal{G}(w')$ für jedes Teilwort w' eines Wortes w mit optimaler Greedy-Färbung geradzahlig laminar ist. Da wir voraussetzen, dass die Anzahl der Farbwechsel minimal ist, enthält jede Hyperkante in $\mathcal{G}(w)$ eine ungerade Anzahl von Farbwechsel. Wenn ein Intervall $I(i, j)$ in $G(w)$ ist, dann gilt durch die Konstruktion des Hypergraphen, dass $w_i = w_j$, $g_i \neq g_j$ und $g_{j-1} \neq g_j$ ist. Da für jede gute Färbung eines zulässigen Wortes die Anzahl der Farbwechsel zwischen den beiden Vorkommen eines Buchstabens ungerade ist (Proposition 3.7) und die Greedy-Färbung für jedes zulässige Wort gut ist (Proposition 3.2), folgt daraus, dass die Anzahl der Farbwechsel zwischen i und j ungerade ist. Die Anzahl der Farbwechsel zwischen i und $(j - 1)$ ist somit gerade. Dies impliziert durch die Konstruktion von $\mathcal{G}(w)$, dass $\mathcal{G}(w)$ eine gerade Anzahl von Mengen $I(i', j')$ mit $(j' - 1) + \frac{1}{2} \in I(i, j)$

enthält. Daraus folgt, dass $\mathcal{G}(w)$ geradzahlig laminar ist, wenn $\mathcal{G}(w)$ laminar ist. Wir zeigen nun durch einen Widerspruch, dass $\mathcal{G}(w)$ für jedes Teilwort w' von w laminar ist, wenn w' kein verbotenes Teilwort enthält.
Wir nehmen zwei Mengen $I(a_1, a_2)$, $I(b_1, b_2) \in \mathcal{G}(w)$ mit den folgenden Eigenschaften an:

(i) $I(a_1, a_2) \not\subseteq I(b_1, b_2)$ und $I(a_1, a_2) \not\supseteq I(b_1, b_2)$,

(ii) $I(a_1, a_2) \cap I(b_1, b_2) \neq \emptyset$ und

(iii) $a_1 < b_1 < a_2 < b_2$ und a_1 ist minimal, entsprechend dazu ist b_2 minimal.

Wir zeigen nun, dass $\mathcal{G}(w)$ nur dann laminar ist, wenn w kein verbotenes Teilwort enthält.

Wir beginnen mit dem Wort (a_1, b_1, a_2, b_2) und zeigen folgende Aussage:

Behauptung 1. *Es gibt zwischen a_2 und b_2 einen einzigen Farbwechsel an der Stelle $(b_2 - 1) + \frac{1}{2}$.*

Welche Möglichkeiten gibt es nun ein Intervall $I(c_1, c_2)$ in das Wort w einzufügen, sodass ein zusätzlicher Farbwechsel zwischen a_2 und b_2 verursacht wird? Wenn wir die beiden Vorkommen von c zwischen a_2 und b_2 einfügen, erhalten wir das verbotene Wort $(a_1, b_1, a_2, c_1, c_2, b_2)$. Dies ist ein Widerspruch zur Voraussetzung, dass dieses verbotene Wort nicht enthalten ist. Sei nun $a_1 < c_1 < a_2$, dann gilt $a_1 < c_1 < a_2 < c_2$ und damit $c_2 < b_2$. Dies ist ein Widerspruch zur angenommenen Minimalität von b_2. Wenn wir $c_1 < a_1$ annehmen folgt mit der vorausgesetzten Eigenschaft (iii) $c_1 < b_1 < c_2 < b_2$ und damit ein Widerspruch zur angenommenen Minimalität von a_1. Wir haben gezeigt, dass es zwischen a_2 und b_2 nur einen Farbwechsel an der Stelle $(b_2 - 1) + \frac{1}{2}$ geben kann.

Mit Proposition 3.2, Proposition 3.7 und Behauptung 1 wissen wir, dass es einen einzigen Farbwechsel zwischen b_1 und $(a_2 - 1)$ gibt. Damit gibt es eine Hyperkante $I(c_1, c_2)$ in $\mathcal{G}(w)$ mit $b_1 < c_2 < a_2$. Wir nehmen c_2 maximal an und zeigen, wie zuvor, die folgende Behauptung:

Behauptung 2. *Es gibt zwischen c_2 und a_2 einen einzigen Farbwechsel an Stelle $(a_2 - 1) + \frac{1}{2}$.*

Wir prüfen nun die verschiedenen Möglichkeiten, die durch ein Intervall $I(d_1, d_2)$ einen zusätzlichen Farbwechsel zwischen c_2 und a_2 verursachen können. Da wir a_1 minimal annehmen, gilt $a_1 < c_1$. Es gilt außerdem:

Behauptung 3. *Seien I_1 und I_2 zwei Mengen in $\mathcal{G}(w)$, dann ist die Anzahl der Farbwechsel in $(I_1 \backslash I_2) \cup (I_2 \backslash I_1)$ gerade.*

Beweis. Die Anzahl der Farbwechsel in $(I_1 \backslash I_2) \cup (I_2 \backslash I_1)$ ist gleich der ungeraden Anzahl der Farbwechsel in I_1, plus der ungeraden Anzahl der Farbwechsel in I_2, minus der doppelten Anzahl der Farbwechsel der Schnittmenge $(I_1 \cap I_2)$. Die Parität von $(I_1 \backslash I_2) \cup (I_2 \backslash I_1)$ ist damit gerade, da wir zwei ungerade Zahlen addieren und eine gerade Zahl subtrahieren. □

Zunächst nehmen wir $b_1 < c_1$ an. Mit Proposition 3.7, Behauptung 1 und 3 gibt es eine Menge $I(d_1, d_2)$ in $\mathcal{G}(w)$ mit $a_1 < d_2 < b_1$. Wenn $d_1 < a_1$ ist, dann gilt $d_1 < a_1 < d_2 < a_2$ und $d_1 < a_1$ ist ein Widerspruch zur Minimalität von a_1. Mit der vorausgesetzten Eigenschaft (iii) erhalten wir $a_1 < d_1 < d_2 < b_1 < c_1 < c_2 < a_2 < b_2$. Dies ist ein Widerspruch zur Annahme, dass kein verbotenes Wort enthalten ist.
Zuletzt nehmen wir $c_1 < b_1$ an. Mit Behauptung 2 und 3 gibt es eine Menge $I(d_1, d_2)$ in $\mathcal{G}(w)$ mit $a_1 < d_2 < c_1$. Da wir a_1 minimal annehmen, ist $d_1 > a_1$. Wenn wir die beiden Vorkommen in das Wort einfügen, folgt die Anordnung $a_1 < d_1 < d_2 < c_1 < b_1 < c_2 < a_2 < b_2$ und dadurch ein Widerspruch zur Annahme, dass das verbotene Wort $(a_1, d_1, d_2, c_1, b_1, c_2, a_2, b_2)$ nicht enthalten ist.

Damit haben wir gezeigt, dass $\mathcal{G}(w')$ für jedes Teilwort w' von w geradzahlig laminar ist, wenn w kein verbotenes Wort enthält.

$(ii) \Rightarrow (i)$**:** Gegeben sei folgende Definition.

Definition 3.9. *(Ungerade Transversale) [4]. Eine ungerade Transversale T eines Hypergraphen $\mathcal{H}$ ist eine Teilmenge seiner Knotenmenge, die jede Hyperkante von $\mathcal{H}$ in einer ungeraden Anzahl von Elementen schneidet.*

Die Implikation $(ii) \Rightarrow (i)$ folgt mit Lemma 2 aus [4]. Dies besagt, dass eine ungerade Transversale T eines geradzahlig laminaren Hypergraphen $\mathcal{H}$ mindestens so viele Elemente enthält, wie es Hyperkanten in $\mathcal{H}$ gibt. Damit gilt $|T| \geq |\mathcal{E}(\mathcal{H})|$. Dies zeigen wir folgendermaßen. Wir können, gegebenfalls durch das Einfügen eines neuen Knotens annehmen, dass jede Hyperkante in $\mathcal{H}$ eine echte Teilmenge der Knotenmenge von $\mathcal{H}$ ist.

Behauptung 4. *Sei e eine Menge von Knoten in $\mathcal{H}$, die k Hyperkanten echt enthält, dann gilt $|T \cap e| \geq k$.*

Beweis. Wir zeigen diese Behauptung durch Induktion von k. Für $k = 0$ ist die Aussage trivial. Sei $k > 0$. Seien $e_1, \ldots, e_l$ die maximalen Hyperkanten, die in e echt enthalten sind. Sei k_i mit $1 \leq i \leq l$ die Anzahl der Hyperkanten, die in e echt enthalten sind. Offensichtlich gilt $k = (k_1 + 1) + \cdots + (k_l + 1)$. Da $\mathcal{H}$ geradzahlig laminar ist, ist k_i gerade für jedes $1 \leq i \leq l$. Da T eine ungerade Transversale ist, ist $|T \cap e_i|$ ungerade für jedes $1 \leq i \leq l$. Damit folgt mit Induktion, $|T \cap e_i| \geq k_i + 1$ und wir erhalten $|T \cap e_i| \geq |T \cap (e_1 \cup \cdots \cup e_l)| = |T \cap e_1| + \cdots + |T \cap e_l| \geq (k_1 + 1) + \cdots + (k_l + 1) = k$. □

Wenden wir nun Behauptung 4 auf die Knotenmenge von $\mathcal{H}$ an, folgt daraus $|T| \geq |E(\mathcal{H})|$. Sei $w = (w_1, \ldots, w_{2n})$ ein zulässiges Wort. Wir zeigen nun die Aussage für $\mathcal{G}(w)$ mit dem Argument, dass die Greedy-Färbung $g = (g_1, \ldots, g_{2n})$ von w höchstens so viele Farbwechsel, wie eine andere gute Färbung $f = (f_1, \ldots, f_{2n})$, hat. Sei

$G = \{i + \frac{1}{2} | 1 \leq i \leq 2n-1, g_i \neq g_{i+1}\}$ und
$F = \{i + \frac{1}{2} | 1 \leq i \leq 2n-1, f_i \neq f_{i+1}\}$.

Mit dieser Definition gilt $|G| = |\mathcal{G}(w)|$. Mit Proposition 3.7 ist F eine ungerade Transversale des geradzahlig laminaren Hypergraphen $\mathcal{G}(w)$. Es folgt $|F| \geq |\mathcal{G}(w)|$ und damit $|F| \geq |G|$.

Damit ist der Beweis von Satz 3.8 vollständig. □

Wir veranschaulichen das Prinzip eines geradzahlig laminaren Hypergraphen aus dem Beweis von Satz 3.8 an folgender Darstellung des nicht laminaren Hypergraphen des Wortes $w = (a, b, d, a, d, c, c, b)$, wobei das verbotene Wort $w' = (a, b, a, c, c, b)$ als Teilwort enthalten ist. Die Abbildung zeigt die Greedy-Färbung des Wortes und die drei Intervalle der Buchstaben, vor deren zweiten Vorkommen ein Farbwechsel auftritt. Diese Intervalle $I(1, 4)$, $I(2, 8)$ und $I(6, 7)$ bilden die Hyperkanten in $\mathcal{G}(w)$. Wie wir sehen können, ist das Intervall $I(6, 7)$ im Intervall $I(2, 8)$ enthalten. Allerdings sind die beiden Intervalle $I(1, 4)$ und $I(2, 8)$ nicht disjunkt. Damit sind die Voraussetzungen von Definition 3.5 nicht erfüllt und $\mathcal{G}(w)$ ist nicht laminar.

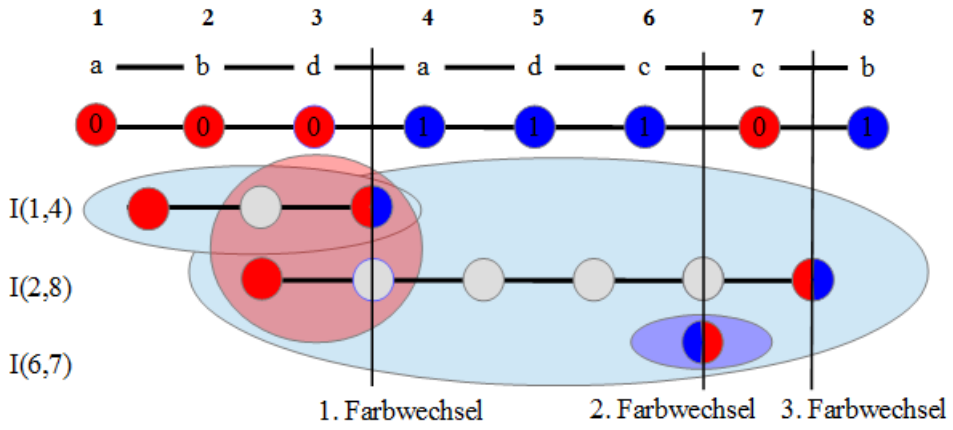

Abbildung 3.4: Die grafische Darstellung des Hypergraphen $\mathcal{G}(w)$ des Wortes $w = (a, b, d, a, d, c, c, b)$. Die Schnittmenge der beiden nicht disjunkten Intervalle wird durch einen roten Kreis dargestellt.

3.2 Der Red-First-Algorithmus

3.2.1 Die Vorgehensweise des Algorithmus

Eine weitere Heuristik ist der sogenannte Red-First-Algorithmus [1]. Mit der Red-First-Strategie wird das erste Vorkommen jedes Buchstabens mit 0 (rot) gefärbt und entsprechend das zweite Auftreten mit 1 (blau). Der Name dieses Algorithmus rührt daher, dass in früheren Arbeiten zum Paint-Shop-Problem die Färbung nicht mit 0 und 1 erfolgte, sondern mit rot und blau. Durch diesen Ansatz werden gewisse Worst-Case Fälle des Greedy-Algorithmus optimal gefärbt [1].

Wir nennen eine Färbung, die durch den Red-First-Algorithmus erzeugt wird, Red-First-Färbung $rf = (rf_1, ..., rf_{2n}) \in \{0,1\}^{2n}$.

Algorithm 2 Red-First-Algorithmus

```
1: rf_1 = 0;
2: i = 2;
3: while i ≤ 2n do
4:     if w_i ist das erste Vorkommen des Buchstaben then
5:         setze rf_i = 0
6:     else
7:         setze rf_i = 1
8:     end if
9:     i = i + 1;
10: end while
```

Sei $w = (w_1, ..., w_{2n})$ ein zulässiges Wort und sei $I(w_i, ..., w_j)$ mit $w_i = w_j$ ein Intervall, das mit den beiden Vorkommen eines Buchstabens abschließt. Der erste Farbwechsel tritt in der Red-First-Färbung beim ersten Auftreten eines zweiten Vorkommens eines Buchstabens auf. Weitere Farbwechsel treten auf, wenn ein Intervall $I(w_i, ..., w_j)$ innerhalb einer Folge erster beziehungsweise zweiter Vorkommen von Buchstaben ist.

In der folgenden Darstellung veranschaulichen wir das Prinzip der Red-First-Färbung anhand des Wortes $w = (d, a, a, b, c, b, c, d)$. Das Wort wird beginnend mit 0 (rot) von links nach rechts gefärbt. Zur besseren Anschaulichkeit wird in jeder Zeile die Färbung bis zum ersten beziehungsweise nächsten Farbwechsel dargestellt. Es tritt in jeder Zeile jeweils ein Farbwechsel auf. Die Farbwechsel werden verursacht durch das zweite Vorkommen von a (Zeile 1), das erste Vorkommen von b (Zeile 2) und das zweite Vorkommen von b (Zeile 3) auf.

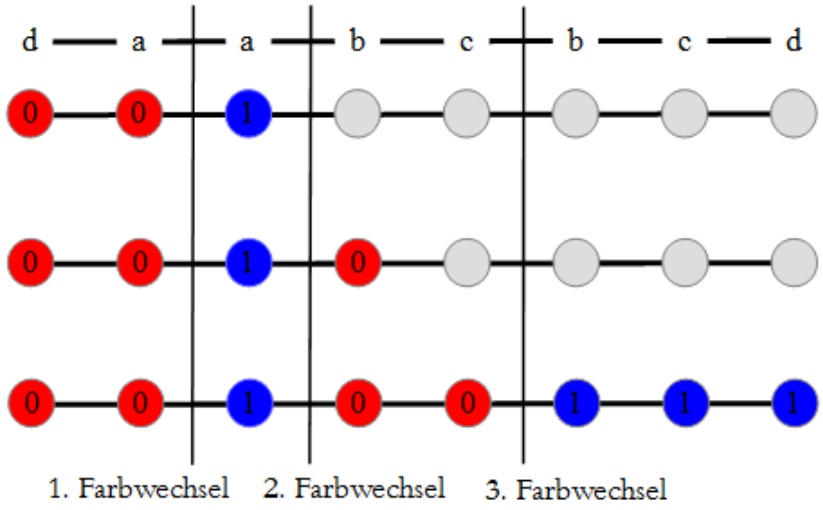

Abbildung 3.5: Die Darstellung zeigt die Red-First-Färbung des Wortes $w = (d, a, a, b, c, b, c, d)$

3.2.2 Die Charakteristik des Red-First-Algorithmus

Mit der Red-First-Strategie tritt, wie in Abschnitt 3.2.1 beschrieben, ein Farbwechsel in der Färbung $rf = (rf_1, ..., rf_{2n})$ eines zulässigen Wortes $w = (w_1, ..., w_{2n})$ auf, wenn für zwei aufeinanderfolgende Buchstaben eine der beiden folgenden Aussagen für $i \in \mathbb{N}$ mit $1 \leq i < 2n$ gilt:

(i) w_i ist das erste Vorkommen eines Buchstabens und w_{i+1} ist das zweite Vorkommen eines Buchstabens.

(ii) w_i ist das zweite Vorkommen eines Buchstabens und w_{i+1} ist das erste Vorkommen eines Buchstabens.

Es tritt ein zusätzlicher Farbwechsel auf, wenn sich ein Intervall $I(w_i, w_j)$ von zwei gleichen Buchstaben $w_i = w_j$ mit $1 \leq i < j \leq 2n$ innerhalb einer Sequenz von ersten beziehungsweise zweiten Vorkommen von Buchstaben des Wortes w befindet. Mit anderen Worten, es tritt ein zusätzlicher Farbwechsel auf, wenn w ein Teilwort der Form (a, a, b, b) enthält. Analog zum Greedy-Algorithmus werden Wörter, die dem fifo-Prinzip folgen, in allen Teilwörtern optimal gefärbt. Wir können den Red-First-Algorithmus mit dem folgenden Satz charakterisieren.

Satz 3.10. *Die Red-First-Färbung $rf = (rf_1, ..., rf_{2n})$ ist für jedes zulässige Wort $w = (w_1, ..., w_{2n})$, das kein Teilwort der Form $w' = (a, a, b, b)$ enthält, optimal.*

Beweis. Sei das zulässige Wort $w = (a, a, b, b)$ gegeben.

Die Red-First-Färbung rf und die optimale Färbung f dieses Wortes sind:

$rf(w) = (0, 1, 0, 1)$,

$f(w) = (0, 1, 1, 0)$

Durch Zählen der Farbwechsel sehen wir, dass die Red-First-Färbung $rf(w)$ drei Farbwechsel enthält, wohingegen die optimale Färbung $f(w)$ nur zwei Farbwechsel benötigt. Die Red-First-Färbung ist somit für $w = (a, a, b, b)$ nicht optimal.

Sei $n \in \mathbb{N}$ und w ein zulässiges Wort der Länge $2n$, das kein Teilwort der Form (a, a, b, b) enthält, dann treten in diesem Wort w zuerst alle n Buchstaben genau einmal auf und anschließend treten alle n Buchstaben zum zweiten Mal auf. Das Wort besteht somit aus einer Sequenz $(w_1, ..., w_n)$ der ersten Vorkommen der n Buchstaben und einer Sequenz $(w_{n+1}, \ldots, w_{2n})$ der zweiten Vorkommen der n Buchstaben. Mit dem Red-First-Algorithmus werden die ersten Vorkommen mit 0 und die zweiten Vorkommen mit 1 gefärbt. Die Red-First-Färbung eines Wortes w, das kein Teilwort (a, a, b, b) enthält, ist daraus folgend

$rf(w) = (w_1 = 0, \ldots, w_{n-1} = 0, w_n = 0, w_{n+1} = 1, \ldots, w_{2n-1} = 1, w_{2n} = 1)$.

Es tritt nur ein einziger Farbwechsel zwischen w_n und w_{n+1} auf. Die Färbung $rf(w)$ ist somit optimal.

□

3.3 Der rekursive Greedy-Algorithmus

3.3.1 Die Vorgehensweise des Algorithmus

Die Betrachtung der beiden Heuristiken in Abschnitt 3.1 und 3.2 hat gezeigt, dass es für die Färbung eines zulässigen Wortes w des Binary-Paint-Shop-Problems nicht optimal ist, wenn nur die Farbe des Vorgängers oder das Vorkommen des Buchstabens im Wort betrachtet wird. Dieses Ergebnis führt zu einer neuen Heuristik, dem rekursiven Greedy-Algorithmus [1]. In diesem Abschnitt beschreiben wir dessen Vorgehensweise.

Ein zulässiges Wort $w = (w_1, ..., w_{2n})$ wird in einer rekursiven Prozedur, die sich selbst aufruft, bis ein vordefiniertes Abbruchkriterium erreicht ist gefärbt. In einem ersten Schritt werden jeweils beide Vorkommen des letzten Buchstabens des Wortes gelöscht. Wir nennen diesen Buchstaben Z und das so entstandene Teilwort w'. Dieses Verfahren wird solange rekursiv wiederholt, bis das Wort w' aus zwei gleichen Buchstaben besteht. Anschließend werden die zuvor gelöschten Buchstaben in der umgekehrten Reihenfolge der Löschung an der ursprünglichen Position im Wort gefärbt. Dabei kann das erste Vorkommen des zu färbenden Buchstaben am Anfang oder Ende der bis zu diesem Zeitpunkt erzeugten Färbung sein oder aber in einer Lücke zwischen den bisher gefärbten Buchstaben. Wir unterscheiden vier verschiedene Lücken $(0,0)$, $(1,1)$, $(0,1)$ und $(1,0)$. Das zweite Vorkommen des zu färbenden Buchstabens befindet sich immer an der letzten Position des jeweiligen Teilwortes. Für die Färbung des ersten Vorkommens von Z gelten die folgenden Regeln. Dem zweiten Vorkommen wird entsprechend die andere Farbe zugeordnet.

- Sei $n = 1$. Die erste Färbung des Wortes w' der Länge zwei ist $(0,1)$.
- Sei $n > 1$.
  - Wenn das erste Vorkommen von Z am Anfang des Wortes w' ist, wird es mit 0 gefärbt.
  - Wenn das erste Vorkommen von Z am Ende des Wortes w' ist, wird es in der Farbe des Vorgängers gefärbt.

    Sei $k \in \mathbb{N}$ die Anzahl der Farbwechsel in der bisher erzeugten Färbung des Wortes w'.

  - Wenn das erste Vorkommen von Z in der Färbung des Wortes w' in einer Lücke $(1,1)$ ist, wird das erste Z mit 1 gefärbt.
  - Wenn das erste Vorkommen von Z in der Färbung des Wortes w' in einer Lücke $(0,0)$ ist, wird das erste Z mit 0 gefärbt.

- Wenn k ungerade und das erste Vorkommen von Z in der Färbung des Wortes w' in einer Lücke $(0,1)$ oder $(1,0)$ ist, wird das erste Z mit 0 gefärbt.
- Wenn k gerade und das erste Vorkommen von Z in der Färbung des Wortes w' in einer Lücke $(0,1)$ oder $(1,0)$ ist, wird das erste Z mit 1 gefärbt.

Wir nennen die Färbung, die durch den rekursiven Greedy-Algorithmus erzeugt wird, rekursive Greedy-Färbung $rg = (rg_1, ..., rg_{2n}) \in \{0,1\}^{2n}$.

Nachfolgend zeigen wir einen Pseudocode des rekursiven Greedy-Algorithmus

Algorithm 3 rekursiver Greedy-Algorithmus

if Länge$(w') = 2$ **then**
  $rg = (0,1)$
else
  Sei Länge$(w') > 2$
  if erstes Z ist am Anfang von w' **then**
    setze $rg_Z = 0$
  if erstes Z ist am Ende von w' **then**
    setze $rg_Z = rg_{Z-1}$
  Sei k die Anzahl der Farbwechsel in $rg(w')$
  if k ist ungerade **then**
    if erstes Z ist in einer Lücke $(1,1)$ von $rg(w')$ **then**
      setze $rg_Z = 1$
    if erstes Z ist in einer Lücke $(0,0)$, $(0,1)$, oder $(1,0)$ von $rg(w')$ **then**
      setze $rg_Z = 0$
  else
    if erstes Z ist in einer Lücke $(0,0)$ von $rg(w')$ **then**
      setze $rg_Z = 0$
    if erstes Z ist in einer Lücke $(1,1)$, $(0,1)$, oder $(1,0)$ von $rg(w')$ **then**
      setze $rg_Z = 1$
  end if

Anhand des Wortes $w = (d, a, a, b, c, b, c, d)$ veranschaulichen wir die Vorgehensweise des rekursiven Greedy-Algorithmus in der folgenden Darstellung. Im oberen Teil der Darstellung sehen wir die Erzeugung des Wortes w' mit der rekursiven Prozedur, die in jedem Durchlauf beide Vorkommen des jeweils letzten Buchstabens löscht. Wenn das Wort w' aus zwei gleichen Buchstaben besteht, beginnen wir die Färbung der Buchstaben in der umgekehrten Reihenfolge der Löschung. Im unteren Teil der Abbildung ist die Färbung der zuletzt gelöschten Buchstaben in der rekursiven Prozedur, nach den Regeln des rekursiven Greedy-Algorithmus, dargestellt.

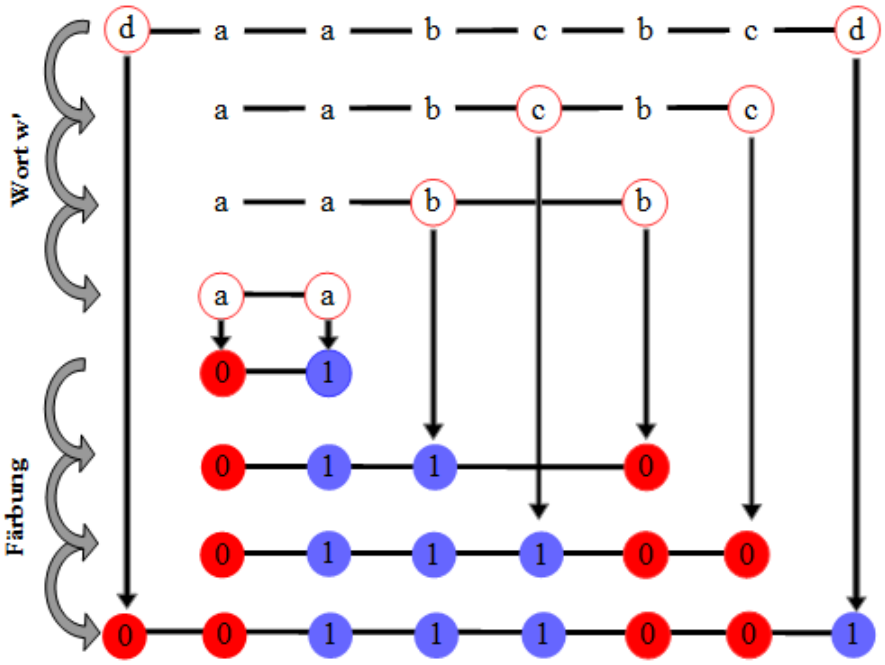

Abbildung 3.6: Hier sehen wir die Färbung eines zulässigen Wortes $w = (d, a, a, b, c, b, c, d)$ mit dem rekursiven Greedy-Algorithmus.

3.3.2 Die Charakteristik des rekursiven Greedy-Algorithmus

In diesem Abschnitt zeigen wir die bereits existierenden Ergebnisse von *S. D. Andres* und *W. Hochstättler* zur Charakteristik des rekursiven Greedy-Algorithmus [1]. Die folgenden Ausführungen orientieren sich an [1].

Der rekursive Greedy-Algorithmus beachtet nicht nur die Farbe des Vorgängers, wie der Greedy-Algorithmus, oder das Vorkommen des Buchstabens, wie der Red-First-Algorithmus. Dadurch kann beim Einfügen des ersten Vorkommens von Z in eine Lücke $(0, 1)$ oder $(1, 0)$ immer ein Farbwechsel vermieden werden. Ein zusätzlicher Farbwechsel kann in einem Schritt des rekursiven Greedy-Algorithmus nur dann auftreten, wenn k ungerade ist und

das erste Z in einer Lücke $(1,1)$ eingefügt wird oder wenn k gerade ist und das erste Z sich in einer Lücke $(0,0)$ der rekursiven Greedy-Färbung befindet.

In der nachfolgenden Tabelle zeigen wir die Erwartungswerte für die Anzahl der Farbwechsel der drei vorgestellten Heuristiken [1]. Wir setzen voraus, dass die Fälle der Worte w der Länge $2n$ des Binary-Paint-Shop-Problems gleich verteilt sind [1].

Tabelle 3.1: Erwartungswerte der Anzahl der Farbwechsel

Heuristik	**Erwartungswert Anzahl Farbwechsel**
Greedy-Algorithmus	$\mathbb{E}_n(g) = \sum_{k=0}^{n-1} \frac{2k^2-1}{4k^2-1}$
Red-First-Algorithmus	$\mathbb{E}_n(rf) = \frac{2n+1}{3}$
rekursiver Greedy-Algorithmus	$\frac{2n}{5} + \frac{8}{15} \leq \mathbb{E}_n(rg) \leq \frac{2n}{5} + \frac{7}{10}$

Beweis. Die Beweise zu den Ergebnissen der Tabelle 3.1 finden wir in [1]. □

Diese Ergebnisse lassen vermuten, dass der rekursive Greedy-Algorithmus um einige Klassen besser ist als die beiden anderen vorgestellten Heuristiken. Die rekursive Greedy-Färbung ist für die beiden verbotenen Wörter $w = (a, b, a, c, c, b)$ und $w = (a, d, d, c, b, c, a, b)$ des Greedy-Algorithmus, sowie für das verbotene Wort $w = (a, a, b, b)$ der Red-First-Heuristik, optimal. Allerdings wird das dritte verbotene Wort $w = (a, d, d, b, c, c, a, b)$ des Greedy-Algorithmus aus Satz 3.8 auch nicht mit dem rekursiven Greedy-Algorithmus optimal gefärbt.
Wir werden zunächst untersuchen für welche Wörter beziehungsweise Teilwörter der rekursive Greedy-Algorithmus keine optimale Färbung liefert und alle zulässigen Wörter der Länge $2n = 8, 10, 12, 14$ und deren rekursive Greedy-Färbung analysieren.

4 Analyse der rekursiven Greedy-Färbungen

In diesem Kapitel zeigen wir die Vorgehensweise und Ergebnisse der Analyse der rekursiven Greedy-Färbungen aller Wörter der Länge $2n$ für $n = 4, 5, 6, 7$ des Binary-Paint-Shop-Problems. Um bewerten zu können, welche Wörter und Teilwörter vom rekursiven Greedy-Algorithmus optimal gefärbt werden, zählen wir die Farbwechsel und untersuchen ob diese Anzahl minimal ist. Hierfür entwickeln wir ein Analyse-Programm, das wir im Kapitel 4.1 beschreiben. Im Kapitel 4.2 gehen wir auf die Auswertung der Ausgabe dieses Programms ein, um abschließend im Kapitel 4.3 die Ergebnisse der Untersuchung vorzustellen.

4.1 Das Programm zur Analyse

Zur Untersuchung der rekursiven Greedy-Färbungen der Wörter des Binary-Paint-Shop-Problems nutzen wir eine Java-Implementierung in der Entwicklungsumgebung Eclipse der Marke Oracle. Mit Hilfe dieses Programms erzeugen wir im ersten Schritt alle möglichen Kombinationen der zulässigen Wörter w der Länge $2n$ für $n = 4, 5, 6, 7$. Anschließend werden diese Wörter mit dem rekursiven Greedy-Algorithmus gefärbt und die Anzahl der Farbwechsel bestimmt. Diese Zahl wird mit einer theoretisch bestimmten unteren Schranke für die Anzahl der Farbwechsel verglichen. Wörter, deren rekursive Greedy-Färbung mehr Farbwechsel benötigt, wie minimal angenommen, werden als potentiell verbotene Wörter eingestuft. Diese werden weiter untersucht, da in der Liste der potentiell verbotenen Wörter auch Wörter enthalten sein können, die bereits bekannte verbotene Teilwörter enthalten oder deren untere Schranke für die Anzahl der Farbwechsel zu gering ist. Letzteres werden wir im Detail im Kapitel 4.1.1 beschreiben. Zunächst stellen wir alle Methoden und deren Funktion vor, um anschließend die Vorgehensweise des gesamten Programms in einer Übersicht darzustellen.

4.1.1 Die Methoden des Analyse-Programms

Die Methode *'kombinationen'*

Im ersten Schritt werden alle möglichen Kombinationen der zulässigen Wörter w des Binary-Paint-Shop-Problems der Länge $2n$ mit $n = 4, 5, 6, 7$ erzeugt. Wir betrachten alle möglichen Kombinationen der n Buchstaben anstelle aller Permutationen, da Vertauschungen vorkommen bei der die Anordnung der Buchstaben gleich bleibt und für die Färbung derselbe Fall dargestellt wird. Ein Beispiel hierfür sind die beiden Wörter $w_1 = (a, b, a, b)$ und $w_2 = (b, a, b, a)$ mit den gleichen rekursiven Greedy-Färbungen $rg(w_1) = rg(w_2) = (0, 0, 1, 1)$. Die Methode arbeitet nach dem folgendem Prinzip: Wir beginnen mit dem ersten Buchstaben-Paar und erhalten ein Wort w der Länge zwei. Um alle zulässigen Wörter der Länge vier zu erzeugen, setzen

wir einen der beiden neu einzufügenden Buchstaben ans Ende dieser Folge. Der zweite Buchstabe dieses Typen durchläuft die Folge von links nach rechts. Wir wiederholen diesen Vorgang für jede erzeugte Buchstabenfolge mit jeweils zwei neuen, ununterscheidbaren Buchstaben bis wir das Wort der gewünschten Länge $2n$ erhalten.

Dieses Prinzip wird mit einer verschachtelten *for*-Schleife realisiert. Es gibt für jedes Buchstabenpaar, das in dem Wort enthalten ist eine Schleife. Beginnend mit der übergeordneten des ersten Paares wird jede weitere Schleife zur Generierung jeder möglichen, eindeutigen Folge mit zwei zusätzlichen Buchstaben der jeweils Nächsthöheren untergeordnet. In einem ersten Schleifendurchlauf werden die Buchstaben am Anfang und Ende des Wortes hinzugefügt. In allen weiteren $(2n-2)$ Schritten wird das erste Vorkommen des neuen Buchstabens solange eine Position weiter nach rechts gerückt bis das zweite Vorkommen am Ende des Wortes erreicht ist. Wir veranschaulichen diese Vorgehensweise in folgender Darstellung:

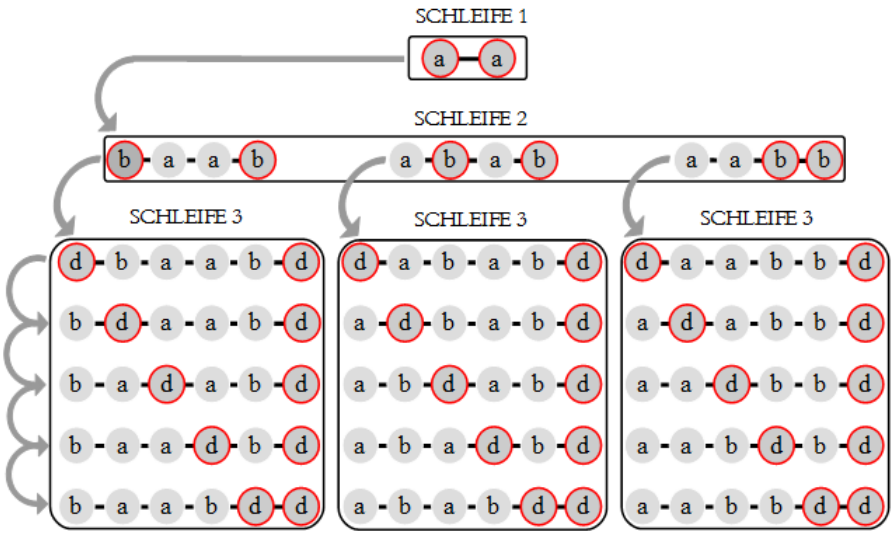

Abbildung 4.7: Beispiel Methode *'kombinationen'*

Für die unterschiedlichen Kombinationen der Buchstaben eines zulässigen Wortes w der Länge $2n$ gibt es

$$\frac{(2n)!}{2^n n!}$$

Möglichkeiten. Diese Anzahl ergibt sich aus folgender Überlegung: Zwei neue Buchstaben, die in einem Schritt eingefügt werden, sind ununterscheidbar. Für den ersten Buchstaben, der neu hinzugefügt wird, gibt es $2n$ mögliche Plätze und daraus folgend gibt es für den zweiten Buchstaben, der die Folge

durchläuft, $(2n-1)$ Möglichkeiten. Insgesamt erhalten wir also $(2n)!$ Möglichkeiten die beiden Buchstaben zu platzieren. Wir reduzieren diese Anzahl um den Faktor $\frac{1}{2^n n!}$. Der Faktor $\frac{1}{2^n}$ ergibt sich, da die beiden Vorkommen eines Buchstabens ununterscheidbar sind und wir keine neuen Wörter erhalten, wenn wir zwei gleiche Buchstaben untereinander tauschen. Da wir jeweils nur eine eindeutige Anordnung einer Permutation für den rekursiven Greedy-Algorithmus betrachten, schließen wir die Anzahl der nicht relevanten Permutationen mit dem Faktor $\frac{1}{n!}$ aus.
Wir erhalten dadurch folgende Anzahl von zu betrachtenden Wörtern der Länge $2n$ mit $n = 4, 5, 6, 7$ für die Analyse.

Tabelle 4.2: Anzahl möglicher Wörter der Länge $2n$

Länge $2n$ des Wortes	**Anzahl der Wörter**
8	105
10	945
12	10395
14	135135

Alle so erzeugten Wörter w der Länge $2n$ für $n = 4, 5, 6, 7$ werden in eine Liste *'kombinationen'* geschrieben.

Die Methode *'w_ strich'*
Um alle zuvor erzeugten Wörter mit dem rekursiven Greedy-Algorithmus zu färben, benötigen wir, wie in Abschnitt 3.3.1 beschrieben, zwei Schritte: die Erzeugung der Teilwörter w' und die anschließende Färbung. Die Methode *'w_ strich'* ruft jedes Wort w der Länge $2n$ aus der Liste *'kombinationen'* auf. Zunächst wird der letzte Buchstabe und dessen erstes Vorkommen im Wort bestimmt. Der Index des ersten Vorkommens wird in einer Liste *'indexliste'* gespeichert. Anschließend werden beide Vorkommen dieses Buchstabens gelöscht und das Wort w' der Länge $(2n-2)$ erzeugt. Mit diesem Wort w' wird die Prozedur erneut aufgerufen und mit dem gleichen Verfahren die beiden Vorkommen des letzten Buchstaben von w' entfernt. Dieses rekursive Vorgehen wird solange wiederholt bis wir ein Teilwort w' der Länge zwei erhalten. Die Indizes der ersten Vorkommen aller gelöschten Buchstaben werden der Reihe nach in einer Liste *'indexliste'* gespeichert. Diese benötigen wir, um anschließend die gefärbten Buchstaben wieder an ihrer ursprünglichen Stelle einzufügen.

Wir veranschaulichen diese Funktion in der folgenden Darstellung.

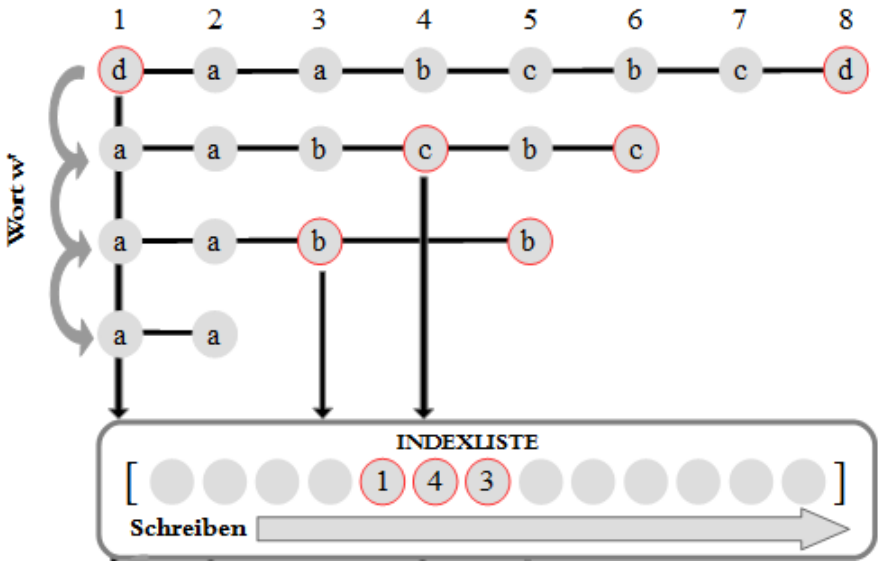

Abbildung 4.8: Beispiel Methode *'w_strich'*

Die *'rekursive Greedy-Färbung'*

Mit Hilfe der zuvor erzeugten *'indexliste'* wird nun jedes Wort w mit dem rekursiven Greedy-Algorithmus nach den Regeln von Abschnitt 3.3.1 gefärbt. Die rekursive Prozedur zur Färbung aller Wörter w ist mit zwei verschachtelten Schleifen realisiert. In der übergeordneten Schleife wird die Position des zu färbenden Wortes in der Liste *'kombinationen'* bestimmt. Wir bezeichnen diese Position mit $j \in \mathbb{N}$. Diese Schleife wird so oft durchlaufen bis alle möglichen Wörter einer Länge $2n$ gefärbt sind. Die erste Färbung $rg = (0, 1)$ jedes Wortes an Stelle j wird an dessen zugehörige Position in einer Liste *'färbung'* gespeichert. In der zweiten Schleife werden nun die Farben der beiden Vorkommen der zuvor gelöschten $(n-1)$ Buchstaben in diese erste Färbung eingefügt. Um die Positionen der zu färbenden Buchstaben im Wort zu bestimmen, werden die zugehörigen Indizes aus der Liste *'indexliste'* ausgelesen. Da wir die Färbung in der umgekehrten Reihenfolge der Löschung vornehmen, müssen wir zuerst den zuletzt gespeicherten Index auslesen. Wir lokalisieren den zu einem Wort zugehörigen Index über die zuvor bestimmte Position j des Wortes, indem wir die Positionszahl j mit der Anzahl der $(n-1)$ einzufügenden Buchstaben multiplizieren. Wir erhalten dadurch für das zu färbende Wort die Stelle des ersten Vorkommens der zweiten Färbung. Mit diesem Index bestimmen wir nun in einer *if-else* Abfrage, welche der Regeln des rekursiven Greedy-Algorithmus angewandt werden muss und damit die Farbe des Buchstabens an dieser Position. Die komplementäre Farbe wird am Ende der Färbung für das zweite Vorkommen angehängt. Mit der ersten Färbung deklarieren wir einen Farbwechselzähler $k \in \mathbb{N}$ mit Wert eins. Tritt nun für eine Färbung einer der Fälle auf, die einen zusätzlichen

Farbwechsel verursachen, wird dieser Zähler um Eins erhöht. Dieser Schleifendurchlauf wird $(n-2)$-mal wiederholt. Am Ende der Schleifendurchläufe erhalten wir eine Liste *'färbungen'* die für jedes Wort die zugehörige rekursive Greedy-Färbung enthält und eine Liste *'farbwechsel_real'* mit der tatsächlichen Anzahl von Farbwechsel. Die folgende Darstellung veranschaulicht die Funktionsweise der Methode.

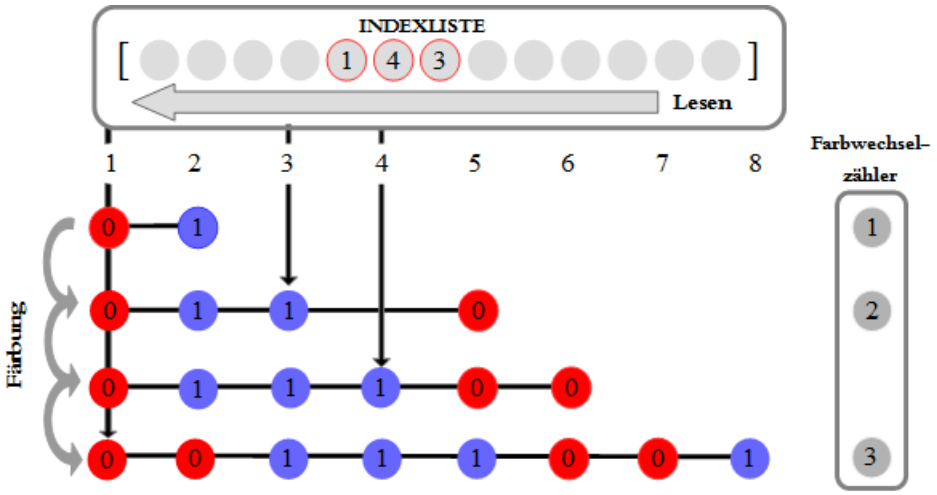

Abbildung 4.9: Beispiel Methode *'rekursive Greedy-Färbung'*

Die Methode *'farbwechsel _ minimal'*

Diese Methode dient zur Ermittlung einer unteren Schranke für die Anzahl der Farbwechsel, um zu überprüfen, ob die rekursive Greedy-Färbung optimal ist. Die Methode arbeitet nach dem folgenden Prinzip. Wir durchlaufen jedes Wort w der Länge $2n$ von links nach rechts und markieren die kürzeste Folge zweier Vorkommen eines Buchstabens innerhalb der nächsten n Zeichen. Wir erhalten Folgen von zwei gleichen Buchstaben der Länge $2, 3, ..., (n-1), n$. Buchstaben, die kein Teil einer Folge sind, werden mit Null markiert. Für jede dieser Folgen, die ein Wort enthält, wird ein Farbwechsel gezählt. Wir definieren die Intervalle $I(i,j)$ und die Menge dieser Intervalle $l(w)$ der Wörter w nach Abschnitt 3.2.2. Seien $i, j \in \mathbb{N}$.

$$I(i,j) = \left\{ i + \tfrac{1}{2}, (i+1) + \tfrac{1}{2}, ..., (j-1) + \tfrac{1}{2} \right\}$$

$$l(w) = \{I(i,j) | 1 \leq i < j \leq 2n, w_i = w_j \text{ mit } i \neq j\}$$

Durch die Bildung dieser Buchstabenfolgen erzeugen wir eine Menge von paarweise disjunkten Intervallen $I(i,j)$ eines Wortes w, die eine Teilmenge von $l(w)$ sind. Die maximale Anzahl von paarweise disjunkten Intervallen

von $l(w)$ ist eine untere Schranke für die Anzahl von Farbwechsel eines zulässigen Wortes w siehe [7]. Wir verbessern diese untere Schranke, indem wir zusätzliche Farbwechsel zählen, die auftreten, wenn zwischen zwei gleichen Buchstaben eine gerade Anzahl von Farbwechsel ist.

Beispiel 4.1. *Wir betrachten das zulässige Wort $w = (a, b, a, e, c, b, c, d, d, e)$ von links nach rechts. Die kürzeste Folge innerhalb der ersten n Buchstaben ist die 3er-Folge (a, b, a). Wir zählen hierfür den ersten Farbwechsel. Weiter betrachten wir die Sequenz nach dem zweiten Vorkommen von a, dann treten weitere Farbwechsel in der 3er-Folge (c, b, c) und im Paar (d, d) auf. Da wir zwei Farbwechsel zwischen den beiden Vorkommen von e zählen, addieren wir einen zusätzlichen Farbwechsel. Eine untere Schranke für die Anzahl der Farbwechsel ist somit vier.*

In unserem Programm wird jedes Wort mit einer Schleife durchlaufen und alle Zeichen einer Folge der Länge $k \in \mathbb{N}$ mit $2 \leq k \leq n$ mit k markiert. Zum Beispiel werden einem Paar (a,a) die Werte (2,2) zugewiesen. Buchstaben, die kein Teil einer Folge sind, werden mit Null markiert. In einer weiteren Schleife werden die zugehörigen Zahlenfolgen für jedes Wort durchlaufen. Eine Folge wird erkannt, wenn die aktuell betrachtete Position eine Zahl $k \neq 0$ enthält. In diesem Fall wird ein Farbwechsel gezählt und die Schleife springt k Stellen weiter nach rechts. Um die Farbwechsel zu ermitteln, die bei einer geraden Anzahl von Farbwechsel zwischen zwei gleichen Buchstaben auftreten, gibt es einen zusätzlichen internen Farbwechselzähler. Sobald dieser Zähler eine gerade Anzahl von Farbwechsel enthält, springt die Prozedur in eine weitere Schleife. Wir betrachten die Sequenz der Folgen, deren Summe von Farbwechsel im internen Zähler gerade ist. Die Schleife überprüft, ob es einen Buchstaben gibt, dessen erstes Vorkommen vor diesen Folgen und dessen zweites Vorkommen nach diesen Folgen auftritt. In einer verschachtelten Schleife wird für jedes mit Null markierte Zeichen vor dieser Sequenz abgefragt, ob das Teilwort nach dieser Sequenz das gleiche Zeichen enthält. Die Schleife durchläuft dieses Teilwort bis zu einem Zeichen ungleich Null oder bis zum Ende des Wortes. Ist letzteres der Fall wird ein Farbwechsel addiert und der interne Zähler zurückgesetzt. Tritt kein zusätzlicher Farbwechsel auf wird der interne Zähler um Eins reduziert. Dadurch wird der erste Farbwechsel für die weitere Auswertung ignoriert. Wir veranschaulichen dieses Prinzip an dem folgenden Beispiel.

Beispiel 4.2. *Betrachten wir das Wort $w = (a, b, a, e, c, b, c, d, d, e)$ aus dem vorherigen Beispiel mit der zugehörigen Zahlenfolge $(3, 3, 3, 0, 3, 3, 3, 2, 2, 0)$. Diese wird mit einer Schleife von links nach rechts durchlaufen. Da es sich beim ersten Zeichen um eine Drei handelt, wird jeweils ein Farbwechsel im Farbwechselzähler und im internen Zähler addiert. Die Schleife springt nach*

dieser 3er-Folge drei Positionen nach rechts auf die nächste Stelle. Da dieses Zeichen gleich Null ist, geht die Schleife ohne Aktion eine Position weiter nach rechts. Das Zeichen dort ist eine Drei. Es wird ein zusätzlicher Farbwechsel addiert und die Routine wird drei Stellen weiter rechts fortgesetzt. Der Wert des internen Farbwechselzählers ist nun zwei und somit gerade. Das Programm überprüft nun, ob vor der ersten und nach der zweiten 3er-Folge zwei gleiche Buchstaben existieren. Dies ist nicht möglich, da die erste 3er-Folge am Anfang des Wortes steht. Es tritt kein zusätzlicher Farbwechsel auf und der interne Zähler wird um Eins reduziert. Wir fahren nun auf der Position nach der zweiten 3er-Folge fort. Dieses Zeichen ist eine Zwei und somit wird ein Farbwechsel zum Farbwechselzähler und dem internen Zähler addiert. Der Wert des Farbwechselzählers ist Drei und der des internen Zählers Zwei. Es wird erneut überprüft, ob es einen zusätzlichen Farbwechsel gibt. Um dies zu überprüfen betrachten wir die mit Null markierten Zeichen ab dem ersten Index vor der zweiten 3er-Folge. In unserem Beispiel ist das die Position des ersten Buchstabens e. Für diesen Buchstaben wird überprüft, ob nach dem letzten Zeichen der zweiten Folge der gleiche Buchstabe auftritt. Da vor und hinter den beiden betrachteten Folgen der Buchstabe e steht, addieren wir einen zusätzlichen Farbwechsel. Die untere Schranke für die Anzahl der Farbwechsel dieses Wortes ist Vier.

Für jedes Wort wird diese ermittelte untere Schranke für die Anzahl der Farbwechsel in eine Liste *'farbwechsel_min'* geschrieben und über die eindeutige Position j dem Wort zugeordnet. Wir beschreiben nun Fälle von Wörtern, deren ermittelte untere Schranke für die Anzahl der Farbwechsel kleiner ist als die tatsächlich minimal benötigte.

Fall 1: In diesem Fall sind mehrere Folgen von Buchstaben so verkettet, dass in der Betrachtung des Wortes von links nach rechts weniger Farbwechsel auftreten, als im Durchlauf des Wortes von rechts nach links. Betrachten wir hierzu folgendes Beispiel.

Beispiel 4.3. *Für das Wort* $w = (b, a, e, c, a, b, c, d, d, e)$ *erhalten wir die Farbwechselmarkierung* $(0, 4, 4, 4, 4, 0, 0, 2, 2, 0)$ *und daraus folgend die untere Schranke für die Anzahl der Farbwechsel von Zwei. Betrachten wir das Wort von rechts nach links, erhalten wir die Zahlenfolge* $(0, 0, 0, 4, 4, 4, 4, 2, 2, 0)$ *und einen zusätzlichen Farbwechsel, da zwei Folgen zwischen den beiden Vorkommen des Buchstabens e auftreten. Die ermittelte untere Schranke für die Farbwechselanzahl ist dadurch Drei und entspricht der tatsächlich minimalen Anzahl benötigter Farbwechsel.*

Wir können dieses Problem umgehen, indem wir Wörter, deren ermittelte untere Schranke für die Anzahl der Farbwechsel kleiner ist als die Anzahl der Farbwechsel in der rekursiven Greedy-Färbung, erneut in umgekehrter Anordnung von rechts nach links mit der Methode *'farbwechsel_minimal'*

durchlaufen. Die dadurch ermittelte unterer Schranke für die Anzahl der Farbwechsel wird in eine Liste *'farbwechsel_ min_ rück'* geschrieben und in der anschließenden Bewertung der Ausgabe des Programms mit der tatsächlichen Farbwechselanzahl verglichen.

Fall 2: Im zweiten Fall werden Farbwechsel nicht erkannt, da diese nicht mit dem Folgen-Markierungs-Prinzip abgebildet werden können. Ein Beispiel hierzu ist Folgendes.

Beispiel 4.4. *Sei das Wort $w = (b, a, a, c, e, b, c, d, d, e)$ gegeben. Die Methode 'farbwechsel _minimal' zählt für dieses Wort je einen Farbwechsel für die 2er-Folge (a, a), die 4er-Folge (c, e, b, c) und die 2er-Folge (d, d). Dasselbe gilt, wenn wir das Wort von rechts nach links betrachten. Wir erhalten drei Farbwechsel und dadurch einen Farbwechsel weniger, als tatsächlich benötigt wird. Dazu betrachten wir die zwei möglichen Fälle. Im ersten Fall gibt es nur einen Farbwechsel bei (a, a) zwischen den beiden Vorkommen von b und damit genau zwei Farbwechsel zwischen den beiden Vorkommen von e. Dies erfordert einen zusätzlichen vierten Wechsel der Farbe. Auch wenn es nur einen Farbwechsel zwischen den beiden Vorkommen von e bei (d, d) gibt, erhalten wir zwei Farbwechsel zwischen den beiden Vorkommen von b und benötigen dadurch einen zusätzlichen vierten Farbwechsel.*

Wörter dieses zweiten Falls müssen in der späteren manuellen Analyse der Ausgabe des Programms betrachtet werden.

Fall 3: Im dritten Fall sind mehrere Folgen von Buchstaben verkettet, sodass nicht alle Farbwechsel erkannt werden können. Betrachten wir hierzu das folgende Beispiel.

Beispiel 4.5. *Für das Wort $w = (a, d, b, a, b, c, e, c, d, e)$ zählen wir je einen Farbwechsel für die 4er-Folge (a, d, b, a) und für die 3er-Folge (c, e, c) von links nach rechts. Durchlaufen wir das Wort von rechts nach links, erhalten wir wieder zwei Farbwechsel. Es wird jeweils ein Farbwechsel für die 4er-Folge (e, c, d, e) und für die 3er-Folge (b, a, b) gezählt. Tatsächlich werden aber drei Farbwechsel benötigt. Das sehen wir, wenn wir anstelle des Farbwechsels der 4er-Folge (a, d, b, a) den Farbwechsel der 3er-Folge (b, a, b) zählen. Mit der zweiten 3er-Folge (c, e, c) gibt zwei Farbwechsel zwischen den beiden Vorkommen von d und wir benötigen einen zusätzlichen dritten Wechsel der Farbe.*

Auch diese Wörter müssen in der anschließenden Analyse der Ausgabe des Programms nachbewertet werden.

Die Methode *'verbotene_ wörter'*
Die Methode *'verbotene_ wörter'* vergleicht die Anzahl der Farbwechsel der rekursiven Greedy-Färbung für jedes Wort mit der zuvor ermittelten unteren Schranke für die Anzahl der Farbwechsel. Dazu durchläuft eine Schleife für jedes Wort die zugehörigen Einträge in den Listen *'farbwechsel_ real'* und *'farbwechsel_ min'*. Ist die Anzahl der Farbwechsel der rekursiven Greedy-Färbung größer als die ermittelte untere Schranke, wird dieses Wort in die Liste *'verbotene_ wörter'* geschrieben.

Die Methode *'bekannte_ verbotene_ wörter'*
Wörter, die nicht optimal durch den rekursiven Greedy-Algorithmus gefärbt werden, sind auch in längeren Wörtern als Teilwort enthalten. Wir sind an Wörtern interessiert, deren rekursive Greedy-Färbung nicht optimal ist und die kein bereits bekanntes verbotenes Teilwort enthalten. Deshalb werden Wörter, die ein solch bekanntes verbotenes Teilwort kürzerer Länge enthalten, aus der Liste *'verbotene_ wörter'* ausgeschlossen. Hierzu werden beide Vorkommen jedes Buchstabens einmal gelöscht und es wird überprüft, ob die Anordnung der verbleibenden $(2n-2)$ Buchstaben einem der bekannten verbotenen Wörter entspricht. Für jede dieser Buchstabenfolgen wird dieses Vorgehen so oft wiederholt, bis die Wortlänge acht erreicht ist. Alle verbotenen Wörter, die kein bereits bekanntes verbotenes Teilwort enthalten, werden in eine Liste *'verbotene_ wörter_ neu'* geschrieben. Wir veranschaulichen diese Vorgehensweise anhand des folgenden Beispiels.

Beispiel 4.6. *Wir untersuchen das Wort $w = (e, b, c, a, a, b, f, c, d, d, e, f)$ der Länge zwölf. Wir erzeugen alle sechs Teilwörter der Länge zehn, indem wir beide Vorkommen jedes Buchstabens einmal aus diesem Wort löschen.*

$w_1 = (e, b, c, b, f, c, d, d, e, f)$, $w_2 = (e, c, a, a, f, c, d, d, e, f)$,
$w_3 = (e, b, a, a, b, f, d, d, e, f)$, $w_4 = (e, b, c, a, a, b, f, c, e, f)$,
$w_5 = (b, c, a, a, b, f, c, d, d, f)$, $w_6 = (e, b, c, a, a, b, c, d, d, e)$

Keines dieser Teilwörter entspricht der Anordnung eines bekannten verbotenen Wortes der Länge zehn. Wir wiederholen den Vorgang für jedes Teilwort und erzeugen auf dieselbe Weise alle 30 Teilwörter der Länge acht. Betrachten wir hierzu das Teilwort w_3 der Länge zehn und erzeugen alle Teilwörter der Länge acht.

$w_{31} = (e, b, b, f, d, d, e, f)$, $w_{32} = (e, a, a, f, d, d, e, f)$, $w_{33} = (e, b, a, a, b, f, e, f)$,
$w_{34} = (b, a, a, b, f, d, d, f)$, $w_{35} = (e, b, a, a, b, d, d, e)$

Das Wort $w = (e, b, c, a, a, b, f, c, d, d, e, f)$ enthält ein bekanntes verbotenes Teilwort, da die Teilwörter w_{31} und w_{32} die gleiche Anordnung, wie das bekannte verbotene Wort (a, d, d, b, c, c, a, b) aufweisen.

4.1.2 Die Funktionsweise des Programms

Die folgende Abbildung stellt die gesamte Funktion und das Zusammenspiel der eben vorgestellten Methoden dar.

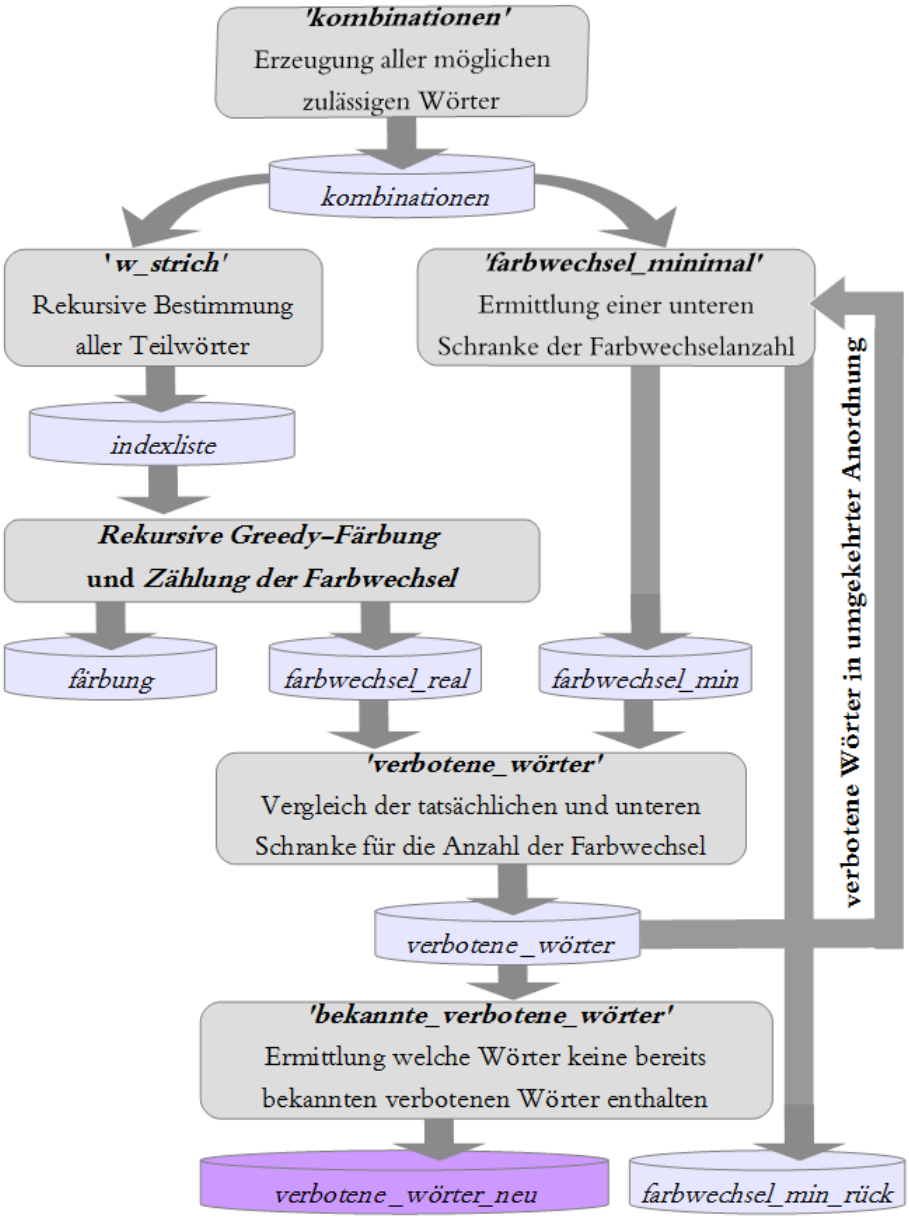

Abbildung 4.10: Darstellung der Funktionsweise des Programms zur Analyse der rekursiven Greedy-Färbungen

4.2 Die Auswertung der Resultate des Analyse-Programms

Um die Ergebnisse der Analyse übersichtlich zu dokumentieren, erfolgt die Auswertung der Ausgabe des Programms in einem Tabellenkalkulations-Programm.

Alle von den Methoden des Programms erzeugten Listen werden im Textformat ausgegeben und zur Analyse in die Spalten der Tabelle importiert, um die einzelnen Resultate für jedes Wort darzustellen. Für die nachgehende Auswertung sind die unbekannten verbotenen Wörter von Interesse. Alle anderen Wörter wurden bereits durch das Programm als *'optimal gefärbt'* oder *'enthält bekanntes verbotenes Wort'* bewertet. Dadurch wird die Anzahl der Wörter, die manuell analysiert werden müssen, signifikant verringert. Wir bewerten alle Wörter der Liste *'verbotene_ wörter_ neu'*, deren untere Schranke für die Anzahl der Farbwechsel auch von rechts nach links kleiner ist als die Farbwechselanzahl der rekursiven Greedy-Färbung. In der Auswertung wird überprüft, ob ein Wort ein neues verbotenes Wort oder eines der Fälle 2 oder 3 aus Abschnitt 4.1.1 ist. In der folgenden Tabelle sehen wir die Anzahl der auszuwertenden Wörter.

Tabelle 4.3: Anzahl der auszuwertenden Wörter

Länge $2n$ des Wortes	**Anzahl der Wörter**	**Anzahl der auszuwertenenden Wörter**	**Anzahl neuer verbotener Wörter**
8	105	2	2
10	945	8	6
12	10395	78	3
14	135135	669	2

4.3 Die Ergebnisse der Auswertung

Durch die Auswertung des Analyse-Programms und die anschließende Untersuchung dieser Resultate, erhalten wir als Ergebnis die folgenden verbotenen Wörter der Länge $2n$ mit $n = 4, 5, 6, 7$, deren rekursive Greedy-Färbung nicht optimal ist. Wir bezeichnen in den folgenden Tabellen die rekursive Greedy-Färbung eines zulässigen Wortes w mit $rg(w)$ und die optimale Fär-

bung mit $f(w)$. Die Anzahl der Farbwechsel benennen wir entsprechend mit K_{rg} und K_f, mit $K_{rg}, K_f \in \mathbb{N}$.

Verbotene Wörter der Länge 8

Tabelle 4.4: Verbotene Wörter der Länge 8 und deren optimale und rekursive Greedy-Färbung

Wort	$rg(w)$	K_{rg}	$f(w)$	K_f
(a,d,d,b,c,c,a,b)	$(0,0,1,1,1,0,1,0)$	4	$(0,1,0,0,0,1,1,1)$	3
(a,b,c,a,d,c,d,b)	$(0,0,0,1,1,1,0,1)$	3	$(0,1,1,1,1,0,0,0)$	2

Verbotene Wörter der Länge 10

Tabelle 4.5: Verbotene Wörter der Länge 10 und deren rekursive Greedy-Färbung

Wort	$rg(w)$	K_{rg}
(d,b,a,a,c,e,b,c,d,e)	$(0,0,0,1,1,1,1,0,1,0)$	4
(b,d,a,a,c,e,b,c,d,e)	$(0,0,0,1,1,1,1,0,1,0)$	4
(a,b,a,c,e,b,c,d,d,e)	$(0,0,1,1,1,1,0,0,1,0)$	4
(a,d,b,a,e,b,c,c,d,e)	$(0,0,0,1,1,1,1,0,1,0)$	4
(d,a,c,e,a,b,b,c,d,e)	$(0,0,1,1,1,1,0,0,1,0)$	4
(a,c,e,a,b,b,d,c,d,e)	$(0,1,1,1,1,0,0,0,1,0)$	4

Tabelle 4.6: Verbotene Wörter der Länge 10 und deren optimale Färbung

Wort	$f(w)$	K_f
(d,b,a,a,c,e,b,c,d,e)	$(0,0,1,0,0,0,1,1,1,1)$	3
(b,d,a,a,c,e,b,c,d,e)	$(0,0,1,0,0,0,1,1,1,1)$	3
(a,b,a,c,e,b,c,d,d,e)	$(0,1,1,1,0,0,0,0,1,1)$	3
(a,d,b,a,e,b,c,c,d,e)	$(0,0,1,1,0,0,0,1,1,1)$	3
(d,a,c,e,a,b,b,c,d,e)	$(0,0,0,0,1,1,0,1,1,1)$	3
(a,c,e,a,b,b,d,c,d,e)	$(0,0,0,1,1,0,0,1,1,1)$	3

Verbotene Wörter der Länge 12

Tabelle 4.7: Verbotene Wörter der Länge 12 und deren rekursive Greedy-Färbung

Wort	$rg(w)$	K_{rg}
$(c,a,a,b,d,f,b,c,e,d,e,f)$	$(0,0,1,1,0,0,0,1,1,1,0,1)$	5
$(c,a,a,b,d,f,b,c,d,e,e,f)$	$(0,0,1,1,0,0,0,1,1,1,0,1)$	5
$(d,a,a,c,b,b,f,c,d,e,e,f)$	$(0,0,1,1,1,0,0,0,1,1,0,1)$	5

Tabelle 4.8: Verbotene Wörter der Länge 12 und deren optimale Färbung

Wort	$f(w)$	K_f
$(c,a,a,b,d,f,b,c,e,d,e,f)$	$(0,1,0,0,0,1,1,1,1,1,0,0)$	4
$(c,a,a,b,d,f,b,c,d,e,e,f)$	$(0,1,0,0,0,1,1,1,1,1,0,0)$	4
$(d,a,a,c,b,b,f,c,d,e,e,f)$	$(0,1,0,0,0,1,1,1,1,1,0,0)$	4

Verbotene Wörter der Länge 14

Tabelle 4.9: Verbotene Wörter der Länge 14 und deren rekursive Greedy-Färbung

Wort	$rg(w)$	K_{rg}
$(b,a,a,c,f,b,e,g,c,d,d,e,f,g)$	$(0,0,1,1,1,1,0,0,0,0,1,1,0,1)$	5
$(f,a,a,e,b,b,d,c,c,g,d,e,f,g)$	$(0,0,1,1,1,0,0,0,1,1,1,0,1,0)$	6

Tabelle 4.10: Verbotene Wörter der Länge 14 und deren optimale Färbung

Wort	$f(w)$	K_f
$(b,a,a,c,f,b,e,g,c,d,d,e,f,g)$	$(0,1,0,0,1,1,1,1,1,1,0,0,0,0)$	4
$(f,a,a,e,b,b,d,c,c,g,d,e,f,g)$	$(0,1,0,0,0,1,1,1,0,0,0,1,1,1)$	5

5 Untersuchung des rekursiven Greedy-Algorithmus

Die Ergebnisse aus Kapitel 4 haben gezeigt, dass es 13 verbotene Wörter bis zur Wortlänge 14 des rekursiven Greedy-Algorithmus gibt. Die ursprüngliche Vermutung lautete, dass eine Charakterisierung des rekursiven Greedy-Algorithmus, basierend auf der Idee der Charakterisierung des Greedy-Algorithmus, möglich ist. Mit anderen Worten vermuteten wir, dass man die Beweisstruktur von Satz 3.8 auf den rekursiven Greedy-Algorithmus anwenden kann. Zunächst analysieren wir verschiedene Definitionen von Hypergraphen bezüglich der rekursiven Greedy-Färbung. Wir untersuchen, ob diese Hypergraphen für Wörter, die vom rekursiven Greedy-Algorithmus in allen Teilwörtern optimal gefärbt werden, geradzahlig laminar sind und die Optimalität der Färbung beweisen. Nach einigen negativen Ergebnissen können wir diese Vermutung abschließend widerlegen, da wir ein optimal gefärbtes Wort finden konnten, für das es nicht möglich ist, einen Hypergraphen bezüglich der rekursiven Greedy-Färbung zu definieren, der für dieses Wort geradzahlig laminar ist. Dieses Wort werden wir in Abschnitt 5.2 vorstellen und untersuchen, ob es weitere Wörter dieser Form gibt.

Zunächst werden wir die Versuche und negativen Ergebnisse in Abschnitt 5.1 dokumentieren. Eine Dokumentation ist nötig, da diese Resultate für die weitere Arbeit an der Charakterisierung des rekursiven Greedy-Algorithmus wichtig sein können.

Zur einfacheren Dokumentation formulieren wir den untersuchten Teil der Beweisstruktur der Charakterisierung des Greedy-Algorithmus als Hypothese für den rekursiven Greedy-Algorithmus.

Hypothese 5 *Für jedes zulässige Wort w des Binary-Paint-Shop-Problems sind die beiden folgenden Aussagen äquivalent.*

(i) Die rekursive Greedy-Färbung ist für jedes zulässige Teilwort w' von w optimal.

(ii) $\mathcal{H}(w')$ ist ein geradzahlig laminarer Hypergraph für jedes zulässige Teilwort w' von w.

5.1 Die Untersuchung verschiedener Hypergraphen

Analog zum Vorgehen in Abschnitt 3.1.2 verwenden wir die Intervalle $I(i,j)$ mit $i,j \in \mathbb{N}$ und $0 < i < i+1 < \cdots < j-1 < j \leq 2n$, wobei i und j die beiden Vorkommen eines Buchstabens in einem zulässigen Wort $w = (w_1, ..., w_{2n})$ sind. Die Intervalle $I(i,j)$ und die Menge dieser Intervalle $l(w)$ definieren wir nach Abschnitt 3.1.2.

$I(i,j) = \left\{ i + \frac{1}{2}, (i+1) + \frac{1}{2}, ..., (j-1) + \frac{1}{2} \right\}$

$l(w) = \{I(i,j) | 1 \leq i < j \leq 2n, w_i = w_j\}$

Wir untersuchen im Folgenden verschiedene Hypergraphen, die wir mit $\mathcal{H}_x$ bezeichnen und mit $1 \leq x \leq 7$ nummerieren. Wir beginnen mit der Untersuchung des Hypergraphen $\mathcal{H}_1$.

Untersuchung des Hypergraphen $\mathcal{H}_1$

Der Hypergraph $\mathcal{H}_1$ entspricht dem Hypergraphen $\mathcal{G}$ der Charakterisierung des Greedy-Algorithmus im Kapitel 3.1.2. Die Hyperkanten des Hypergraphen $\mathcal{H}_1(w)$ für ein zulässiges Wort w des Binary-Paint-Shop-Problems sind die Intervalle $I(i,j)$ der Buchstaben, vor deren zweiten Vorkommen ein Farbwechsel auftritt.

Definition 5.1. *(Hypergraph $\mathcal{H}_1$) Sei $\mathcal{H}_1$ ein Hypergraph. Die Hyperkanten von $\mathcal{H}_1(w)$ sind die Intervalle $I(i,j)$ der Buchstaben eines zulässigen Wortes w, vor deren zweiten Vorkommen an der Stelle $((j-1)+\frac{1}{2})$ ein Farbwechsel auftritt. Es ist*

$\mathcal{H}_1(w) = \{I(i,j) \in l(w) | rg_{j-1} \neq rg_j\}$.

Im Folgenden betrachten wir ein Beispiel für ein zulässiges Wort w, das vom rekursiven Greedy-Algorithmus in allen Teilwörtern w' optimal gefärbt wird und $\mathcal{H}_1(w)$ geradzahlig laminar ist.

Beispiel 5.2. *Sei das zulässige Wort $w = (c,a,a,b,b,d,c,d)$ mit der rekursiven Greedy-Färbung $rg = (0,0,1,1,0,0,1,1)$ gegeben. Die Intervalle dieses Wortes sind $I(1,7), I(2,3), I(4,5)$ und $I(6,8)$. Die Hyperkanten von $\mathcal{H}_1(w)$ sind die Intervalle $I(1,7), I(2,3), I(4,5)$, da hier an der Stelle $((j-1)+\frac{1}{2})$ ein Farbwechsel auftritt. Damit werden alle Farbwechsel durch eine Hyperkante abgebildet. Da die Intervalle $I(2,3), I(4,5)$ disjunkt sind und das Intervall $I(1,7)$ die Intervalle $I(2,3), I(4,5)$ enthält, sind die Bedingungen aus Definition 3.5 und 3.6 gegeben. Es folgt, dass $\mathcal{H}_1(w)$ geradzahlig laminar ist.*

Die weitere Analyse dieses Hypergraphen zeigt, dass es zulässige Wörter gibt, deren rekursive Greedy-Färbung für jedes Teilwort w' von w optimal ist, aber wir dies nicht mit dem Hypergraphen $\mathcal{H}_1$ beweisen können.

Betrachten wir das Wort $w = (c,a,a,b,d,b,c,d)$ mit der rekursiven Greedy-Färbung $rg = (0,0,1,1,0,0,1,1)$ mit drei Farbwechsel. Es gilt die folgende Behauptung.

Behauptung 5. *Die rekursive Greedy-Färbung $rg = (0,0,1,1,0,0,1,1)$ des Wortes $w = (c,a,a,b,d,b,c,d)$ ist optimal.*

Beweis. Sei das zulässige Wort $w = (c, a, a, b, d, b, c, d)$ gegeben. Es tritt ein Farbwechsel zwischen den beiden Vorkommen des Buchstabens a auf. Es tritt mindestens ein weiterer Wechsel der Farbe in der Folge (b, d, b) an der Stelle vor dem ersten Vorkommen von d oder vor dem zweiten Vorkommen von b auf. Daraus ergeben sich mindestens zwei Farbwechsel zwischen den beiden Vorkommen von c. Ist die Anzahl genau zwei und somit gerade, tritt mit Proposition 3.7 ein zusätzlicher Farbwechsel vor dem zweiten Vorkommen von c auf. Im anderen Fall gibt es drei Farbwechsel zwischen c. In beiden Fällen ist die untere Schranke für die Anzahl der Farbwechsel drei. Es folgt die Behauptung. □

Wir können mit dem Hypergraphen $\mathcal{H}_1(w)$ nicht beweisen, dass die rekursive Greedy-Färbung für das Wort $w = (c, a, a, b, d, b, c, d)$ optimal ist.

Behauptung 6. *Sei $w = (c, a, a, b, d, b, c, d)$, dann ist die Anzahl der Hyperkanten in $\mathcal{H}_1(w)$ geringer als die untere Schranke für die Anzahl der Farbwechsel der rekursiven Greedy-Färbung des Wortes w.*

Beweis. Die Intervalle $I(i, j)$ des Wortes w sind $I(1, 7), I(2, 3), I(4, 6)$ und $I(5, 8)$. Die Hyperkanten in $\mathcal{H}_1(w)$ sind die Intervalle $I(2, 3)$ und $I(4, 6)$. Der dritte notwendige Farbwechsel kann nicht abgebildet werden. □

Es existiert ein Hypergraph, bezüglich der rekursiven Greedy-Färbung, der die Optimalität der rekursiven Greedy-Färbung dieses Wortes beweist. Diesen Hypergraphen werden wir detailliert im Abschnitt der Untersuchung des Hypergraphen $\mathcal{H}_3$ beschreiben.

Untersuchung des Hypergraphen $\mathcal{H}_2$

Im nächsten Schritt definieren wir den Hypergraphen $\mathcal{H}_2$. Die Hyperkanten von $\mathcal{H}_2$ sind die Intervalle $I(i, j)$ der Buchstaben eines zulässigen Wortes, nach deren ersten Vorkommen ein Farbwechsel auftritt. Wir definieren den Hypergraphen $\mathcal{H}_2$ folgendermaßen.

Definition 5.3. *(Hypergraph $\mathcal{H}_2$) Sei $\mathcal{H}_2$ ein Hypergraph. Die Hyperkanten von $\mathcal{H}_2(w)$ sind die Intervalle $I(i, j)$ der Buchstaben eines zulässigen Wortes w, nach deren ersten Vorkommen an der Stelle $(i + \frac{1}{2})$ ein Farbwechsel auftritt. Es ist*

$$\mathcal{H}_2(w) = \{I(i, j) \in l(w) | rg_i \neq rg_{i+1}\}.$$

Im folgenden Beispiel zeigen wir, dass $\mathcal{H}_2(w)$ für ein zulässiges Wort w, das vom rekursiven Greedy-Algorithmus in allen Teilwörtern w' von w optimal gefärbt wird, geradzahlig laminar ist und genau soviele Hyperkanten enhält, wie es Farbwechsel in der rekursiven Greedy-Färbung gibt.

Beispiel 5.4. *Sei das zulässige Wort* $w = (a, a, b, d, b, c, c, d)$ *mit der rekursiven Greedy-Färbung* $rg = (0, 1, 1, 0, 0, 0, 1, 1)$ *gegeben. Die Intervalle dieses Wortes sind* $I(1, 2), I(3, 5), I(4, 8)$ *und* $I(6, 7)$*. Die Hyperkanten von* $\mathcal{H}_2(w)$ *sind die Intervalle* $I(1, 2), I(3, 5)$ *und* $I(6, 7)$*, da in diesen Intervallen jeweils an der Stelle* $(i + \frac{1}{2})$ *ein Farbwechsel auftritt. Die Intervalle* $I(1, 2), I(3, 5)$ *und* $I(6, 7)$ *sind paarweise disjunkt, dadurch werden die Bedingungen aus Definition 3.5 und 3.6 erfüllt und* $\mathcal{H}_2(w)$ *ist somit geradzahlig laminar.*

In der weiteren Analyse dieses Hypergraphen stellt sich heraus, dass es Wörter gibt, deren rekursive Greedy-Färbung in allen Teilwörtern optimal ist, aber wir dies mit dem Hypergraphen $\mathcal{H}_2$ nicht beweisen können. Ein Beispiel hierfür ist das Wort $w = (a, a, b, c, b, d, c, d)$ mit der rekursiven Greedy-Färbung $rg = (0, 1, 1, 1, 0, 0, 0, 1)$. Wir zeigen zunächst folgende Behauptung.

Behauptung 7. *Die rekursive Greedy-Färbung* $rg = (0, 1, 1, 1, 0, 0, 0, 1)$ *des Wortes* $w = (a, a, b, c, b, d, c, d)$ *ist optimal.*

Beweis. Sei das zulässige Wort $w = (a, a, b, c, b, d, c, d)$ gegeben. Es tritt ein Farbwechsel zwischen den beiden Vorkommen des Buchstabens a auf. Mindestens ein weiterer Farbwechsel tritt jeweils in den Folgen (b, c, b) und (d, c, d) auf. Damit ist die untere Schranke für die Anzahl der Farbwechsel drei und es folgt die Behauptung. □

Wir können mit dem Hypergraphen $\mathcal{H}_2(w)$ nicht beweisen, dass die rekursive Greedy-Färbung für das Wort $w = (a, a, b, c, b, d, c, d)$ optimal ist.

Behauptung 8. *Sei* $w = (a, a, b, c, b, d, c, d)$*, dann ist die Anzahl der Hyperkanten in* $\mathcal{H}_2(w)$ *geringer als die untere Schranke für die Anzahl der Farbwechsel der rekursiven Greedy-Färbung des Wortes.*

Beweis. Die Intervalle $I(i, j)$ des Wortes w sind $I(1, 2), I(3, 5), I(4, 7)$ und $I(6, 8)$. Die Hyperkanten in $\mathcal{H}_2(w)$ sind die Intervalle $I(1, 2)$ und $I(4, 7)$. Mit Behauptung 7 sind mindestens drei Farbwechsel nötig. Der dritte notwendige Farbwechsel kann nicht als Hyperkante in $\mathcal{H}_2(w)$ abgebildet werden. □

Es existiert ein Hypergraph, bezüglich der rekursiven Greedy-Färbung, der die Optimalität der rekursiven Greedy-Färbung dieses Wortes beweist. Diesen Hypergraphen werden wir detailliert im nächsten Abschnitt beschreiben.

Untersuchung des Hypergraphen $\mathcal{H}_3$

Die Wörter aus Behauptung 6 und 8, für die wir die Optimalität der rekursiven Greedy-Färbung mit Hilfe der Hypergraphen $\mathcal{H}_1$ und $\mathcal{H}_2$ nicht beweisen können, führen zur Definition des Hypergraphen $\mathcal{H}_3$. Die Hyperkanten des Hypergraphen $\mathcal{H}_3$ sind die Intervalle $I(i, j)$ der Buchstaben eines zulässigen

Wortes, nach deren ersten Vorkommen oder vor deren zweiten Vorkommen ein Farbwechsel auftritt. Befindet sich ein Farbwechsel nach dem ersten Vorkommen eines Buchstabens und vor dem zweiten Vorkommen eines anderen Buchstabens, ist das Intervall $I(i,j)$ des Buchstabens eine Hyperkante in $\mathcal{H}_3$, nach dessen ersten Vorkommen der Farbwechsel auftritt.

Definition 5.5. *(Hypergraph $\mathcal{H}_3$) Sei $\mathcal{H}_3$ ein Hypergraph. Die Hyperkanten von $\mathcal{H}_3(w)$ sind die Intervalle $I(i,j)$ der Buchstaben eines zulässigen Wortes w, nach deren ersten Vorkommen an der Stelle $(i+\frac{1}{2})$ oder vor deren zweiten Vorkommen an der Stelle $((j-1)+\frac{1}{2})$ ein Farbwechsel auftritt. Der Farbwechsel an der Stelle vor dem zweiten Vorkommen wir nur berücksichtigt, wenn nach dem ersten Vorkommen kein Farbwechsel auftritt. Es ist*

$\mathcal{H}_3(w) = \{I(i,j) \in l(w) | rg_i \neq rg_{i+1}$ *oder* $rg_{j-1} \neq rg_j$, *wenn gilt* $rg_i = rg_{i+1}\}$.

Mit dem Hypergraphen $\mathcal{H}_3$ können wir die Optimalität der rekursiven Greedy-Färbungen der Wörter aus den Gegenbeispielen der Hypergraphen $\mathcal{H}_1$ und $\mathcal{H}_2$ beweisen. Betrachten wir hierzu im Folgenden den Hypergraphen $\mathcal{H}_3(w)$ für das Wort $w = (c,a,a,b,d,b,c,d)$ aus Behauptung 6.

Beispiel 5.6. *Sei das zulässige Wort $w = (c,a,a,b,d,b,c,d)$ mit der rekursiven Greedy-Färbung $rg = (0,0,1,1,0,0,1,1)$ gegeben. Die Intervalle dieses Wortes sind $I(1,7), I(2,3), I(4,6)$ und $I(5,8)$. Die Hyperkanten von $\mathcal{H}_3$ sind die Intervalle $I(1,7), I(2,3)$ und $I(4,6)$, da in diesen Intervallen jeweils an der Stelle $(i+\frac{1}{2})$ oder $((j-1)+\frac{1}{2})$ ein Farbwechsel auftritt. Die Intervalle $I(2,3), I(4,6)$ sind disjunkt und $I(1,7)$ enthält die beiden Intervalle $I(2,3), I(4,6)$. Dadurch sind die Bedingungen aus Definition 3.5 und 3.6 erfüllt und $\mathcal{H}_3(w)$ ist somit geradzahlig laminar.*

In der weiteren Analyse dieses Hypergraphen stellt sich heraus, dass einige Wörter Hypothese 5 widerlegen. Das Wort $w = (c,a,d,a,b,b,c,d)$ mit der rekursiven Greedy-Färbung $rg = (0,0,0,1,1,0,1,1)$ ist hierfür ein Beispiel. Wir zeigen zunächst folgende Behauptung.

Behauptung 9. *Die rekursive Greedy-Färbung $rg = (0,0,0,1,1,0,1,1)$ des Wortes $w = (c,a,d,a,b,b,c,d)$ ist optimal.*

Beweis. Sei das Wort $w = (c,a,d,a,b,b,c,d)$ gegeben. Es tritt jeweils mindestens ein Farbwechsel in der Buchstabenfolge (a,d,a) und (b,b) auf. Damit gibt es mindestens zwei Farbwechsel zwischen den beiden Vorkommen von c. Ist die Anzahl genau zwei und somit gerade, tritt mit Proposition 3.7 ein zusätzlicher Farbwechsel vor dem zweiten Vorkommen von c auf. Im anderen Fall gibt es mindestens drei Farbwechsel zwischen den beiden Vorkommen des Buchstabens c. In beiden Fällen ist die untere Schranke für die Anzahl der Farbwechsel drei. Es folgt die Behauptung. □

Im Folgenden zeigen wir, dass $\mathcal{H}_3(w)$ für $w = (c, a, d, a, b, b, c, d)$ nicht geradzahlig laminar ist und damit Hypothese 5 widerlegt. Dadurch können wir die Optimalität der rekursiven Greedy-Färbung dieses Wortes mit $\mathcal{H}_3(w)$ nicht beweisen.

Behauptung 10. *Sei $w = (c, a, d, a, b, b, c, d)$, dann ist $\mathcal{H}_3(w)$ nicht geradzahlig laminar.*

Beweis. Die Intervalle $I(i, j)$ des Wortes w sind $I(1, 7), I(2, 4), I(3, 8)$ und $I(5, 6)$. Die Hyperkanten in $\mathcal{H}_3(w)$ sind die Intervalle $I(3, 8)$, $I(5, 6)$ und $I(1, 7)$. Die Anzahl der Hyperkanten in $\mathcal{H}_3(w)$ entspricht damit der Anzahl der Farbwechsel der rekursiven Greedy-Färbung des Wortes w. Da die beiden Intervalle $I(3, 8)$ und $I(1, 7)$ nicht disjunkt sind folgt mit Definition 3.5 und 3.6, dass $\mathcal{H}_3(w)$ nicht geradzahlig laminar ist. □

Die folgende Darstellung veranschaulicht dieses Gegenbeispiel.

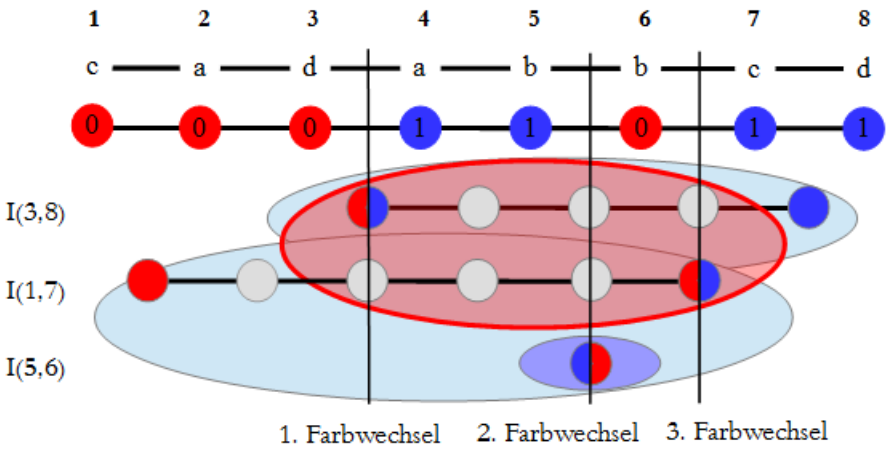

Abbildung 5.11: Die Darstellung zeigt das zulässige Wort w=(c,a,d,a,b,b,c,d) mit der rekursiven Greedy-Färbung rg=(0,0,0,1,1,0,1,1) und dessen Hyperkanten im Hypergraph $\mathcal{H}_3(w)$. Die Schnittmenge der beiden nicht disjunkten Intervalle $I(3, 8)$ und $I(1, 7)$ ist durch einen roten Kreis markiert.

Es existiert ein Hypergraph, bezüglich der rekursiven Greedy-Färbung, der die Optimalität der rekursiven Greedy-Färbung dieses Wortes beweist. Diesen Hypergraphen werden wir detailliert im nächsten Abschnitt beschreiben.

Untersuchung des Hypergraphen $\mathcal{H}_4$

Die bisherigen Versuche einen Hypergraphen durch das Auftreten eines Farbwechsels innerhalb eines Intervalls von Buchstaben zu definieren sind fehl-

geschlagen. Deshalb verfolgen wir nun in einem neuen Ansatz einen Hypergraphen $\mathcal{H}_4$ über die rekursive Prozedur zu konstruieren. Wir definieren den Hypergraphen $\mathcal{H}_4$, dessen Hyperkanten die Intervalle $I(i,j)$ der beiden Vorkommen eines Buchstabens sind, durch deren Färbung in der Prozedur des rekursiven Greedy-Algorithmus ein Farbwechsel verursacht wird.

Definition 5.7. *(Hypergraph $\mathcal{H}_4$) Sei $\mathcal{H}_4$ ein Hypergraph. Die Hyperkanten in $\mathcal{H}_4$ sind die Intervalle $I(i,j)$ der Buchstaben eines zulässigen Wortes w, durch deren Färbung mit dem rekursiven Greedy-Algorithmus ein Farbwechsel verursacht wird. Es ist*

$$\mathcal{H}_4(w) = \{I(i,j) \in l(w) |\ \textit{Färbung von } w_i = w_j \textit{ verursacht einen Farbwechsel}\}.$$

Wir veranschaulichen am Beispiel des Wortes $w = (c,a,d,a,b,b,c,d)$ aus dem vorherigen Abschnitt die Definition des Hypergraphen $\mathcal{H}_4$.

Beispiel 5.8. *Sei das zulässige Wort $w = (c,a,d,a,b,b,c,d)$ gegeben. Es enthält die Intervalle $I(1,7), I(2,4), I(3,8)$ und $I(5,6)$. Wir färben nun das Wort mit dem rekursiven Greedy-Algorithmus und überprüfen, welche Intervalle Hyperkanten in $\mathcal{H}_4$ sind. Wir löschen zuerst beide Vorkommen des jeweils letzten Buchstabens des Wortes w' und erhalten zuletzt das Paar (a,a). Dieses Paar färben wir nach den Regeln des rekursiven Greedy-Algorithmus mit $(0,1)$. Es tritt ein Farbwechsel auf und das Intervall von (a,a) bildet eine Hyperkante in $\mathcal{H}_4$, es folgt $I(2,4) \in \mathcal{H}_4$. Im nächsten Schritt werden die beiden Vorkommen des Buchstabens b gefärbt und wir erhalten die Färbung $(0,1,1,0)$. Das Intervall $I(5,6)$ ist eine weitere Hyperkante in $\mathcal{H}_4$, da durch die Färbung der beiden Vorkommen des Buchstabens b ein Farbwechsel verursacht wird. Im dritten Schritt werden die beiden Vorkommen des Buchstabens c gefärbt. Da das erste Vorkommen des Buchstabens c am Anfang des Wortes auftritt erhalten wir die Färbung $(0,0,1,1,0,1)$ und somit einen dritten Farbwechsel. Es gilt $I(1,7) \in \mathcal{H}_4$. Durch die Färbung der beiden Vorkommen des Buchstabens d erhalten wir die endgültige rekursive Greedy-Färbung des Wortes $rg = (0,0,0,1,1,0,1,1)$. Da die letzte Färbung keinen weiteren Farbwechsel verursacht, ist das Intervall $I(3,8)$ keine Hyperkante in $\mathcal{H}_4$. Da die Intervalle $I(2,4)$ und $I(5,6)$ disjunkt sind und $I(1,7)$ die beiden anderen Intervalle enthält, ist der Hypergraph $\mathcal{H}_4(w)$ geradzahlig laminar.*

In einer weiteren Analyse dieses Hypergraphen finden wir Wörter, die Hypothese 5 widerlegen. Das Wort $w = (a,e,d,b,a,b,c,c,d,e)$ mit der rekursiven Greedy-Färbung $rg = (0,0,0,0,1,1,1,0,1,1)$ ist hierfür ein Beispiel. Wir zeigen zunächst folgende Behauptung.

Behauptung 11. *Die rekursive Greedy-Färbung $rg = (0,0,0,0,1,1,1,0,1,1)$ des Wortes $w = (a,e,d,b,a,b,c,c,d,e)$ ist optimal.*

Beweis. Sei das Wort $w = (a,e,d,b,a,b,c,c,d,e)$ gegeben. Es tritt jeweils mindestens ein Farbwechsel in der Buchstabenfolge (b,a,b) und (c,c) auf.

Daraus folgend gibt es mindestens zwei Farbwechsel zwischen den beiden Vorkommen des Buchstabens d. Ist die Anzahl genau zwei und somit gerade, tritt mit Proposition 3.7 ein zusätzlicher Farbwechsel vor dem zweiten Vorkommen des Buchstabens d auf. Im anderen Fall gibt es mindestens drei Farbwechsel zwischen den Vorkommen des Buchstabens d. In beiden Fällen ist die untere Schranke für die Anzahl der Farbwechsel drei. Es folgt die Behauptung. □

Im Folgenden zeigen wir, dass $\mathcal{H}_4(w)$ für das Wort $w = (a, e, d, b, a, b, c, c, d, e)$ nicht geradzahlig laminar ist und damit Hypothese 5 widerlegt ist.

Behauptung 12. *Sei $w = (a, e, d, b, a, b, c, c, d, e)$, dann ist $\mathcal{H}_4(w)$ nicht geradzahlig laminar.*

Beweis. Die Intervalle $I(i, j)$ des Wortes w sind $I(1, 5), I(2, 10), I(3, 9), I(4, 6)$ und $I(7, 8)$. Durch die Färbung der beiden Vorkommen der Buchstaben a, c und d tritt jeweils ein Farbwechsel auf. Damit sind die Intervalle $I(1, 5)$, $I(7, 8)$ und $I(3, 9)$ Hyperkanten in $\mathcal{H}_4(w)$. Die Anzahl der Hyperkanten in $\mathcal{H}_4(w)$ entspricht somit der Anzahl der Farbwechsel in der rekursiven Greedy-Färbung des Wortes. Da die beiden Intervalle $I(1, 5)$ und $I(3, 9)$ nicht disjunkt sind, folgt mit Definition 3.5 und 3.6, dass $\mathcal{H}_4(w)$ nicht geradzahlig laminar ist. □

Dadurch können wir die Optimalität der rekursiven Greedy-Färbung dieses Wortes mit dem Hypergraphen $\mathcal{H}_4(w)$ nicht beweisen. Die folgende Darstellung veranschaulicht das Gegenbeispiel.

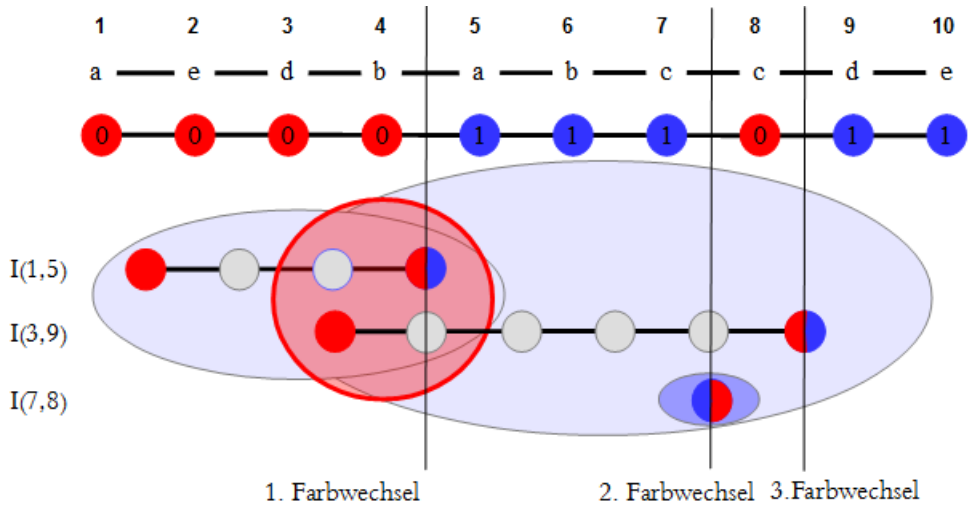

Abbildung 5.12: Die Darstellung zeigt das zulässige Wort w=(a,e,d,b,a,b,c,c,d,e) mit der rekursiven Greedy-Färbung rg=(0,0,0,0,1,1,1,0,1,1) und dessen Hyperkanten im Hypergraph $\mathcal{H}_4(w)$. Die Schnittmenge der beiden nicht disjunkten Intervalle $I(1, 5)$ und $I(3, 9)$ ist durch einen roten Kreis markiert.

Es existiert ein Hypergraph, bezüglich der rekursiven Greedy-Färbung, der die Optimalität der rekursiven Greedy-Färbung dieses Wortes beweist. Diesen Hypergraphen werden wir detailliert im nächsten Abschnitt beschreiben.

Untersuchung des Hypergraphen $\mathcal{H}_5$

Wir betrachten das Gegenbeispiel aus der Untersuchung des Hypergraphen $\mathcal{H}_4$. Ein Hypergraph ist für das Wort $w = (a, e, d, b, a, b, c, c, d, e)$ aus Behauptung 12 geradzahlig laminar, wenn die Intervalle der Buchstaben b, c und d die Hyperkanten bilden. Es stellt sich die Frage, wie wir einen Hypergraphen $\mathcal{H}_5$ bezüglich einer guten rekursiven Greedy-Färbung konstruieren können, sodass genau diese Intervalle als Hyperkanten enthalten sind. Diese Frage führt uns zu dem neuen Ansatz eine Sequenz der Färbung zu betrachten und Regeln zu definieren, die festlegen welches Intervall innerhalb dieser Sequenz eine Hyperkante im Hypergraph $\mathcal{H}_5$ ist. Wir definieren den Hypergraphen $\mathcal{H}_5$, dessen Hyperkanten innerhalb einer betrachteten Sequenz bestimmten Regeln genügen. Zunächst betrachten wir die Sequenz der Färbung bis zum zweiten Farbwechsel. Eine Hyperkante in $\mathcal{H}_5$ ist das Intervall des Buchstabens, dessen beide Vorkommen innerhalb dieser Sequenz auftreten und dessen zweites Vorkommen in dieser Sequenz am weitesten rechts steht. Wir wiederholen diesen Vorgang für alle Farbwechsel und betrachten jeweils die Sequenz bis zum nächsten Farbwechsel.

Definition 5.9. *(Hypergraph $\mathcal{H}_5$) Sei $k \in \mathbb{N}$ die Anzahl der Farbwechsel einer rekursiven Greedy-Färbung rg. Sei $rg_q = (rg_0, ..., rg_l)$ mit $q, l \in \mathbb{N}$ und $1 \leq q \leq k$ eine Sequenz der Färbung, die genau q Farbwechsel enthält. Sei $\mathcal{H}_5$ ein Hypergraph. Die Hyperkanten in $\mathcal{H}_5$ sind die Intervalle $I(i, j)$ zweier Vorkommen eines Buchstabens in w, die in einer Sequenz rg_q enthalten sind und für die $j = max\{j | I(i, j)$ ist in $rg_q\}$ für alle $1 \leq q \leq k$ gilt. Es ist*

$\mathcal{H}_5(w) = \{I(i, j) \in l(w) | I(i, j) \in rg_q$ *und* $j = max\{j | I(i, j)$ *ist in* $rg_q\}$*, für alle* $1 \leq q \leq k\}$.

Mit dem Hypergraphen $\mathcal{H}_5$ zeigen wir im folgenden Beispiel die Optimalität der rekursiven Greedy-Färbung des Wortes $w = (a, e, d, b, a, b, c, c, d, e)$ aus dem Gegenbeispiel des Hypergraphen $\mathcal{H}_4$.

Beispiel 5.10. *Sei das zulässige Wort $w = (a, e, d, b, a, b, c, c, d, e)$ mit der rekursiven Greedy-Färbung $rg = (0, 0, 0, 0, 1, 1, 1, 0, 1, 1)$ gegeben. Es enthält die Intervalle $I(1, 5)$, $I(2, 10)$, $I(3, 9)$, $I(4, 6)$ und $I(7, 8)$. Wir betrachten nun die rekursive Greedy-Färbung des Wortes und überprüfen, welche Intervalle Hyperkanten in $\mathcal{H}_5$ sind. Im ersten Schritt betrachten wir die Sequenz von $rg = (0, 0, 0, 0, 1, 1, 1, 0, 1, 1)$, die genau einen Farbwechsel enthält. Wir erhalten die Teilsequenz $rg_1 = (0, 0, 0, 0, 1, 1, 1)$ und das zugehörige Teilwort $w_1 = (a, e, d, b, a, b, c)$. In dieser Sequenz sind die beiden Intervalle $I(1, 5)$*

und $I(4,6)$ *enthalten. Da das zweite Vorkommen des Intervalls* $I(4,6)$ *weiter rechts steht als das des Intervalls* $I(1,5)$*, gilt* $I(4,6) \in \mathcal{H}_5$*. Weiter erhalten wir für* $q = 2$ *die Teilsequenz* $rg_2 = (0,0,0,0,1,1,1,0)$ *und das zugehörige Teilwort* $w_2 = (a,e,d,b,a,b,c,c)$ *mit den enthaltenen Intervallen* $I(1,5)$*,* $I(4,6)$ *und* $I(7,8)$*. Es gilt* $I(7,8) \in \mathcal{H}_5$*, da* $j = 8$ *für diese Intervalle maximal ist. Im letzten Schritt gilt* $q = 3$ *und somit* $rg_3 = rg = (0,0,0,0,1,1,1,0,1,1)$ *und* $w_3 = w = (a,e,d,b,a,b,c,c,d,e)$*. Es gilt* $I(2,10) \in \mathcal{H}_5$*. Die drei Hyperkanten in* $\mathcal{H}_5$ *sind die Intervalle* $I(4,6)$*,* $I(7,8)$ *und* $I(2,10)$*. Da die Intervalle* $I(4,6)$ *und* $I(7,8)$ *disjunkt sind und* $I(2,10)$ *die beiden anderen Intervalle enthält, ist* $\mathcal{H}_5(w)$ *geradzahlig laminar.*

Die weitere Untersuchung dieses Hypergraphen zeigt, dass es Wörter gibt, die Hypothese 5 widerlegen. Das Wort $w = (d,b,a,a,c,b,c,d)$ mit der rekursiven Greedy-Färbung $rg = (0,0,0,1,1,1,0,1)$ ist hierfür ein Beispiel. Wir zeigen zunächst folgende Behauptung.

Behauptung 13. *Die rekursive Greedy-Färbung* $rg = (0,0,0,1,1,1,0,1)$ *des Wortes* $w = (d,b,a,a,c,b,c,d)$ *ist optimal.*

Beweis. Sei das Wort $w = (d,b,a,a,c,b,c,d)$ gegeben. Es tritt jeweils mindestens ein Farbwechsel in der Buchstabenfolge (a,a) und (c,b,c) auf. Daraus folgend gibt es mindestens zwei Farbwechsel zwischen den beiden Vorkommen des Buchstabens d. Ist die Anzahl genau zwei und somit gerade, tritt mit Proposition 3.7 ein zusätzlicher Farbwechsel vor dem zweiten Vorkommen des Buchstabens d auf. Im anderen Fall gibt es mindestens drei Farbwechsel zwischen den Vorkommen des Buchstabens d. In beiden Fällen ist die untere Schranke für die Anzahl der Farbwechsel drei. Es folgt die Behauptung. □

Im Folgenden zeigen wir, dass $\mathcal{H}_5(w)$ für $w = (d,b,a,a,c,b,c,d)$ nicht geradzahlig laminar ist und damit Hypothese 5 widerlegt ist.

Behauptung 14. *Sei* $w = (d,b,a,a,c,b,c,d)$*, dann ist* $\mathcal{H}_5(w)$ *nicht geradzahlig laminar.*

Beweis. Die Intervalle $I(i,j)$ des Wortes $w = (d,b,a,a,c,b,c,d)$ sind $I(1,8)$, $I(2,6)$, $I(3,4)$ und $I(5,7)$. Betrachten wir die drei Farbwechsel und die zugehörigen Teilsequenzen $rg_1 = (0,0,0,1,1,1)$, $rg_2 = (0,0,0,1,1,1,0)$ und $rg_3 = (0,0,0,1,1,1,0,1)$, dann sind die Intervalle $I(2,6)$, $I(5,7)$ und $I(1,8)$ Hyperkanten in $\mathcal{H}_5(w)$. Da die beiden Intervalle $I(2,6)$ und $I(5,7)$ nicht disjunkt sind folgt mit Definition 3.5 und 3.6, dass $\mathcal{H}_5(w)$ nicht geradzahlig laminar ist. □

Dadurch können wir die Optimalität der rekursiven Greedy-Färbung dieses Wortes mit dem Hypergraphen $\mathcal{H}_5(w)$ nicht beweisen. Die folgende Darstellung veranschaulicht das Gegenbeispiel.

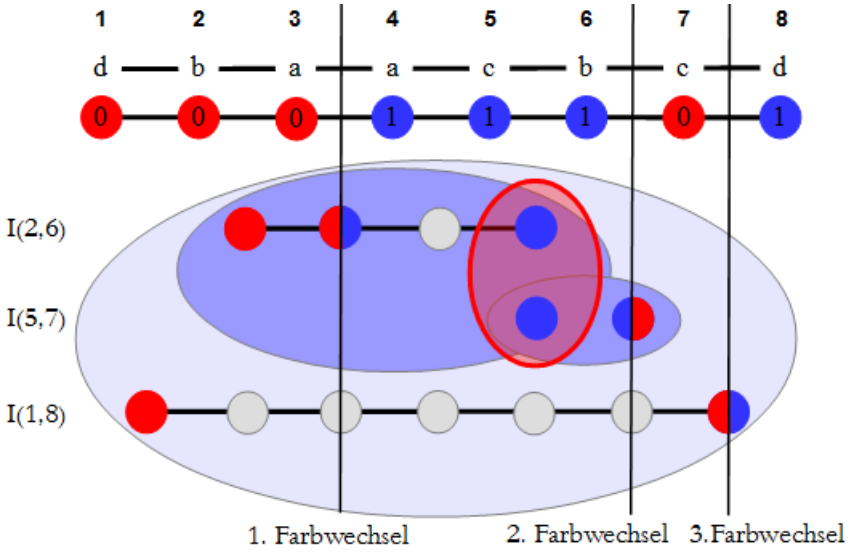

Abbildung 5.13: Die Darstellung zeigt das zulässige Wort w=(d,b,a,a,c,b,c,d) mit der rekursiven Greedy-Färbung rg=(0,0,0,1,1,1,0,1) und dessen Hyperkanten im Hypergraphen $\mathcal{H}_5(w)$. Die Schnittmenge der beiden nicht disjunkten Intervalle $I(2,6)$ und $I(5,7)$ ist durch einen roten Kreis markiert.

Wir können die Optimalität der rekursiven Greedy-Färbung des Wortes $w = (d, b, a, a, c, b, c, d)$ mit dem Hypergraphen $\mathcal{H}_3$ zeigen.

Wir verfolgen weiter den Ansatz die Hyperkanten innerhalb einer Teilsequenz der Färbung zu betrachten. Die Überlegung einen Hypergraphen zu definieren, dessen Hyperkanten die Intervalle innerhalb der betrachteten Sequenz sind und deren zweites Vorkommen an der am zweit weitest rechts stehenden Position liegt, führt nicht zum Ziel. Betrachten wir hierzu den Hypergraphen $\mathcal{H}_{5_1}$.

Definition 5.11. *(Hypergraph $\mathcal{H}_{5_1}$) Sei $k \in \mathbb{N}$ die Anzahl der Farbwechsel einer rekursiven Greedy-Färbung rg. Sei $rg_q = (rg_0, ..., rg_l)$ mit $q, l \in \mathbb{N}$ und $1 \leq q \leq k$ eine Sequenz der Färbung, die genau q Farbwechsel enthält. Sei $\mathcal{H}_{5_1}$ ein Hypergraph, dessen Hyperkanten die Intervalle $I(i, j)$ der Buchstaben in w sind, die in einer Sequenz rg_q enthalten sind und deren zweites Vorkommen an der am zweit weitest rechts stehenden Position in dieser Sequenz liegt. Es ist*

$\mathcal{H}_{5_1}(w) = \{I(i, j) \in l(w) | I(i, j) \in rg_q$ *und j liegt an der zweit weitest rechts stehenden Position innerhalb dieser Sequenz, für alle* $1 \leq q \leq k\}$.

Betrachten wir dazu das Gegenbeispiel aus Abbildung 5.13 und zeigen die folgende Behauptung.

Behauptung 15. *Sei $w = (d, b, a, a, c, b, c, d)$, dann ist $\mathcal{H}_{5_1}(w)$ nicht geradzahlig laminar.*

Beweis. Die Intervalle $I(i, j)$ des Wortes w sind $I(1, 8), I(2, 6), I(3, 4)$ und $I(5, 7)$. Betrachten wir die drei Farbwechsel und zugehörigen Teilsequenzen $rg_1 = (0, 0, 0, 1, 1, 1)$, $rg_2 = (0, 0, 0, 1, 1, 1, 0)$ und $rg_3 = (0, 0, 0, 1, 1, 1, 0, 1)$, dann sind die Hyperkanten in $\mathcal{H}_{5_1}(w)$ die Intervalle $I(3, 4)$, $I(2, 6)$ und $I(5, 7)$. Da die beiden Intervalle $I(2, 6)$ und $I(5, 7)$ nicht disjunkt sind folgt mit Definition 3.5 und 3.6, dass $\mathcal{H}_{5_1}(w)$ nicht geradzahlig laminar ist. □

Dadurch können wir die Optimalität der rekursiven Greedy-Färbung dieses Wortes mit dem Hypergraphen $\mathcal{H}_{5_1}(w)$ nicht beweisen.

Analog zu den beiden bisher betrachteten Möglichkeiten, die Hyperkanten innerhalb einer Sequenz der rekursiven Greedy-Färbung eines Wortes w zu definieren, konstruieren wir nun den Hypergraphen $\mathcal{H}_{5_2}$. Die Hyperkanten von $\mathcal{H}_{5_2}$ sind die Intervalle der Buchstaben innerhalb der betrachteten Sequenz, deren erstes Vorkommen an der am weitest rechts stehenden Position ist.

Definition 5.12. *(Hypergraph $\mathcal{H}_{5_2}$) Sei $k \in \mathbb{N}$ die Anzahl der Farbwechsel einer rekursiven Greedy-Färbung rg. Sei $rg_q = (rg_0, ..., rg_l)$ mit $q, l \in \mathbb{N}$ und $1 \leq q \leq k$ eine Sequenz der Färbung, die genau q Farbwechsel enthält. Sei $\mathcal{H}_{5_2}$ ein Hypergraph. Die Hyperkanten in $\mathcal{H}_{5_2}$ sind die Intervalle $I(i, j)$ der Buchstaben eines zulässigen Wortes w, die in einer Sequenz rg_q enthalten sind und für die $i = max\{i | I(i, j)$ ist in $rg_q\}$ für alle $1 \leq q \leq k$ gilt. Es ist*

$\mathcal{H}_{5_2}(w) = \{I(i, j) \in l(w) | I(i, j) \in rg_q$ *und* $i = max\{i | I(i, j)$ *ist in* $rg_q\}$, *für alle* $1 \leq q \leq k\}$

Betrachten wir dazu das Gegenbeispiel aus Abbildung 5.13 und zeigen die folgende Behauptung.

Behauptung 16. *Sei $w = (d, b, a, a, c, b, c, d)$, dann ist $\mathcal{H}_{5_2}(w)$ nicht geradzahlig laminar.*

Beweis. Die Intervalle $I(i, j)$ des Wortes w sind $I(1, 8), I(2, 6), I(3, 4)$ und $I(5, 7)$. Betrachten wir die drei Farbwechsel und zugehörigen Teilsequenzen $rg_1 = (0, 0, 0, 1, 1, 1)$, $rg_2 = (0, 0, 0, 1, 1, 1, 0)$ und $rg_3 = (0, 0, 0, 1, 1, 1, 0, 1)$ dann sind die Hyperkanten in $\mathcal{H}_{5_2}(w)$ die Intervalle $I(3, 4)$, $I(5, 7)$ und $I(2, 6)$. Da die beiden Intervalle $I(5, 7)$ und $I(2, 6)$ nicht disjunkt sind, folgt mit Definition 3.5 und 3.6, dass $\mathcal{H}_{5_2}(w)$ nicht geradzahlig laminar ist. □

Dadurch können wir die Optimalität der rekursiven Greedy-Färbung dieses Wortes mit dem Hypergraphen $\mathcal{H}_{5_2}(w)$ nicht beweisen.

Untersuchung des Hypergraphen $\mathcal{H}_6$

Wir verfolgen nun den Ansatz einen Hypergraphen $\mathcal{H}_6$ über ein zweistufiges Verfahren zu definieren. In der ersten Stufe betrachten wir den Hypergraphen, dessen Hyperkanten die Intervalle $I(i,j)$ der Buchstaben sind, durch deren Färbung im rekursiven Greedy-Algorithmus ein Farbwechsel verursacht wird. Wir haben diesen Hypergraphen als $\mathcal{H}_4$ bereits definiert.

Definition 5.13. *(Hypergraph $\mathcal{H}_4$) Sei $\mathcal{H}_4$ ein Hypergraph. Die Hyperkanten in $\mathcal{H}_4$ sind die Intervalle $I(i,j)$ der Buchstaben eines zulässigen Wortes w, durch deren Färbung mit dem rekursiven Greedy-Algorithmus ein Farbwechsel verursacht wird. Es ist*

$\mathcal{H}_4(w) = \{I(i,j) \in l(w) |$ *Färbung von* $w_i = w_j$ *verursacht einen Farbwechsel*$\}$.

In der zweiten Stufe betrachten wir nun jeweils die längstmögliche Teilsequenz der Färbung und das zugehörige Teilwort, die eine der Hyperkanten aus $\mathcal{H}_4$, aber keinen zusätzlichen Farbwechsel, enthält. In dieser Teilsequenz betrachten wir das darin enthaltene Intervall $I(i,j)$ eines Buchstabens, dessen Vorkommen w_i und w_j einen minimalen Abstand zueinander haben. Dies führt uns zur Definition des Hypergraphen $\mathcal{H}_6$.

Definition 5.14. *(Hypergraph $\mathcal{H}_6$) Sei $I(i,j)_q \in \mathcal{H}_4$ und sei $k \in \mathbb{N}$ die Anzahl der Farbwechsel der rekursiven Greedy-Färbung rg. Sei $rg_q = (rg_0, ..., rg_l)$ mit $q, l \in \mathbb{N}$ und $1 \leq q \leq k$ eine Sequenz der Färbung, die den Farbwechsel eines Intervalls $I(i,j)_q$, aber keinen zusätzlichen Farbwechsel, enthält. Sei $\mathcal{H}_6$ ein Hypergraph. Die Hyperkanten in $\mathcal{H}_6$ sind die Intervalle $I(i,j)$ der Buchstaben eines zulässigen Wortes, die in einer Sequenz rg_q enthalten sind und für die $|i-j| = min\{|i-j||I(i,j)$ ist in $rg_q\}$ gilt. Es ist*

$\mathcal{H}_6 = \{I(i,j) \in l(w) | I(i,j) \in rg_q$ *und* $|i-j| = min\{|i-j||I(i,j)$ *ist in* $rg_q\}$, *für alle* $1 \leq q \leq k\}$.

Wir veranschaulichen den Hypergraphen $\mathcal{H}_6$ und die Definition seiner Hyperkanten an folgendem Beispiel.

Beispiel 5.15. *Sei das zulässige Wort $w = (a,e,d,b,a,b,c,c,d,e)$ mit der rekursiven Greedy-Färbung $rg = (0,0,0,0,1,1,1,0,1,1)$ gegeben. Es enthält die Intervalle $I(1,5), I(2,10), I(4,6), I(3,9)$ und $I(7,8)$. Im ersten Schritt betrachten wir die rekursive Greedy-Färbung des Wortes und überprüfen welche Intervalle in $\mathcal{H}_4$ sind. Es werden Farbwechsel durch die Färbung der Buchstaben a, c und d verursacht. Damit sind die Intervalle $I(1,5)$, $I(7,8)$ und $I(3,9)$ Hyperkanten in $\mathcal{H}_4$. Betrachten wir nun im zweiten Schritt die Teilsequenzen die diese Hyperkanten, aber keine zusätzlichen Farbwechsel, enthalten. Für die erste Hyperkante $I(1,5)$ ist das zugehörige Teilwort $w_1 = (a,e,d,b,a,b,c)$ mit der $0/1-Sequenz$ $(0,0,0,0,1,1,1)$. In diesem Teilwort*

ist der Abstand der beiden Vorkommen des Buchstabens b am geringsten. Damit ist $I(4,6) \in \mathcal{H}_6$. *Die zweite Hyperkante* $I(7,8) \in \mathcal{H}_4$ *ist ein Paar, damit ist der Abstand der Vorkommen bereits minimal und es folgt* $I(7,8) \in \mathcal{H}_6$. *Da in der letzten Hyperkante* $I(3,9) \in \mathcal{H}_4$ *der Farbwechsel vor dem Buchstaben d auftritt und der Abstand zwischen den beiden Vorkommen des Buchstabens d geringer ist als der zwischen den beiden Vorkommen des Buchstabens e, gilt somit* $I(3,9) \in \mathcal{H}_6$. *Da die Intervalle* $I(4,6)$ *und* $I(7,8)$ *disjunkt sind und* $I(3,9)$ *die beiden anderen Intervalle enthält ist* $\mathcal{H}_6(w)$ *geradzahlig laminar.*

Die weitere Untersuchung dieses Hypergraphen zeigt, dass es Wörter gibt die Hypothese 5 widerlegen. Das Wort $w = (a, d, b, e, a, b, c, c, d, e)$ mit der rekursiven Greedy-Färbung $rg = (0, 0, 0, 0, 1, 1, 1, 0, 1, 1)$ ist hierfür ein Beispiel. Wir zeigen zunächst folgende Behauptung.

Behauptung 17. *Die rekursive Greedy-Färbung* $rg = (0, 0, 0, 0, 1, 1, 1, 0, 1, 1)$ *des Wortes* $w = (a, d, b, e, a, b, c, c, d, e)$ *ist optimal.*

Beweis. Sei das Wort $w = (a, d, b, e, a, b, c, c, d, e)$ gegeben. Es tritt jeweils mindestens ein Farbwechsel in der Buchstabenfolge (b, e, a, b) und (c, c) auf. Damit gibt es mindestens zwei Farbwechsel zwischen den beiden Vorkommen des Buchstabens d. Ist die Anzahl genau zwei und somit gerade, tritt mit Proposition 3.7 ein zusätzlicher Farbwechsel vor dem zweiten Vorkommen des Buchstabens d auf. Im anderen Fall gibt es mindestens drei Farbwechsel zwischen den Buchstaben d. In beiden Fällen ist die untere Schranke für die Anzahl der Farbwechsel drei. Es folgt die Behauptung. □

Im Folgenden zeigen wir, dass $\mathcal{H}_6(w)$ für $w = (a, d, b, e, a, b, c, c, d, e)$ nicht geradzahlig laminar ist und damit Hypothese 5 widerlegt ist.

Behauptung 18. *Sei* $w = (a, d, b, e, a, b, c, c, d, e)$, *dann ist* $\mathcal{H}_6(w)$ *nicht geradzahlig laminar.*

Beweis. Die Intervalle $I(i, j)$ des Wortes w sind $I(1,5), I(2,9), I(3,6), I(4,10)$ und $I(7,8)$. Die Hyperkanten in $\mathcal{H}_4(w)$ sind die Intervalle $I(1,5)$, $I(7,8)$ und $I(2,9)$, da durch die rekursive Greedy-Färbung dieser Buchstaben ein Farbwechsel verursacht wird. Betrachten wir das Teilwort, das die Hyperkante $I(1,5)$ und keinen zusätzlichen Farbwechsel enthält. Es folgt $I(3,6) \in \mathcal{H}_6$, da $|3-6| < |1-5|$ gilt. Weiter ist $I(7,8)$ eine Hyperkante in $\mathcal{H}_6$, da $|i-j| = |7-8| = 1$ bereits minimal ist. Zur dritten Hyperkante $I(2,9) \in \mathcal{H}_4(w)$ betrachten wir das zugehörige Teilwort mit den Intervallen $I(2,9)$ und $I(4,10)$. Da die Distanz der Buchstaben im Intervall $I(4,10)$ geringer ist als die im Intervall $I(2,9)$, gilt $I(4,10) \in \mathcal{H}_6$. Da die beiden Intervalle $I(3,6)$ und $I(4,10)$ nicht disjunkt sind folgt mit Definition 3.5 und 3.6, dass $\mathcal{H}_6(w)$ nicht geradzahlig laminar ist. □

Somit können wir die Optimalität der rekursiven Greedy-Färbung dieses Wortes mit dem Hypergraphen $\mathcal{H}_6(w)$ nicht beweisen. Die folgende Darstellung veranschaulicht das Gegenbeispiel.

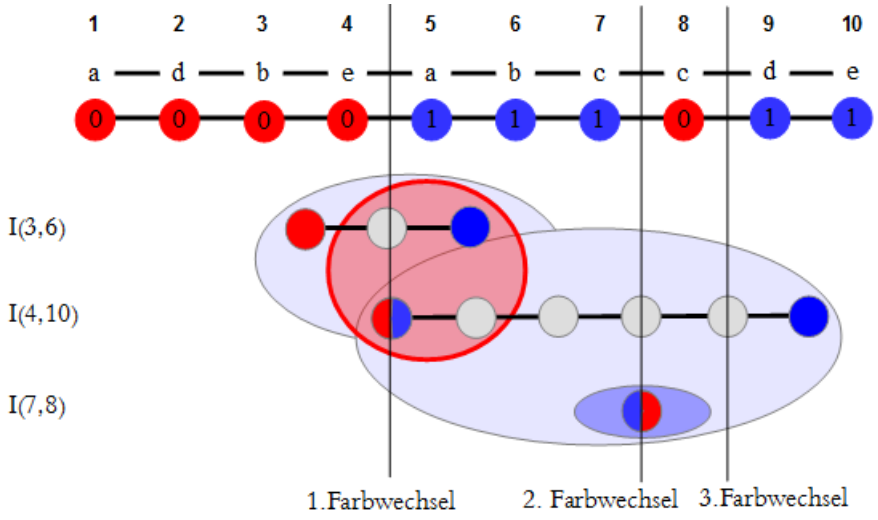

Abbildung 5.14: Die Darstellung zeigt das zulässige Wort w=(a,d,b,e,a,b,c,c,d,e) mit der rekursiven Greedy-Färbung rg=(0,0,0,0,1,1,1,0,1,1) und dessen Hyperkanten im Hypergraph $\mathcal{H}_6(w)$. Die Schnittmenge der beiden nicht disjunkten Intervalle $I(3,6)$ und $I(4,10)$ ist durch einen roten Kreis markiert.

Es existiert ein Hypergraph, bezüglich der rekursiven Greedy-Färbung, der die Optimalität der rekursiven Greedy-Färbung dieses Wortes beweist. Diesen Hypergraphen werden wir detailliert im nächsten Abschnitt beschreiben.

Untersuchung des Hypergraphen $\mathcal{H}_7$

Das Gegenbeispiel aus dem vorherigen Abschnitt führt zu folgender Überlegung. Wie können wir mit dem Ansatz des Hypergraphen $\mathcal{H}_6$ sicherstellen, dass keine Hyperkante eine andere Hyperkante schneidet? Deshalb ergänzen wir den Hypergraphen $\mathcal{H}_6$ mit folgender Regel und konstruieren den Hypergraphen $\mathcal{H}_7$. Betrachten wir ein zu einer Hyperkante aus $\mathcal{H}_4$ zugehöriges Teilwort, dann ist die Hyperkante in $\mathcal{H}_7$ ein Intervall das den Regeln von $\mathcal{H}_6$ genügt und keine bereits in $\mathcal{H}_7$ enthaltene Hyperkante schneidet. Dies führt uns zur Definition des Hypergraphen $\mathcal{H}_7$.

Definition 5.16. *(Hypergraph $\mathcal{H}_7$) Sei $I(i,j)_q \in \mathcal{H}_4$ und sei $k \in \mathbb{N}$ die Anzahl der Farbwechsel der rekursiven Greedy-Färbung rg. Sei $rg_q = (rg_0, ..., rg_l)$ mit $q, l \in \mathbb{N}$ und $1 \leq q \leq k$ eine Sequenz der Färbung, die den Farbwechsel eines Intervalls $I(i,j)_q$ und keinen Zusätzlichen enthält. Sei $\mathcal{H}_7$ ein Hypergraph. Die Hyperkanten in $\mathcal{H}_7$ sind die Intervalle $I(i,j)$ der Buchstaben in*

w, die in einer Sequenz rg_q enthalten sind und für die gilt:

(i) $|i-j| = min\{|i-j||I(i,j)$ *ist in* rg_q *enthalten* $\}$ *und*

(ii) $I(i,j) \cap I(i,j)_{\mathcal{H}_7} = \emptyset$, *falls gilt* $I(i,j) \not\subset I(i,j)_{\mathcal{H}_7}$ *und* $I(i,j) \not\supset I(i,j)_{\mathcal{H}_7}$, *für alle* $I(i,j)_{\mathcal{H}_7} \in \mathcal{H}_7$.

Dann ist

$\mathcal{H}_7(w) = \{I(i,j) \in l(w) | I(i,j) \in rg_q$ *und* $I(i,j)$ *erfüllt (i) und (ii) für alle* $1 \leq q \leq k\}$.

Wir veranschaulichen die Definition des Hypergraphen $\mathcal{H}_7$ am Gegenbeispiel des vorherigen Abschnitts.

Beispiel 5.17. *Sei das zulässige Wort* $w = (a,d,b,e,a,b,c,c,d,e)$ *mit der rekursiven Greedy-Färbung* $rg = (0,0,0,0,1,1,1,0,1,1)$ *gegeben. Es enthält die Intervalle* $I(1,5), I(2,9), I(3,6), I(4,10)$ *und* $I(7,8)$. *Im ersten Schritt betrachten wir die rekursive Greedy-Färbung des Wortes und überprüfen, welche Intervalle in* $\mathcal{H}_4$ *sind. Farbwechsel werden durch die Färbung der Buchstaben* a, c *und* d *verursacht. Damit sind die Intervalle* $I(1,5), I(2,9)$ *und* $I(7,8)$ *Hyperkanten in* $\mathcal{H}_4$. *Betrachten wir nun im zweiten Schritt die Teilwörter, die diese Hyperkanten, aber keine zusätzlichen Farbwechsel enthalten. Das zur ersten Hyperkante* $I(1,5)$ *zugehörige Teilwort ist* $w_1 = (a,d,b,e,a,b,c)$ *mit der* $0/1-Sequenz$ $(0,0,0,0,1,1,1)$. *In diesem Teilwort ist der Abstand der beiden Vorkommen des Buchstabens* b *am kleinsten und das Intervall* $I(3,6)$ *schneidet keine weitere Hyperkante aus* $\mathcal{H}_7$. *Damit sind die Bedingungen* (i) *und* (ii) *aus Definition 5.16 erfüllt und somit ist* $I(3,6) \in \mathcal{H}_7$. *Die zweite Hyperkante* $I(7,8) \in \mathcal{H}_4$ *ist ein Paar, damit ist der Abstand der Vorkommen bereits minimal und die beiden Intervalle* $I(3,6)$*und* $I(7,8)$ *sind disjunkt. Es folgt* $I(7,8) \in \mathcal{H}_7$. *In der letzten Hyperkante* $I(2,9) \in \mathcal{H}_4$ *tritt der Farbwechsel vor dem zweiten Vorkommen des Buchstabens* d *auf. Betrachten wir das zugehörige Teilwort, dann ist der Abstand zwischen den beiden Vorkommen des Buchstabens* e *in diesem Teilwort am geringsten. Da das zugehörige Intervall* $I(4,10)$ *nicht disjunkt zur Hyperkante* $I(3,6)$ *aus* $\mathcal{H}_7$ *ist, wird Bedingung* (ii) *aus Definition 5.16 nicht erfüllt. Somit ist das Intervall mit der zweitkleinsten Distanz* $I(2,9)$ *eine Hyperkante in* $\mathcal{H}_7$. *Da die Intervalle* $I(3,6)$ *und* $I(7,8)$ *disjunkt sind und* $I(2,9)$ *die beiden anderen Intervalle enthält ist* $\mathcal{H}_7(w)$ *geradzahlig laminar.*

In der weiteren Analyse des Hypergraphen $\mathcal{H}_7$ zeigt sich, dass es Wörter gibt, deren rekursive Greedy-Färbung in allen Teilwörtern optimal ist, aber wir dies mit dem Hypergraphen $\mathcal{H}_7$ nicht beweisen können. Ein Beispiel hiefür ist das Wort $w = (b,a,a,c,e,b,c,d,d,e)$ mit der rekursiven Greedy-Färbung $rg = (0,0,1,1,1,1,0,0,1,0)$. Wir zeigen zunächst folgende Behauptung.

Behauptung 19. *Die rekursive Greedy-Färbung* $rg = (0, 0, 1, 1, 1, 1, 0, 0, 1, 0)$ *des Wortes* $w = (b, a, a, c, e, b, c, d, d, e)$ *ist optimal.*

Beweis. Sei das Wort $w = (b, a, a, c, e, b, c, d, d, e)$ gegeben. Es tritt jeweils mindestens ein Farbwechsel in den Buchstabenfolgen (a, a), (c, e, b, c) und (d, d) auf. Tritt im ersten Fall nur ein Farbwechsel an der Stelle (a, a) zwischen den beiden Vorkommen des Buchstabens b auf, dann muss ein weiterer Farbwechsel vor dem zweiten Vorkommen des Buchstabens c stattfinden. Mit dem nötigen Farbwechsel an der Stelle (d, d) gibt es zwei und somit eine gerade Anzahl von Farbwechsel zwischen den beiden Vorkommen des Buchstabens e. Dadurch tritt mit Proposition 3.7 ein zusätzlicher Farbwechsel vor dem zweiten Vorkommen des Buchstabens e auf. Im anderen Fall tritt nach dem Farbwechsel in der Folge (a, a) der zweite Farbwechsel in der Folge (c, e, b, c), vor dem ersten Vorkommen des Buchstabens e, auf. Damit gibt es zwischen den beiden Vorkommen des Buchstabens b zwei und somit eine gerade Anzahl von Farbwechsel. Mit Proposition 3.7 tritt ein zusätzlicher Farbwechsel vor dem zweiten Vorkommen des Buchstabens b auf. Ein vierter Farbwechsel findet in der Folge (d, d) statt. In beiden Fällen ist die untere Schranke für die Anzahl der Farbwechsel vier. Es folgt die Behauptung. □

Im Folgenden zeigen wir, dass wir die Optimalität der rekursiven Greedy-Färbung für das Wort $w = (b, a, a, c, e, b, c, d, d, e)$ mit dem Hypergraphen $\mathcal{H}_7$ nicht beweisen können.

Behauptung 20. *Sei* $w = (b, a, a, c, e, b, c, d, d, e)$*, dann ist die Anzahl der Hyperkanten in* $\mathcal{H}_7(w)$ *geringer als die untere Schranke für die Anzahl der Farbwechsel der rekursiven Greedy-Färbung des Wortes* w*.*

Beweis. Die Intervalle $I(i, j)$ des Wortes w sind $I(1, 6)$, $I(2, 3)$, $I(4, 7)$, $I(8, 9)$ und $I(5, 10)$. Die Hyperkanten in $\mathcal{H}_4(w)$ sind die Intervalle $I(2, 3)$, $I(4, 7)$, $I(8, 9)$ und $I(5, 10)$, da durch die rekursive Greedy-Färbung dieser Buchstaben ein Farbwechsel verursacht wird. Betrachten wir das Teilwort, das die Hyperkante $I(2, 3)$ und keinen weiteren Farbwechsel enthält. Es folgt $I(2, 3) \in \mathcal{H}_7$, da $|2-3| = 1$ minimal ist und es keine Schnittmenge mit einem anderen Intervall gibt. Mit der gleichen Argumentation folgt $I(4, 7), I(7, 8) \in \mathcal{H}_7$. Betrachten wir nun das Teilwort des Intervalls $I(5, 10)$ aus $\mathcal{H}_4(w)$, dann sind die beiden Intervalle $I(1, 6)$ und $I(5, 10)$ darin enthalten. Die Distanz beider Vorkommen der Buchstaben dieser Intervalle ist gleich. Da jedes dieser beiden Intervalle das Intervall $I(4, 7)$ schneidet, erfüllen diese die Bedingung (ii) der Definition 5.16 des Hypergraphen $\mathcal{H}_7$ nicht. Der vierte Farbwechsel lässt sich durch eine Hyperkante nicht abbilden und es folgt die Aussage. □

Mit diesem Resultat beenden wir die Untersuchung verschiedener Hypergraphen und formulieren die Ergebnisse.

5.2 Das Ergebnis der Untersuchung verschiedener Hypergraphen

In diesem Kapitel werden wir das Gegenbeispiel $w = (b, a, a, c, e, b, c, d, d, e)$ des vorherigen Abschnitts betrachten und zeigen, dass wir mit diesem Wort Hypothese 5 für alle Hypergraphen, die wir bezüglich einer guten Färbung des rekursiven Greedy-Algorithmus definieren widerlegen können. Wir werden untersuchen, ob es mehrere Wörter dieser Form gibt und die Ergebnisse dieser Analyse vorstellen.

Betrachten wir zunächst das Wort $w = (b, a, a, c, e, b, c, d, d, e)$ und dessen rekursive Greedy-Färbung $rg = (0, 0, 1, 1, 1, 1, 0, 0, 1, 0)$. Mit Behauptung 19 ist die untere Schranke für die Anzahl der Farbwechsel vier und die rekursive Greedy-Färbung somit optimal. Die Intervalle des Wortes w sind $I(1, 6)$, $I(2, 3)$, $I(4, 7)$, $I(5, 10)$ und $I(8, 9)$. Um Hypothese 5 zu beweisen, müssen wir einen Hypergraphen basierend auf einer guten Färbung des rekursiven Greedy-Algorithmus konstruieren, der für das Wort w geradzahlig laminar ist und dessen Anzahl von Hyperkanten der Anzahl von Farbwechsel der rekursiven Greedy-Färbung des betrachteten Wortes entspricht. Wir zeigen folgende Behauptung.

Behauptung 21. *Sei $w = (b, a, a, c, e, b, c, d, d, e)$. Es existiert kein bezüglich einer guten rekursiven Greedy-Färbung definierter Hypergraph $\mathcal{H}(w)$, der für w geradzahlig laminar ist und alle nötigen Farbwechsel durch Hyperkanten abbildet.*

Beweis. Sei $w = (b, a, a, c, e, b, c, d, d, e)$ und $\mathcal{H}$ ein Hypergraph, der bezüglich einer guten rekursiven Greedy-Färbung definiert ist. Da mit Behauptung 19 die untere Schranke für die Anzahl der Farbwechsel vier ist muss $\mathcal{H}(w)$ vier Hyperkanten enthalten, um die Optimalität der rekursiven Greedy-Färbung beweisen zu können. Betrachten wir die Intervalle $I(1, 6)$, $I(2, 3)$, $I(4, 7)$, $I(5, 10)$ und $I(8, 9)$ des Wortes w unabhängig vom Auftreten eines Farbwechsels. Untersuchen wir nun alle fünf Möglichkeiten vier Hyperkanten aus der Menge der fünf Intervalle zu wählen.
Fall 1: Die Hyperkanten in $\mathcal{H}$ sind die Intervalle $I(2, 3)$, $I(4, 7)$, $I(5, 10)$ und $I(8, 9)$. Da die beiden Intervalle $I(4, 7)$ und $I(5, 10)$ nicht disjunkt sind und einander nicht enthalten folgt, dass der Hypergraph $\mathcal{H}(w)$ nicht geradzahlig laminar ist.
Fall 2: Die Hyperkanten in $\mathcal{H}$ sind die Intervalle $I(1, 6)$, $I(4, 7)$, $I(5, 10)$ und $I(8, 9)$. Da die beiden Intervalle $I(4, 7)$ und $I(5, 10)$ nicht disjunkt sind und einander nicht enthalten folgt, dass der Hypergraph $\mathcal{H}(w)$ nicht geradzahlig laminar ist.
Fall 3: Die Hyperkanten in $\mathcal{H}$ sind die Intervalle $I(1, 6), I(2, 3), I(5, 10)$ und $I(8, 9)$. Da die beiden Intervalle $I(1, 6)$ und $I(5, 10)$ nicht disjunkt sind und einander nicht enthalten folgt, dass der Hypergraph $\mathcal{H}(w)$ nicht geradzahlig

laminar ist.
Fall 4: Die Hyperkanten in $\mathcal{H}$ sind die Intervalle $I(1,6), I(2,3), I(4,7)$ und $I(8,9)$. Da die beiden Intervalle $I(1,6)$ und $I(4,7)$ nicht disjunkt sind und einander nicht enthalten folgt, dass der Hypergraph $\mathcal{H}(w)$ nicht geradzahlig laminar ist.
Fall 5: Die Hyperkanten in $\mathcal{H}$ sind die Intervalle $I(1,6), I(2,3), I(4,7)$ und $I(5,10)$. Da die Intervalle $I(1,6), I(4,7)$ und $I(5,10)$ nicht paarweise disjunkt sind und einander nicht enthalten folgt, dass der Hypergraph $\mathcal{H}(w)$ nicht geradzahlig laminar ist.

Damit gibt es keine Möglichkeit einen Hypergraphen $\mathcal{H}$, bezüglich einer guten rekursiven Greedy-Färbung zu definieren, sodass $\mathcal{H}(w)$ für das zulässige Wort $w = (b, a, a, c, e, b, c, d, d, e)$ geradzahlig laminar ist und die Anzahl der Hyperkanten in $\mathcal{H}(w)$ der Anzahl der Farbwechsel in der rekursiven Greedy-Färbung des Wortes entspricht. Es folgt die Behauptung. □

Daraus folgend müssen wir die Hypothese 5 und damit die Vermutung, dass eine Charakterisierung des rekursiven Greedy-Algorithmus, basierend auf der Idee der Charakterisierung des Greedy-Algorithmus, möglich ist verwerfen.
Diese Erkenntnis führt uns zur Frage, ob es mehrere Wörter der Form von Behauptung 21 gibt. Um dies zu untersuchen entwickeln wir eine weitere Java Methode *'laminar_hyper'*. Zunächst definieren wir den Begriff der laminaren Intervalle und beschreiben anschließend die Vorgehensweise dieser Methode.

Definition 5.18. *Sei $l(w)$ die Menge der Intervalle eines Wortes. Wir nennen ein Intervall $I(i,j) \in l(w)$ ein laminares Intervall, wenn für alle $I(i,j)_A$, $I(i,j)_B \in l(w)$ eine der beiden Aussagen gilt:*

(i) $I(i,j)_A \subset I(i,j)_B$ *oder* $I(i,j)_B \subset I(i,j)_A$

(ii) $I(i,j)_A \cap I(i,j)_B = \emptyset$.

Die Methode *'laminar_hyper'*
Diese Methode durchläuft in einer Schleife alle Wörter der Liste *'kombinationen'* der jeweils betrachteten Wortlänge und ermittelt alle Intervalle des Wortes. Diese Intervalle werden in die Liste *'intervalle'* geschrieben. In einer weiteren Schleife wird für jedes Intervall eines Wortes jeweils die erste und letzte Zahl bestimmt. Damit wird überprüft, ob ein Intervall die Bedingungen aus Definition 5.18 erfüllt. Ist dies der Fall wird dieses Intervall in eine Liste *'lam_intervall'* geschrieben. Die Anzahl dieser laminaren Intervalle wird für jedes Wort bestimmt und an die zum Wort zugehörige Position in eine List *'anzahl_lam_intervalle'* geschrieben.

Diese Anzahl vergleichen wir mit der Anzahl der Farbwechsel der rekursiven Greedy-Färbung jedes Wortes. Ist die Anzahl der laminaren Intervalle kleiner als die Anzahl der Farbwechsel der rekursiven Greedy-Färbung des

Wortes gibt das einen Hinweis darauf, dass es sich um ein Wort der Form aus Behauptung 21 handeln könnte. Diese Wörter werden ausgewertet.

Mit dieser Methode und der Analyse der 'verdächtigen' Wörter der Wortlängen 10 und 12 erhalten wir als Ergebnis folgende Wörter der Form aus Behauptung 21. In den folgenden Tabellen bezeichnen wir die maximale Anzahl der laminaren Intervalle mit $K_{I_{lam}}$ und die Anzahl der Farbwechsel der rekursiven Greedy-Färbung mit K_{rg}, wobei gilt $K_{I_{lam}}, K_{rg} \in \mathbb{N}$.

Tabelle 5.11: Wörter der Länge 10 und deren rekursive Greedy-Färbung

Wort	$rg(w)$	K_{rg}
$(b, a, a, c, e, b, c, d, d, e)$	$(0, 0, 1, 1, 1, 1, 0, 0, 1, 0)$	4

Tabelle 5.12: Wörter der Länge 10 und deren maximale Anzahl laminarer Intervalle

Wort	**laminare Intervalle**	$K_{I_{lam}}$
$(b, a, a, c, e, b, c, d, d, e)$	$(I(2, 3), I(4, 7), I(8, 9))$	3

Tabelle 5.13: Wörter der Länge 12 und deren rekursive Greedy-Färbung

Wort	$rg(w)$	K_{rg}
$(b,c,a,a,d,f,b,c,d,e,e,f)$	$(0,0,0,1,1,1,1,1,0,0,1,0)$	4
$(b,d,a,a,c,b,c,e,d,f,e,f)$	$(0,0,0,1,1,1,0,0,1,1,1,0)$	4
$(b,a,d,a,c,f,b,c,e,d,e,f)$	$(0,0,1,1,1,1,1,0,0,0,1,0)$	4
$(b,a,d,a,c,f,b,c,d,e,e,f)$	$(0,0,1,1,1,1,1,0,0,0,1,0)$	4
$(b,a,a,d,c,f,b,c,e,d,e,f)$	$(0,0,1,1,1,1,1,0,0,0,1,0)$	4
$(b,a,a,d,c,f,b,c,d,e,e,f)$	$(0,0,1,1,1,1,1,0,0,0,1,0)$	4
$(b,a,a,c,f,d,b,c,d,e,e,f)$	$(0,0,1,1,1,1,1,0,0,0,1,0)$	4
$(b,a,a,c,d,f,b,c,d,e,e,f)$	$(0,0,1,1,1,1,1,0,0,0,1,0)$	4
$(f,b,a,a,c,e,b,c,d,d,e,f)$	$(0,0,0,1,1,1,1,0,0,1,0,1)$	5
$(b,a,a,c,f,e,b,c,d,d,e,f)$	$(0,0,1,1,1,1,1,0,0,1,0,0)$	4
$(b,a,a,c,e,f,b,c,d,d,e,f)$	$(0,0,1,1,1,1,1,0,0,1,0,0)$	4
$(b,a,a,c,e,b,c,d,d,e,f,f)$	$(0,0,1,1,1,1,0,0,1,0,0,1)$	5
$(b,a,a,c,f,b,c,d,e,d,e,f)$	$(0,0,1,1,1,1,0,0,0,1,1,0)$	4

Tabelle 5.14: Wörter der Länge 12 und deren maximale Anzahl laminarer Intervalle

Wort	**laminare Intervalle**	$K_{I_{lam}}$
$(b, c, a, a, d, f, b, c, d, e, e, f)$	$(I(1,7), I(3,4), I(10,11))$	3
$(b, d, a, a, c, b, c, e, d, f, e, f)$	$(I(1,6), I(3,4), I(8,11))$	3
$(b, a, d, a, c, f, b, c, e, d, e, f)$	$(I(1,7), I(2,4), I(9,11))$	3
$(b, a, d, a, c, f, b, c, d, e, e, f)$	$(I(1,7), I(2,4), I(10,11))$	3
$(b, a, a, d, c, f, b, c, e, d, e, f)$	$(I(1,7), I(2,3), I(9,11))$	3
$(b, a, a, d, c, f, b, c, d, e, e, f)$	$(I(1,7), I(2,3), I(10,11))$	3
$(b, a, a, c, f, d, b, c, d, e, e, f)$	$(I(1,7), I(2,3), I(10,11))$	3
$(b, a, a, c, d, f, b, c, d, e, e, f)$	$(I(1,7), I(2,3), I(10,11))$	3
$(f, b, a, a, c, e, b, c, d, d, e, f)$	$(I(1,12), I(2,7), I(3,4), I(9,10))$	4
$(b, a, a, c, f, e, b, c, d, d, e, f)$	$(I(1,7), I(2,3), I(9,10))$	3
$(b, a, a, c, e, f, b, c, d, d, e, f)$	$(I(1,7), I(2,3), I(9,10))$	3
$(b, a, a, c, e, b, c, d, d, e, f, f)$	$(I(1,6), I(2,3), I(8,9), I(11,12))$	4
$(b, a, a, c, f, b, c, d, e, d, e, f)$	$(I(1,6), I(2,3), I(8,10))$	3

6 Odd-Row-Sum-Packings

In diesem Kapitel wird der Begriff Odd-Row-Sum-Packing eingeführt. Wir werden beweisen, dass die geradzahlig laminaren Familien von Hypergraphen ein Spezialfall eines Odd-Row-Sum-Packings sind. Anschließend werden wir die Wörter aus Kapitel 5.2 untersuchen und zeigen, dass für diese Wörter ein Odd-Row-Sum-Packing existert, deren Anzahl paarweise disjunkter Odd-Row-Sums gleich der Anzahl von Farbwechsel in der rekursiven Greedy-Färbung ist. Abschließend stellen wir die Ergebnisse der Analyse vor und beschreiben eine Vermutung, die den rekursiven Greedy-Algorithmus mit Hilfe von Odd-Row-Sum-Packings charakterisieren kann.

Zunächst führen wir einige Begrifflichkeiten ein und definieren das Odd-Row-Sum-Packing. Die folgenden Ausführungen und Definitionen orientieren sich an [7]. Wir betrachten die Intervall-Darstellung eines Wortes aus der Definition in Abschnitt 5.1.

Definition 6.1. *(Unabhängige Menge) [7]. Wir nennen eine Teilmenge paarweise disjunkter Intervalle $I(i,j) \in l(w)$ unabhänige Menge von $l(w)$.*

Proposition 6.2. *Eine maximale unabhängige Menge von $l(w)$ eines zulässigen Wortes w ist eine untere Schranke für die Anzahl der Farbwechsel.*

Beweis. Die Aussage ist evident. □

Im Folgenden verbessern wir diese untere Schranke.

Definition 6.3. *(Odd-Row-Sum) [7]. Sei $l(w)$ die Menge der Intervalle eines zulässigen Wortes w. Wir nennen die symmetrische Differenz $S := I(i,j)_1 \triangle I(i,j)_2 \triangle \ldots \triangle I(i,j)_{2k+1}$ mit $k \geq 0$ einer ungeraden Teilmenge von $l(w)$ eine Odd-Row-Sum.*

Definition 6.4. *(Odd-Row-Sum-Packing) [7]. Wir nennen eine Menge paarweise disjunkter Odd-Row-Sums $\{S_1, \ldots, S_l\}$ ein Odd-Row-Sum-Packing.*

Proposition 6.5. *([7]) Sei $(S_1, \ldots, S_l)$ ein Odd-Row-Sum-Packing eines Wortes w des Binary-Paint-Shop-Problems, dann benötigt eine zulässige Lösung des Binary-Paint-Shop-Problems mindestens l Farbwechsel.*

Beweis. Da die Odd-Row-Sums paarweise disjunkt sind, ist es ausreichend zu zeigen, dass es mindestens einen Farbwechsel in jeder Odd-Row-Sum $I(i,j)_1 \triangle I(i,j)_2 \triangle \ldots \triangle I(i,j)_{2k+1}$ gibt. Sei $K_B \in \mathbb{N}$ die Anzahl der Farbwechsel in einem Intervall $I(i,j)_B$. In einer zulässigen Lösung ist K_B für jeden Buchstaben $B \in \Sigma$ ungerade und da die Anzahl der Intervalle ungerade ist, ist $\sum_{B=1}^{2k+1} K_B$ ungerade. Es folgt die Behauptung.[7] □

Wir zeigen nun, dass die geradzahlig laminaren Familien von Hypergraphen ein einfacher Spezialfall eines Odd-Row-Sum-Packings sind.

Proposition 6.6 (7). *Sei $\mathcal{H}$ ein Hypergraph, der bezüglich einer guten Färbung eines zulässigen Wortes w des Binary-Paint-Shop-Problems definiert ist. Sei $\{I(i,j)_1, \ldots, I(i,j)_k\}$ eine Teilmenge der Menge der Intervalle $l(w)$ eines zulässigen Wortes w mit den folgenden Eigenschaften eines geradzahlig laminaren Hypergraphen $\mathcal{H}$.*

(i) Für alle $I(i,j)_A$ und $I(i,j)_B$ aus $\{I(i,j)_1, \ldots, I(i,j)_k\}$ gilt: $I(i,j)_A \subset I(i,j)_B$ oder $I(i,j)_B \subset I(i,j)_A$ oder $I(i,j)_A \cap I(i,j)_B = \emptyset$.

(ii) Für alle $I(i,j)_A \in \{I(i,j)_1, \ldots, I(i,j)_n\}$ gibt es eine gerade Anzahl von Teilmengen in $\{I(i,j)_1, \ldots, I(i,j)_n\}$, die in $I(i,j)_A$ enthalten und von $I(i,j)_A$ unterschiedlich sind.

Dann existiert für jedes zulässige Wort w des Binary-Paint-Shop-Problems, das solche Intervalle enthält, ein Odd-Row-Sum-Packing der Größe n und daraus folgend sind mindestens n Farbwechsel nötig.

Beweis. Sei $\mathcal{I} = \{I(i,j)_1, \ldots, I(i,j)_n\}$ eine Menge von Intervallen mit den Eigenschaften aus Proposition 5.25. Die geradzahlig laminaren Hypergraphen $\mathcal{H}$ besitzen, bezüglich der Inklusion, eine Baumstruktur. Es folgt mit (ii), dass jeder Knoten in diesem Baum eine gerade Anzahl von Kindern hat, das heißt jedes Intervall enthält eine gerade Anzahl von Intervallen.

Sind alle Intervalle $\mathcal{I} = \{I(i,j)_1, \ldots, I(i,j)_n\}$ paarweise disjunkt folgt die Aussage. Wir beweisen nun mit Induktion von n, dass eine Menge $\mathcal{I} = \{I(i,j)_1, \ldots, I(i,j)_n\}$ ein Odd-Row-Sum-Packing der Größe n besitzt, wenn es ein Intervall $I(i,j)_1 \in \mathcal{I} = \{I(i,j)_1, \ldots, I(i,j)_n\}$ gibt, das alle anderen Intervalle enthält. Es gilt $I(i,j)_l \subset I(i,j)_1$ für jedes $2 \leq l \leq n$. Sei nun $\mathcal{I}^* = \{I(i,j)_2, \ldots, I(i,j)_k\}$ die Menge der Intervalle, die unter dem Knoten $I(i,j)_1$ liegt. Die Menge der Intervalle in $\mathcal{I}^*$ ist nach Voraussetzung gerade. Da wir vorausgesetzt haben, dass jedes Intervall aus $\mathcal{I}$ in dem Intervall $I(i,j)_1$ enthalten ist, sind alle anderen Intervalle die nicht in $\mathcal{I}^*$ sind in einem Intervall aus $\mathcal{I}^*$ enthalten. Diese Intervalle in $\mathcal{I}^*$ enthalten wieder eine gerade Anzahl von Intervallen. Bilden wir nun für diese Intervalle aus $\mathcal{I}^*$ und deren Kinder $\mathcal{I}^{**}$ die symmetrische Differenz, dann erhalten wir für jedes dieser Intervalle mit der Induktionsannahme ein Odd-Row-Sum-Packing der Größe $|\mathcal{I}^{**}| + 1$. Da die Intervalle in $\mathcal{I}^*$ paarweise disjunkt sind, sind auch deren Odd-Row-Sums paarweise disjunkt. Wir erhalten damit ein Odd-Row-Sum-Packing der Größe $n - 1$. Da die Menge der Intervalle in $\mathcal{I}^*$ gerade ist, erhalten wir eine weitere Odd-Row-Sum die zu allen anderen Odd-Row-Sums paarweise disjunkt ist, wenn wir die symmetrische Differenz der Intervalle $I(i,j)_2 \triangle \ldots \triangle I(i,j)_k \triangle I(i,j)_1$ bilden. Dadurch erhalten wir ein Odd-Row-Sum-Packing der Größe n.
Gibt es kein Intervall, das alle anderen Intervalle enthält, dann gibt es mit dem bisher gezeigten eine Menge $\mathcal{I}^{**} = \{I(i,j)_1, \ldots, I(i,j)_k\}$, deren Intervalle alle Intervalle aus $\mathcal{I}$ enthalten, die nicht in $\mathcal{I}^{**}$ sind. Wenn wir erneut die

symmetrische Differenz dieser Intervalle mit der geraden Anzahl der darin enthaltenen Intervalle bilden, erhalten wir n paarweise disjunkte Odd-Row-Sums, da die Intervalle in $\mathcal{I}^{**}$ paarweise disjunkt sind. □

Wir untersuchen die Wörter aus Kapitel 5.2 und überprüfen, ob jedes dieser Wörter ein Odd-Row-Sum-Packing besitzt, das soviele paarweise disjunkte Odd-Row-Sums enthält, wie es Farbwechsel in der rekursiven Greedy-Färbung des Wortes gibt. Dazu bestimmen wir alle Odd-Row-Sums dieser Wörter. In der folgenden Tabelle zeigen wir die Ergebnisse der Analyse. Wir bezeichnen die Anzahl der Odd-Row-Sums im Odd-Row-Sum-Packing mit K_{ors} und die Anzahl der Farbwechsel der rekursiven Greedy-Färbung mit K_{rg}, wobei $K_{ors}, K_{rg} \in \mathbb{N}$ gilt.

Tabelle 6.15: Wort der Länge 10 und dessen rekursive Greedy-Färbung

Wort	$rg(w)$	K_{rg}
$(b, a, a, c, e, b, c, d, d, e)$	$(0, 0, 1, 1, 1, 1, 0, 0, 1, 0)$	4

Tabelle 6.16: Wort der Länge 10 und dessen Odd-Row-Sum-Packing

Wort	**Odd-Row-Sum-Packing**	K_{ors}
$(b, a, a, c, e, b, c, d, d, e)$	$\{\{1, 7\}, \{2, 3\}, \{8, 9\}, \{4, 10\}\}$	4

Tabelle 6.17: Wörter der Länge 12 und deren rekursive Greedy-Färbung

Wort	$rg(w)$	K_{rg}
$(b, c, a, a, d, f, b, c, d, e, e, f)$	$(0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0)$	4
$(b, d, a, a, c, b, c, e, d, f, e, f)$	$(0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0)$	4
$(b, a, d, a, c, f, b, c, e, d, e, f)$	$(0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0)$	4
$(b, a, d, a, c, f, b, c, d, e, e, f)$	$(0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0)$	4
$(b, a, a, d, c, f, b, c, e, d, e, f)$	$(0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0)$	4
$(b, a, a, d, c, f, b, c, d, e, e, f)$	$(0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0)$	4
$(b, a, a, c, f, d, b, c, d, e, e, f)$	$(0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0)$	4
$(b, a, a, c, d, f, b, c, d, e, e, f)$	$(0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0)$	4
$(f, b, a, a, c, e, b, c, d, d, e, f)$	$(0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1)$	5
$(b, a, a, c, f, e, b, c, d, d, e, f)$	$(0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0)$	4
$(b, a, a, c, e, f, b, c, d, d, e, f)$	$(0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0)$	4
$(b, a, a, c, e, b, c, d, d, e, f, f)$	$(0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1)$	5
$(b, a, a, c, f, b, c, d, e, d, e, f)$	$(0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0)$	4

Tabelle 6.18: Wörter der Länge 12 und deren Odd-Row-Sum-Packings

Wort	**Odd-Row-Sum-Packing**	K_{ors}
$(b,c,a,a,d,f,b,c,d,e,e,f)$	$\{\{3,4\},\{1,8\},\{2,9\},\{5,12\}\}$	4
$(b,d,a,a,c,b,c,e,d,f,e,f)$	$\{\{3,4\},\{5,6,7\},\{2,8,9\},\{1,12\}\}$	4
$(b,a,d,a,c,f,b,c,e,d,e,f)$	$\{\{2,3,4\},\{5,6,7,8\},\{9,10,11\},\{1,12\}\}$	4
$(b,a,d,a,c,f,b,c,d,e,e,f)$	$\{\{2,3,4\},\{1,8\},\{10,11\},\{5,9,12\}\}$	4
$(b,a,a,d,c,f,b,c,e,d,e,f)$	$\{\{2,3,4\},\{5,6,7,8\},\{10,11\},\{1,12\}\}$	4
$(b,a,a,d,c,f,b,c,d,e,e,f)$	$\{\{2,3\},\{5,6,7,8\},\{9,10,11\},\{1,12\}\}$	4
$(b,a,a,c,f,d,b,c,d,e,e,f)$	$\{\{2,3\},\{5,6,7,8\},\{10,11\},\{1,12\}\}$	4
$(b,a,a,c,d,f,b,c,d,e,e,f)$	$\{\{2,3\},\{6,7,8,9\},\{10,11\},\{5,12\}\}$	4
$(f,b,a,a,c,e,b,c,d,d,e,f)$	$\{\{3,4\},\{5,6,7,8\},\{9,10\},\{1,12\},\{2,11\}\}$	5
$(b,a,a,c,f,e,b,c,d,d,e,f)$	$\{\{2,3\},\{9,10\},\{1,8\},\{5,12\}\}$	4
$(b,a,a,c,e,f,b,c,d,d,e,f)$	$\{\{2,3\},\{9,10\},\{1,8\},\{5,12\}\}$	4
$(b,a,a,c,e,b,c,d,d,e,f,f)$	$\{\{2,3\},\{8,9\},\{11,12\},\{1,7\},\{4,10\}\}$	5
$(b,a,a,c,f,b,c,d,e,d,e,f)$	$\{\{2,3\},\{9,10,11\},\{1,7\},\{4,8,12\}\}$	4

Diese Ergebnisse führen zur Vermutung, dass eine Charakterisierung des rekursiven Greedy-Algorithmus mit Hilfe von Odd-Row-Sum-Packings möglich ist. Wir stellen folgende Hypothese auf.

Vermutung *Für jedes zulässige Wort w des Binary-Paint-Shop-Problems, das in allen Teilwörtern optimal gefärbt wird, existiert ein Odd-Row-Sum-Packing abhängig von einer guten rekursiven Greedy-Färbung.*

Unsere Idee, ein solches Odd-Row-Sum-Packing zu finden, ist die Folgende. Wir betrachten die Intervall-Darstellung eines Wortes w der Länge $2n$ in der Reihenfolge der rekursiven Greedy-Färbung. Das bedeutet, wir betrachten für alle n Schritte der rekursiven Greedy-Färbung jeweils das Intervall $I(i, j)$ und das Teilwort w_j bis zum zweiten Vorkommen, der in diesem Schritt zu färbenden Buchstaben. Tritt ein Farbwechsel auf, bilden wir eine symmetrische Differenz des betreffenden Intervalls $I(i, j)$ und einer geraden Anzahl von Intervallen, die in dem jeweiligen Teilwort w_j enthalten sind. Diese Odd-Row-Sums fügen wir in das Odd-Row-Sum-Packing des Wortes ein.

Wir präzisieren diese Idee, indem wir Regeln aufstellen, die beschreiben mit welchen Intervallen eine Odd-Row-Sum im Falle eines Farbwechsels gebildet wird. Wir betrachten die Länge des betreffenden Intervalls $I(i, j)$ im gesamten Wort w.

- Ist die Länge des Intervalls $I(i, j)$ zwei oder drei, dann fügen wir dieses Intervall $I(i, j)$ in das Odd-Row-Sum-Packing des Wortes w ein.
- Ist die Länge des Intervalls $I(i, j)$ größer drei, dann bilden wir eine symmetrische Differenz des betreffenden Intervalls mit zwei weiteren Intervallen, die in dem Teilwort w_j enthalten sind. Sind mehr als zwei weitere Intervalle in dem Teilwort w_j enthalten, wählen wir die beiden Intervalle nach folgenden Regeln.

  - Wir bevorzugen die Intervalle, deren zweites Vorkommen innerhalb des Intervalls $I(i, j)$ liegt. Dabei wählen wir die Intervalle, deren zweites Vorkommen möglichst nahe am zweiten Vorkommen j und deren erstes Vorkommen möglichst nahe am ersten Vorkommen i liegt.
  - Wir bevorzugen die Intervalle, die komplett innerhalb von $I(i, j)$ liegen.
  - Gibt es mehr als zwei Intervalle, die diese Eigenschaften erfüllen, werden die Intervalle gewählt, die bereits Teil des Odd-Row-Sum-Packings sind.
  - Gibt es nur ein Intervall, dessen zweites Vorkommen innerhalb des Intervalls $I(i, j)$ liegt, wählen wir als drittes Intervall das Intervall,

dessen zweites Vorkommen möglichst nahe am ersten Vorkommen i von $I(i,j)$ liegt.

Wir veranschaulichen diese Vorgehensweise am Beispiel des zulässigen Wortes $w = (b,a,a,c,e,b,c,d,d,e)$ mit dessen rekursiver Greedy-Färbung $rg = (0,0,1,1,1,1,0,0,1,0)$ aus Behauptung 22.

Beispiel 6.7. *Das erste Intervall, das in der rekursiven Greedy-Prozedur gefärbt wird, ist* $I(2,3)$ *des Buchstabens a. Da die erste Färbung* $(0,1)$ *ist, tritt ein Farbwechsel auf und nach den Regeln ist* $\{2,3\}$ *ein Teil des Odd-Row-Sum-Packings. Im nächsten Schritt wird der Buchstabe b gefärbt und wir erhalten die Färbung* $(0,0,1,1)$*, also keinen Farbwechsel. Durch die Färbung* $(0,0,1,1,1,0)$ *des Buchstabens c tritt erneut ein Farbwechsel auf. Wir betrachten das Teilwort* $w_7 = (b,a,a,c,e,b,c)$*. Die symmetrische Differenz* $I(4,7) \triangle I(1,6) \triangle I(2,3)$ *wird aus den drei enthaltenen Intervallen gebildet. Wir erhalten die Odd-Row-Sum* $\{1,7\}$ *und fügen diese zu dem Odd-Row-Sum-Packing des Wortes w hinzu. Anschließend wird das Paar des Buchstabens d gefärbt und dadurch ein Farbwechsel verursacht. Wir fügen* $\{8,9\}$ *in das Odd-Row-Sum-Packing ein. Zuletzt färben wir den Buchstaben e und erhalten wieder einen Farbwechsel. Wir betrachten nun das Teilwort* $w_{10} = w = (b,a,a,c,e,b,c,d,d,e)$*. Die zweiten Vorkommen der Buchstaben b, c und d liegen innerhalb des Intervalls* $I(5,10)$ *des Buchstabens e. Da das Intervall* $I(8,9)$ *von d komplett darin enthalten ist, wählen wir dieses Intervall für die symmetrische Differenz. Als drittes Intervall wählen wir das Intervall* $I(4,7)$ *des Buchstabens c, da das zweite Vorkommen des Buchstabens c näher als das von b am zweiten Vorkommen von e liegt. Wir bilden die symmetrische Differenz* $I(5,10) \triangle I(8,9) \triangle I(4,7)$ *und erhalten das Odd-Row-Sum-Packing* $\{\{2,3\},\{1,7\},\{8,9\},\{4,10\}\}$.

Wir wenden dieses Vorgehen auf die Wörter der Länge 12 aus Tabelle 6.18 an und finden jeweils, abhängig von der rekursiven Greedy-Färbung, ein Odd-Row-Sum-Packing. Wir stellen die Ergebnisse in der folgenden Tabelle dar.

Tabelle 6.19: Wörter der Länge 12 und deren von der rekursiven Greedy-Färbung abhängigen Odd-Row-Sum-Packings

Wort	**Odd-Row-Sum-Packing**	K_{ors}
$(b,c,a,a,d,f,b,c,d,e,e,f)$	$\{\{3,4\},\{1,9\},\{10,11\},\{5,12\}\}$	4
$(b,d,a,a,c,b,c,e,d,f,e,f)$	$\{\{3,4\},\{5,6,7\},\{2,8,9\},\{10,11,12\}\}$	4
$(b,a,d,a,c,f,b,c,e,d,e,f)$	$\{\{2,3,4\},\{1,8\},\{9,10,11\},\{5,12\}\}$	4
$(b,a,d,a,c,f,b,c,d,e,e,f)$	$\{\{2,3,4\},\{1,8\},\{10,11\},\{5,9,12\}\}$	4
$(b,a,a,d,c,f,b,c,e,d,e,f)$	$\{\{2,3\},\{1,4,8\},\{9,10,11\},\{5,12\}\}$	4
$(b,a,a,d,c,f,b,c,d,e,e,f)$	$\{\{2,3\},\{1,4,8\},\{10,11\},\{5,9,12\}\}$	4
$(b,a,a,c,f,d,b,c,d,e,e,f)$	$\{\{2,3\},\{1,8\},\{10,11\},\{5,12\}\}$	4
$(b,a,a,c,d,f,b,c,d,e,e,f)$	$\{\{2,3\},\{1,8\},\{10,11\},\{5,12\}\}$	4
$(f,b,a,a,c,e,b,c,d,d,e,f)$	$\{\{3,4\},\{2,8\},\{9,10\},\{5,11\},\{1,12\}\}$	5
$(b,a,a,c,f,e,b,c,d,d,e,f)$	$\{\{2,3\},\{1,8\},\{9,10\},\{4,5,11\}\}$	4
$(b,a,a,c,e,f,b,c,d,d,e,f)$	$\{\{2,3\},\{1,8\},\{9,10\},\{4,11\}\}$	4
$(b,a,a,c,e,b,c,d,d,e,f,f)$	$\{\{2,3\},\{1,7\},\{8,9\},\{4,10\},\{11,12\}\}$	5
$(b,a,a,c,f,b,c,d,e,d,e,f)$	$\{\{2,3\},\{1,7\},\{8,9,10\},\{4,11,12\}\}$	4

7 Zusammenfassung und Ausblick

In dieser Arbeit beschäftigten wir uns mit dem Binary-Paint-Shop-Problem und dessen Lösung mit Hilfe dreier Heuristiken. Im Kapitel 2 beschrieben wir eine Problematik der Automobilindustrie, die sich aus der steigenden Vielfalt angebotener Karosserievarianten und eine durch Kundenbestellungen getriggerte Fertigung für den Produktionsabschnitt Paint-Shop ergibt. Wir stellten das mathematische Modell dieses Problems, das sogenannte Paint-Shop-Problem und das daraus abgeleitete Binary-Paint-Shop-Problem, vor.

Im Kapitel 3 beschrieben wir drei Heuristiken und deren Vorgehensweise zur Lösung des Binary-Paint-Shop-Problems. Wir zeigten die Ergebnisse der Charakterisierung des Greedy-Algorithmus von *D. Rautenbach* und *Z. Szigeti* in Satz 3.7 und charakterisierten den Red-First-Algorithmus in Satz 3.9. Außerdem stellten wir Ergebnisse von *S.D. Andres* und *W. Hochstättler* vor, die vermuten lassen, dass der rekursive Greedy-Algorithmus um einige Klassen besser ist, als die beiden anderen Algorithmen.

Dies führte uns zur Frage, ob wir den rekursiven Greedy-Algorithmus mit einer ähnlichen Aussage wie Satz 3.7, der den Greedy-Algorithmus beschreibt, charakterisieren können. Im ersten Schritt analysierten wir im Kapitel 4 die rekursiven Greedy-Färbungen der zulässigen Wörter des Binary-Paint-Shop-Problems bis Länge 14. Als Ergebnis erhielten wir insgesamt 16 Wörter, deren rekursive Greedy-Färbung für die jeweiligen Wortlängen nicht optimal ist.

Mit diesen Ergebnissen setzten wir die Untersuchung des rekursiven Greedy-Algorithmus im Kapitel 5 fort. Motiviert durch die Ergebnisse aus Kapitel 4, verfolgten wir den Ansatz die Beweisstruktur der Charakterisierung des Greedy-Algorithmus auf den rekursiven Greedy-Algorithmus zu adaptieren. Dazu untersuchten wir, ob es eine Definition für einen Hypergraphen bezüglich der rekursiven Greedy-Färbung gibt, sodass der Hypergraph für Wörter, die in allen Teilwörtern optimal gefärbt werden, geradzahlig laminar ist. Nach einigen negativen Ergebnissen konnten wir abschließend zeigen, dass es Wörter gibt, deren rekursive Greedy-Färbung in allen Teilwörtern optimal ist, aber für die es nicht möglich ist einen geradzahlig laminaren Hypergraph, bezüglich der rekursiven Greedy-Färbung, zu definieren. Damit mussten wir den Ansatz, die Beweisstruktur der Charakterisierung des Greedy-Algorithmus auf den rekursiven Greedy-Algorithmus anzuwenden, verwerfen.

Die geradzahlig laminaren Hypergraphen sind ein Spezialfall eines sogenannten Odd-Row-Sum-Packings. Im Kapitel 6 führten wir diesen Begriff ein. Wir untersuchten zuerst, ob wir für Wörter, für die es keinen geradzah-

lig laminaren Hypergraph bezüglich der rekursiven Greedy-Färbung gibt, ein Odd-Row-Sum-Packing finden. Da wir hier zu einem positiven Ergebnis kamen, lies uns dies vermuten, dass eine Charakterisierung des rekursiven Greedy-Algorithmus mit Hilfe von Odd-Row-Sum-Packings möglich ist. Wir formulierten die Hypothese, dass wir zu jedem Wort des Binary-Paint-Shop-Problems, das in allen Teilwörtern optimal gefärbt wird, ein Odd-Row-Sum-Packing, abhängig von der rekursiven Greedy-Färbung, finden können. Wir präzisierten die Idee zur Erzeugung eines solchen Odd-Row-Sum-Packings und konnten zeigen, dass wir für alle Wörter aus Kapitel 5 ein Odd-Row-Sum-Packing abhängig von der rekursiven Greedy-Färbung finden. Durch diese positiven Ergebnisse konnten wir die Vermutung bestätigen. Wir beenden die Arbeit mit den folgenden offenen Fragen:

- Die Analyse der rekursiven Greedy-Färbungen für Wörter der Länge 16 ist mit unserer entwickelten Methode nicht möglich, da zu viele Wörter manuell ausgewertet werden müssen. Wie können wir diese Methode verbessern, um die rekursiven Greedy-Färbungen aller Wörter des Binary-Paint-Shop-Problems der Wortlänge 16 zu analysieren?

- Finden wir für jedes Wort des Binary-Paint-Shop-Problems, das in allen Teilwörtern optimal gefärbt wird, ein Odd-Row-Sum-Packing bezüglich der rekursiven Greedy-Färbung?

- Wie können wir beweisen, dass ein in allen Teilwörtern optimal gefärbtes zulässiges Wort w ein, von der rekursiven Greedy-Färbung abhängiges Odd-Row-Sum-Packing, besitzt?

- Können wir zeigen, dass jedes verbotene Wort kein solches Odd-Row-Sum-Packing besitzt?

A Anhang

A.1 Quelltext Analyse-Programm

Im Folgenden wird der Quelltext des Analyse-Programms am Beispiel der Wortlänge zehn dargestellt. Um das Programm für andere Wortlängen zu verwenden, müssen die entsprechenden Zeilen kommentiert beziehungsweise auskommentiert oder gegebenenfalls angepasst werden.

```
1  import java.util.ArrayList;
2
3  import java.util.List;
4  import java.io.*;
5
6  public class recursive_greedy_code
7  {
8  static ArrayList<String> kombinationen = new ArrayList();
9  static ArrayList<String> erste_faerbung = new ArrayList();
10 static ArrayList<Integer> indexliste = new ArrayList();
11 static ArrayList<String> faerbung = new ArrayList();
12 static ArrayList<Integer> farbwechsel = new ArrayList();
13 static ArrayList<Integer> farbwechsel_real = new ArrayList()
       ;
14 static ArrayList<Integer> farbwechsel_min = new ArrayList();
15 static ArrayList<Integer> farbwechsel_min_rueck = new
       ArrayList();
16 static ArrayList<String> verbotene_woerter = new ArrayList()
       ;
17 static ArrayList<String> verbotene_woerter_neu = new
       ArrayList();
18 static ArrayList<String> kombinationen_input = new ArrayList
       ();
19 static ArrayList<String> verbotene_wortliste_rueckwaerts =
       new ArrayList();
20
21 static void kombinationen (String s, int index_buchstabe)
22 {
23 StringBuffer temp_komb = new StringBuffer();
24 temp_komb.append(new String());
25 String buchstaben = "ABCDEFG";
26
27 temp_komb.append(buchstaben.charAt(index_buchstabe));
28 temp_komb.insert(0, buchstaben.charAt(index_buchstabe));
29
30 for(int a = 0; a < 3; a++)
31 {
32 if (temp_komb.length() == 2)
```

```
33 {
34 temp_komb.append(buchstaben.charAt(index_buchstabe+1));
35 temp_komb.insert(0, buchstaben.charAt(index_buchstabe+1));
36 }
37 else
38 {
39 int index_buchstabe2 = temp_komb.indexOf("B");
40 temp_komb.deleteCharAt(index_buchstabe2);
41 temp_komb.insert(a, "B");
42 }
43 for(int b = 0; b < 5; b++)
44 {
45 if(temp_komb.length() == 4)
46 {
47 temp_komb.append(buchstaben.charAt(index_buchstabe+2));
48 temp_komb.insert(0, buchstaben.charAt(index_buchstabe+2));
49 }
50 else
51 {
52 int index_buchstabe3 = temp_komb.indexOf("C");
53 temp_komb.deleteCharAt(index_buchstabe3);
54 temp_komb.insert(b, "C");
55 }
56 for(int c = 0; c < 7; c++)
57 {
58 if(temp_komb.length() == 6)
59 {
60 temp_komb.append(buchstaben.charAt(index_buchstabe+3));
61 temp_komb.insert(0, buchstaben.charAt(index_buchstabe+3));
62 }
63 else
64 {
65 int index_buchstabe4 = temp_komb.indexOf("D");
66 temp_komb.deleteCharAt(index_buchstabe4);
67 temp_komb.insert(c, "D");
68 }
69 for(int d = 0; d < 9; d++)
70 {
71 if(temp_komb.length() == 8)
72 {
73 temp_komb.append(buchstaben.charAt(index_buchstabe+4));
74 temp_komb.insert(0, buchstaben.charAt(index_buchstabe+4));
75 s = temp_komb.toString();
76 kombinationen.add(s);
77 }
78 else
79 {
80 int index_buchstabe5 = temp_komb.indexOf("E");
81 temp_komb.deleteCharAt(index_buchstabe5);
```

```
82  temp_komb.insert(d, "E");
83  s = temp_komb.toString();
84  kombinationen.add(s);
85  }}}}}}
86
87  static void w_strich(String w_work, String w_ganz, int
        laenge_w)
88  {
89  StringBuffer w_original = new StringBuffer();
90  w_original.append(new String(w_ganz));
91
92  if(laenge_w == 2)
93  {
94  erste_faerbung.add("01");
95  }
96  else
97  {
98  String lastZ = new String(w_work.substring(laenge_w-1));
99  StringBuffer w_strichb = new StringBuffer();
100 w_strichb.append(new String(w_work));
101 int index_firstZ = w_ganz.indexOf(lastZ);
102
103 indexliste.add(w_strichb.indexOf(lastZ));
104 int index_deleteZ = w_strichb.indexOf(lastZ);
105
106 w_strichb.deleteCharAt(w_strichb.length()-1);
107 w_strichb.deleteCharAt(index_deleteZ);
108
109 String w_strich;
110 w_strich = w_strichb.toString();
111 w_strich(w_strich, w_ganz, w_strich.length());
112 }
113 }
114
115 static void farbwechsel_minimal(String wort_w, String
        richtung)
116 {
117 StringBuffer w_farbwechsel_ber = new StringBuffer();
118 w_farbwechsel_ber.append(wort_w);
119 int farbwechsel_anz_ber = 0;
120
121 int laenge_w_ber = w_farbwechsel_ber.length();
122
123 StringBuffer farbwechsel_temp_buffer = new StringBuffer();
124 int g = 0;
125
126 while(g < laenge_w_ber)
127 {
```

```
128 if((g < laenge_w_ber -1) && (w_farbwechsel_ber.charAt(g) ==
        w_farbwechsel_ber.charAt(g+1)))
129 {
130 if((g > 0) && (g < laenge_w_ber -3) && (w_farbwechsel_ber.
        charAt(g-1) == w_farbwechsel_ber.charAt(g+2)))
131 {
132 farbwechsel_anz_ber = farbwechsel_anz_ber +1;
133 farbwechsel_temp_buffer.replace(g-1, g, "4");
134 farbwechsel_temp_buffer.insert(g, '4');
135 farbwechsel_temp_buffer.insert(g+1, '4');
136 farbwechsel_temp_buffer.insert(g+2, '4');
137 g = g+3;
138 }
139 else
140 {
141 farbwechsel_anz_ber = farbwechsel_anz_ber +1;
142 farbwechsel_temp_buffer.insert(g, '2');
143 farbwechsel_temp_buffer.insert(g+1, '2');
144 g = g+2;
145 }}
146 else
147 {
148 if((g < laenge_w_ber -3) && (w_farbwechsel_ber.charAt(g) ==
        w_farbwechsel_ber.charAt(g+2)) && (w_farbwechsel_ber.
        charAt(g+1) == w_farbwechsel_ber.charAt(g+3)))
149 {
150 farbwechsel_anz_ber = farbwechsel_anz_ber +1;
151 farbwechsel_temp_buffer.insert(g, '4');
152 farbwechsel_temp_buffer.insert(g+1, '4');
153 farbwechsel_temp_buffer.insert(g+2, '4');
154 farbwechsel_temp_buffer.insert(g+3, '4');
155 g = g+4;
156 }
157 else
158 if((g < laenge_w_ber -2) && (w_farbwechsel_ber.charAt(g) ==
        w_farbwechsel_ber.charAt(g+2)))
159 {
160 farbwechsel_anz_ber = farbwechsel_anz_ber +1;
161 farbwechsel_temp_buffer.insert(g, '3');
162 farbwechsel_temp_buffer.insert(g+1, '3');
163 farbwechsel_temp_buffer.insert(g+2, '3');
164 g = g+3;
165 }
166 else
167 {
168 if((g < laenge_w_ber -3) && (w_farbwechsel_ber.charAt(g) ==
        w_farbwechsel_ber.charAt(g+3)) &&
169 (w_farbwechsel_ber.charAt(g+1) != w_farbwechsel_ber.charAt(g
        +2)))
```

```
170 {
171 farbwechsel_anz_ber = farbwechsel_anz_ber +1;
172 farbwechsel_temp_buffer.insert(g, '4');
173 farbwechsel_temp_buffer.insert(g+1, '4');
174 farbwechsel_temp_buffer.insert(g+2, '4');
175 farbwechsel_temp_buffer.insert(g+3, '4');
176 g = g+4;
177 }
178 else
179 {
180 if((g < laenge_w_ber -4) && (w_farbwechsel_ber.charAt(g) ==
        w_farbwechsel_ber.charAt(g+4)) &&
181 (w_farbwechsel_ber.charAt(g+1) != w_farbwechsel_ber.charAt(g
        +2)) && (w_farbwechsel_ber.charAt(g+2) !=
        w_farbwechsel_ber.charAt(g+3))
182 && (w_farbwechsel_ber.charAt(g+1) != w_farbwechsel_ber.
        charAt(g+3)))
183 {
184 farbwechsel_anz_ber = farbwechsel_anz_ber +1;
185 farbwechsel_temp_buffer.insert(g, '5');
186 farbwechsel_temp_buffer.insert(g+1, '5');
187 farbwechsel_temp_buffer.insert(g+2, '5');
188 farbwechsel_temp_buffer.insert(g+3, '5');
189 farbwechsel_temp_buffer.insert(g+4, '5');
190 g = g+5;
191 }
192 /*else
193 {
194 if((g < laenge_w_ber -5) && (w_farbwechsel_ber.charAt(g) ==
        w_farbwechsel_ber.charAt(g+5)) &&
195 (w_farbwechsel_ber.charAt(g+1) != w_farbwechsel_ber.charAt(g
        +2)) && (w_farbwechsel_ber.charAt(g+2) !=
        w_farbwechsel_ber.charAt(g+3))
196 && (w_farbwechsel_ber.charAt(g+3) != w_farbwechsel_ber.
        charAt(g+4)) && (w_farbwechsel_ber.charAt(g+2) !=
        w_farbwechsel_ber.charAt(g+4))
197 && (w_farbwechsel_ber.charAt(g+1) != w_farbwechsel_ber.
        charAt(g+4)) && (w_farbwechsel_ber.charAt(g+1) !=
        w_farbwechsel_ber.charAt(g+3)))
198 {
199 farbwechsel_anz_ber = farbwechsel_anz_ber +1;
200
201 farbwechsel_temp_buffer.insert(g, '6');
202 farbwechsel_temp_buffer.insert(g+1, '6');
203 farbwechsel_temp_buffer.insert(g+2, '6');
204 farbwechsel_temp_buffer.insert(g+3, '6');
205 farbwechsel_temp_buffer.insert(g+4, '6');
206 farbwechsel_temp_buffer.insert(g+5, '6');
207 g = g+6;
```

```
208 }
209 else{
210 if((g < laenge_w_ber -6) && (w_farbwechsel_ber.charAt(g) ==
        w_farbwechsel_ber.charAt(g+6)) &&
211 (w_farbwechsel_ber.charAt(g+1) != w_farbwechsel_ber.charAt(g
        +2)) && (w_farbwechsel_ber.charAt(g+2) !=
        w_farbwechsel_ber.charAt(g+3))
212 && (w_farbwechsel_ber.charAt(g+3) != w_farbwechsel_ber.
        charAt(g+4)) && (w_farbwechsel_ber.charAt(g+4) !=
        w_farbwechsel_ber.charAt(g+5))
213 && (w_farbwechsel_ber.charAt(g+1) != w_farbwechsel_ber.
        charAt(g+3)) && (w_farbwechsel_ber.charAt(g+1) !=
        w_farbwechsel_ber.charAt(g+4))
214 && (w_farbwechsel_ber.charAt(g+1) != w_farbwechsel_ber.
        charAt(g+5)) && (w_farbwechsel_ber.charAt(g+2) !=
        w_farbwechsel_ber.charAt(g+4))
215 && (w_farbwechsel_ber.charAt(g+2) != w_farbwechsel_ber.
        charAt(g+5)) && (w_farbwechsel_ber.charAt(g+3) !=
        w_farbwechsel_ber.charAt(g+5)))
216 {
217 farbwechsel_anz_ber = farbwechsel_anz_ber +1;
218
219 farbwechsel_temp_buffer.insert(g, '7');
220 farbwechsel_temp_buffer.insert(g+1, '7');
221 farbwechsel_temp_buffer.insert(g+2, '7');
222 farbwechsel_temp_buffer.insert(g+3, '7');
223 farbwechsel_temp_buffer.insert(g+4, '7');
224 farbwechsel_temp_buffer.insert(g+5, '7');
225 farbwechsel_temp_buffer.insert(g+6, '7');
226
227
228 g = g+7;
229 }*/
230 else
231 {
232 farbwechsel_temp_buffer.insert(g, '0');
233 g = g+1;
234 }}}}}
235
236 String farbwechselfolge_temp = farbwechsel_temp_buffer.
        toString();
237
238 int farbwechsel_anz_temp = 0;
239 int u = 0;
240 boolean zusaetzlicher_wechsel = false;
241 boolean drei_farbwechsel = false;
242
243 List<Integer> marker = new ArrayList<Integer>();
244
```

```
245 while( u < farbwechselfolge_temp.length())
246 {
247 int farbwechsel_marker = farbwechselfolge_temp.charAt(u);
248 if(farbwechsel_marker == '0')
249 {
250 u = u+1;
251 if (zusaetzlicher_wechsel == true)
252 {
253 farbwechsel_anz_temp = farbwechsel_anz_temp -
        farbwechsel_anz_temp ;
254 zusaetzlicher_wechsel = false;
255 }
256 }
257
258 if(farbwechsel_marker == '2')
259 {
260 farbwechsel_anz_temp = farbwechsel_anz_temp +1;
261 if(farbwechsel_anz_temp %2 == 1)
262 {
263 marker.add(u);
264 }
265 else
266 {
267 marker.add(u+2);
268 }
269 u = u+2;
270 }
271 if(farbwechsel_marker == '3')
272 {
273 farbwechsel_anz_temp = farbwechsel_anz_temp +1;
274 if(farbwechsel_anz_temp %2 == 1)
275 {
276 marker.add(u);
277 }
278 else
279 {
280 marker.add(u+3);
281 }
282 u = u+3;
283 }
284 if(farbwechsel_marker == '4')
285 {
286 farbwechsel_anz_temp = farbwechsel_anz_temp +1;
287 if(farbwechsel_anz_temp %2 == 1)
288 {
289 marker.add(u);
290 }
291 else
292 {
```

```
293 marker.add(u+4);
294 }
295 u = u+4;
296 }
297 if(farbwechsel_marker == '5')
298 {
299 farbwechsel_anz_temp = farbwechsel_anz_temp +1;
300 if(farbwechsel_anz_temp %2 == 1)
301 {
302 marker.add(u);
303 }
304 else
305 {
306 marker.add(u+5);
307 }
308 u = u+5;
309 }
310 if(farbwechsel_marker == '6')
311 {
312 farbwechsel_anz_temp = farbwechsel_anz_temp +1;
313 if(farbwechsel_anz_temp %2 == 1)
314 {
315 marker.add(u);
316 }
317 else
318 {
319 marker.add(u+6);
320 }
321 u = u+6;
322 }
323 if(farbwechsel_marker == '7')
324 {
325 farbwechsel_anz_temp = farbwechsel_anz_temp +1;
326 if(farbwechsel_anz_temp %2 == 1)
327 {
328 marker.add(u);
329 }
330 else
331 {
332 marker.add(u+7);
333 }
334 u = u+7;
335 }
336 else;
337
338 boolean gleiche_buchstaben = false;
339 int b1 = 20;
340 int b2 = 22;
341
```

```
342 if ((farbwechsel_anz_temp == 2)||(farbwechsel_anz_temp == 4)
    ||(farbwechsel_anz_temp == 6) )
343 {
344 if ((farbwechsel_anz_temp == 4))
345 {
346 if(((marker.get(0) > 0) && (marker.get(marker.size()-1) <=
    farbwechselfolge_temp.length()-1)) &&
347 (farbwechselfolge_temp.charAt(marker.get(0)-1) == '0') && (
    farbwechselfolge_temp.charAt(marker.get(3)) == '0')
348 && (farbwechselfolge_temp.charAt(marker.get(1)) != '0') && (
    farbwechselfolge_temp.charAt(marker.get(2)) != '0'))
349 {
350 for(int h = farbwechselfolge_temp.indexOf("0"); h < (
    w_farbwechsel_ber.substring(farbwechselfolge_temp.
    indexOf("0"), marker.get(0)).length()); h++)
351 {
352 for(int j = marker.get(marker.size()-1); j <
    farbwechselfolge_temp.lastIndexOf("0")+1; j++)
353 {
354 if(w_farbwechsel_ber.charAt(h) == w_farbwechsel_ber.charAt(j
    ))
355 {
356 gleiche_buchstaben = true;
357 }}}}
358 else
359 {
360 if(((marker.get(marker.size()-1) <= farbwechselfolge_temp.
    length()-1)) && ((farbwechselfolge_temp.charAt(marker.
    get(2)-1) == '0') && (farbwechselfolge_temp.charAt(
    marker.get(3)) == '0')))
361 {
362 for(int i = marker.get(1); i < marker.get(2); i++)
363 {
364 for(int k = marker.get(3); k < farbwechselfolge_temp.
    lastIndexOf("0")+1; k++)
365 {
366 if(w_farbwechsel_ber.charAt(i) == w_farbwechsel_ber.charAt(k
    ))
367 {
368 gleiche_buchstaben = true;
369 }
370 }}}}}
371 else
372 {
373 if((marker.get(marker.size()-2) > 0) && (marker.get(marker.
    size()-1)<= farbwechselfolge_temp.length()-1) )
374 {
375 if((farbwechselfolge_temp.charAt(marker.get(marker.size()-2)
    -1) == '0') && (farbwechselfolge_temp.charAt(marker.get(
```

```
        marker.size()-1)) == '0'))
376     {
377     if (drei_farbwechsel == false)
378     {
379     for(int y1 = farbwechselfolge_temp.indexOf("0"); y1 < marker
            .get(marker.size()-2); y1++)
380     {
381     if(farbwechselfolge_temp.charAt(y1) == '0')
382     {
383     int v1 = marker.get(marker.size()-1);
384     while((v1 < 10) && (farbwechselfolge_temp.charAt(v1) == '0')
            )
385     {
386     if(w_farbwechsel_ber.charAt(y1) == w_farbwechsel_ber.charAt(
            v1))
387     {
388     gleiche_buchstaben = true;
389     b1 = y1;
390     b2 = v1;
391     }
392     else;
393     v1 = v1 +1;
394     }}}}
395     else
396     {
397     for(int y2 = marker.get(marker.size()-3); y2 < marker.get(
            marker.size()-2); y2++)
398     {
399     if(farbwechselfolge_temp.charAt(y2) == '0')
400     {
401     int v2 = marker.get(marker.size()-1);
402
403     while((v2 < 10) && (farbwechselfolge_temp.charAt(v2) == '0')
            && (gleiche_buchstaben != true))
404     {
405
406     if(w_farbwechsel_ber.charAt(y2) == w_farbwechsel_ber.charAt(
            v2))
407     {
408     gleiche_buchstaben = true;
409     b1 = y2;
410     b2 = v2;
411     }
412     else
413     v2 = v2 + 1;
414     }
415     }
416     }
```

```
417 if((gleiche_buchstaben == false) && (drei_farbwechsel ==
        false))
418 {
419 for(int y3 = 0; y3 < marker.get(marker.size()-1); y3++)
420 {
421 if(farbwechselfolge_temp.charAt(y3) == '0')
422 {
423 int v3 = marker.get(marker.size()-1);
424 while((v3 < 10) && (farbwechselfolge_temp.charAt(v3) == '0')
        && (gleiche_buchstaben != true))
425 {
426 if(w_farbwechsel_ber.charAt(y3) == w_farbwechsel_ber.charAt(
        v3))
427 {
428 gleiche_buchstaben = true;
429 b1 = y3;
430 b2 = v3;
431 }
432 else
433 v3 = v3 + 1;
434 }}
435 }}}}}
436 else;}
437 if((farbwechsel_anz_temp == 2) && ( gleiche_buchstaben ==
        false))
438 {
439 farbwechsel_anz_temp = farbwechsel_anz_temp -1;
440 drei_farbwechsel = true;
441 switch (farbwechsel_marker)
442 {
443 case '2':
444 int temp_marker1 = marker.get(marker.size()-1);
445 marker.remove(marker.size()-1);
446 marker.add(temp_marker1 -2);
447 break;
448 case '3':
449 int temp_marker2 = marker.get(marker.size()-1);
450 marker.remove(marker.size()-1);
451 marker.add(temp_marker2 -3);
452 break;
453 case '4':
454 int temp_marker3 = marker.get(marker.size()-1);
455 marker.remove(marker.size()-1);
456 marker.add(temp_marker3 -4);
457 break;
458 case '5':
459 int temp_marker4 = marker.get(marker.size()-1);
460 marker.remove(marker.size()-1);
461 marker.add(temp_marker4 -5);
```

```
462 break;
463 case '6':
464 int temp_marker5 = marker.get(marker.size()-1);
465 marker.remove(marker.size()-1);
466 marker.add(temp_marker5 -6);
467 break;
468 case '7':
469 int temp_marker6 = marker.get(marker.size()-1);
470 marker.remove(marker.size()-1);
471 marker.add(temp_marker6 -7);
472 break;
473 }
474 }
475 if (gleiche_buchstaben == true)
476 {
477 farbwechsel_anz_ber = farbwechsel_anz_ber +1;
478 zusaetzlicher_wechsel = true;
479
480 }
481 if ((farbwechsel_anz_ber == 4) && (farbwechselfolge_temp.
      charAt(0) == '0') && (farbwechselfolge_temp.charAt(
      farbwechselfolge_temp.length() -1) == '0')
482 && (w_farbwechsel_ber.charAt(0) == w_farbwechsel_ber.charAt(
      farbwechselfolge_temp.length() -1))&& (b1 != 0) && (b2
      != 0)&& (b1 != (farbwechselfolge_temp.length() -1)) && (
      b2 != (farbwechselfolge_temp.length() -1)))
483 {
484 farbwechsel_anz_ber = farbwechsel_anz_ber +1;
485 }
486 else;
487 {
488 if ((farbwechsel_anz_ber == 4) && (farbwechselfolge_temp.
      charAt(0) == '0') &&(farbwechselfolge_temp.charAt(1) ==
      '0') && (farbwechselfolge_temp.charAt(
      farbwechselfolge_temp.length() -1) == '0')
489 && (w_farbwechsel_ber.charAt(1) == w_farbwechsel_ber.charAt(
      farbwechselfolge_temp.length() -1)) && (b1 != 0) && (b2
      != 0)&& (b1 != (farbwechselfolge_temp.length() -1)) && (
      b2 != (farbwechselfolge_temp.length() -1))&& (b1 != 1)
      && (b2 != 1))
490 {
491 farbwechsel_anz_ber = farbwechsel_anz_ber +1;
492 }
493 else
494 {
495 if ((farbwechsel_anz_ber == 4) && (farbwechselfolge_temp.
      charAt(0) == '0') &&(farbwechselfolge_temp.charAt(1) ==
      '0') && (farbwechselfolge_temp.charAt(2) == '0') && (
      farbwechselfolge_temp.charAt(farbwechselfolge_temp.
```

```
        length() -1) == '0')
496 && (w_farbwechsel_ber.charAt(2) == w_farbwechsel_ber.charAt(
        farbwechselfolge_temp.length() -1)) && (b1 != 0) && (b2
        != 0) && (b1 != 1) && (b2 != 1)&& (b1 != 2) && (b2 != 2)
         && (b1 != (farbwechselfolge_temp.length() -1)) && (b2
        != (farbwechselfolge_temp.length() -1)))
497 {
498 farbwechsel_anz_ber = farbwechsel_anz_ber +1;
499 }
500 else
501 {
502 if ((farbwechsel_anz_ber == 4) && (farbwechselfolge_temp.
        charAt(0) == '0') &&(farbwechselfolge_temp.charAt(
        farbwechselfolge_temp.length() -1) == '0') && (
        farbwechselfolge_temp.charAt(farbwechselfolge_temp.
        length() -2) == '0')
503 && (w_farbwechsel_ber.charAt(0) == w_farbwechsel_ber.charAt(
        farbwechselfolge_temp.length() -2))&& (b1 != (
        farbwechselfolge_temp.length() -1)) && (b2 != (
        farbwechselfolge_temp.length() -1)) && (b1 != 0) && (b2
        != 0)
504 && (b1 != (farbwechselfolge_temp.length() -2)) && (b2 != (
        farbwechselfolge_temp.length() -2)))
505 {
506 farbwechsel_anz_ber = farbwechsel_anz_ber +1;
507 }
508 else
509 {
510 if ((farbwechsel_anz_ber == 4) && (farbwechselfolge_temp.
        charAt(0) == '0') &&(farbwechselfolge_temp.charAt(
        farbwechselfolge_temp.length() -1) == '0') && (
        farbwechselfolge_temp.charAt(farbwechselfolge_temp.
        length() -2) == '0') && (farbwechselfolge_temp.charAt(
        farbwechselfolge_temp.length() -3) == '0')
511 && (w_farbwechsel_ber.charAt(0) == w_farbwechsel_ber.charAt(
        farbwechselfolge_temp.length() -3))&& (b1 != (
        farbwechselfolge_temp.length() -1)) && (b2 != (
        farbwechselfolge_temp.length() -1)) && (b1 != 0) && (b2
        != 0)
512 && (b1 != (farbwechselfolge_temp.length() -2)) && (b2 != (
        farbwechselfolge_temp.length() -2))&& (b1 != (
        farbwechselfolge_temp.length() -3)) && (b2 != (
        farbwechselfolge_temp.length() -3)))
513 {
514 farbwechsel_anz_ber = farbwechsel_anz_ber +1;
515 }
516 else
517 {
```

```
518 if ((farbwechsel_anz_ber == 4) && (farbwechselfolge_temp.
    charAt(0) == '0') &&(farbwechselfolge_temp.charAt(1) ==
    '0') && (farbwechselfolge_temp.charAt(
    farbwechselfolge_temp.length() -2) == '0') && (
    farbwechselfolge_temp.charAt(farbwechselfolge_temp.
    length() -1) == '0')
519 && (b1 != (farbwechselfolge_temp.length() -1)) && (b2 != (
    farbwechselfolge_temp.length() -1)) && (b1 != 0) && (b2
    != 0)
520 && (b1 != (farbwechselfolge_temp.length() -2)) && (b2 != (
    farbwechselfolge_temp.length() -2))&& (b1 != 1) && (b2
    != 1)  )
521 {
522 if((w_farbwechsel_ber.charAt(0) == w_farbwechsel_ber.charAt(
    farbwechselfolge_temp.length() -2)) || (
    w_farbwechsel_ber.charAt(0) == w_farbwechsel_ber.charAt(
    farbwechselfolge_temp.length() -1))
523 || (w_farbwechsel_ber.charAt(1) == w_farbwechsel_ber.charAt(
    farbwechselfolge_temp.length() -2)) || (
    w_farbwechsel_ber.charAt(1) == w_farbwechsel_ber.charAt(
    farbwechselfolge_temp.length() -1)))
524 {farbwechsel_anz_ber = farbwechsel_anz_ber +1;}
525 else;}
526 else;}}}}}}}
527 if (richtung == "vorwaerts")
528 {
529 farbwechsel_min.add(farbwechsel_anz_ber);
530 }
531 if (richtung == "rueckwaerts")
532 {
533 farbwechsel_min_rueck.add(farbwechsel_anz_ber);
534 }
535 }
536
537 static void verbotene_woerter(ArrayList<Integer>
    farbwechsel_real, ArrayList<Integer> farbwechsel_min )
538 {
539 for(int p = 0; p < kombinationen.size(); p++)
540 {
541 int temp1 = (farbwechsel_real.get(p));
542 int temp2 = (farbwechsel_min.get(p));
543 if (temp1 > temp2)
544 {
545 verbotene_woerter.add(kombinationen.get(p));
546 }
547 else;
548 }}
549 static void bekannte_verbotene_woerter(ArrayList<String>
    verbotene_woerter_temp)
```

```
550 {
551 String buchstaben = "ABCDEFG";
552 for(int q = 0; q < verbotene_woerter_temp.size(); q++)
553 {
554 String verbotenes_wort = verbotene_woerter_temp.get(q);
555 int laenge_verbotenes_wort = verbotenes_wort.length();
556 boolean enthalten = false;
557 for(int r = 0; r < (laenge_verbotenes_wort/2); r++)
558 {
559 StringBuffer wort_minus_buchstabe = new StringBuffer();
560 wort_minus_buchstabe.append(verbotenes_wort);
561 if (enthalten == false)
562 {
563 for(int t = 0; t < wort_minus_buchstabe.length(); t++)
564 {
565 if(wort_minus_buchstabe.charAt(t) == buchstaben.charAt(r))
566 {
567 if ( (t < wort_minus_buchstabe.length()-1) && (
    wort_minus_buchstabe.charAt(t) == wort_minus_buchstabe.
    charAt(t+1)))
568 {
569 wort_minus_buchstabe.deleteCharAt(t);
570 wort_minus_buchstabe.deleteCharAt(t);
571 }
572 else
573 {
574 wort_minus_buchstabe.deleteCharAt(t);
575 }}}
576 // "ADDBCCAB" vorwaerts
577 if ((wort_minus_buchstabe.charAt(0)== wort_minus_buchstabe.
    charAt(6)) && (wort_minus_buchstabe.charAt(1)==
    wort_minus_buchstabe.charAt(2))
578         && (wort_minus_buchstabe.charAt(3)==
                wort_minus_buchstabe.charAt(7)) && (
                wort_minus_buchstabe.charAt(4)==
                wort_minus_buchstabe.charAt(5)))
579 {
580 enthalten = true;
581 }
582 else
583 {// "ADDBCCAB" rueckwaerts
584 if ((wort_minus_buchstabe.charAt(0)== wort_minus_buchstabe.
    charAt(4)) && (wort_minus_buchstabe.charAt(1)==
    wort_minus_buchstabe.charAt(7))
585         && (wort_minus_buchstabe.charAt(2)==
                wort_minus_buchstabe.charAt(3)) && (
                wort_minus_buchstabe.charAt(5)==
                wort_minus_buchstabe.charAt(6)))
586 {
```

```
587 enthalten = true;;
588 }
589 else
590 {// "ABCADCDB" vorwaerts
591 if ((wort_minus_buchstabe.charAt(0)== wort_minus_buchstabe.
        charAt(3)) && (wort_minus_buchstabe.charAt(1)==
        wort_minus_buchstabe.charAt(7))
592 && (wort_minus_buchstabe.charAt(2)== wort_minus_buchstabe.
        charAt(5)) && (wort_minus_buchstabe.charAt(4)==
        wort_minus_buchstabe.charAt(6)))
593 {
594 enthalten = true;
595 }
596 else
597 {// "ABCADCDB" vorwaerts
598 if ((wort_minus_buchstabe.charAt(0)== wort_minus_buchstabe.
        charAt(6)) && (wort_minus_buchstabe.charAt(1)==
        wort_minus_buchstabe.charAt(3))
599          && (wort_minus_buchstabe.charAt(2)==
                wort_minus_buchstabe.charAt(5)) && (
                wort_minus_buchstabe.charAt(4)==
                wort_minus_buchstabe.charAt(7)))
600 {
601 enthalten = true;
602 }
603 }}}}}
604 if (enthalten == false)
605 {
606 verbotene_woerter_neu.add(verbotenes_wort);
607 }}}
608 public static void main(String[] args)
609 {
610 kombinationen("", 0);
611 int anzahl_kombi = kombinationen.size();
612 int v = 0;
613 for(int m = 0; m < anzahl_kombi; m++)
614 {
615 int farbwechsel_anzahl = 1;
616 String tempw = new String ((String) kombinationen.get(m));
617 String w_m = new String ((String) kombinationen.get(m));
618
619 int laenge_tempw = tempw.length();
620
621 w_strich(tempw, w_m, laenge_tempw);
622
623 farbwechsel_minimal(tempw, "vorwaerts");
624
625 String temp_faerbung = new String((String) erste_faerbung.
        get(v));
```

```
626
627 int anzahl_schleife = (laenge_tempw/2)-1;
628 int letzter_index = (anzahl_schleife * (v+1))-1;
629 v = v+1;
630
631 StringBuffer faerbung_buffer = new StringBuffer();
632 faerbung_buffer.append(new String(temp_faerbung));
633
634 for(int n = 0; n < anzahl_schleife; n++)
635 {
636 int index_erstesZ_w_strich_m = indexliste.get(letzter_index)
    ;
637
638 if(index_erstesZ_w_strich_m == 0)
639 {
640 faerbung_buffer.insert(0, "0");
641 faerbung_buffer.append("1");
642
643 if(farbwechsel_anzahl %2 == 0)
644 {
645 farbwechsel_anzahl = farbwechsel_anzahl +1;
646 }
647 else;
648 }
649 else
650 {
651 if(index_erstesZ_w_strich_m >= faerbung_buffer.length())
652 {
653 String temp_vorgaenger_farbe = new String(faerbung_buffer.
    substring(faerbung_buffer.length()-1));
654 faerbung_buffer.append(temp_vorgaenger_farbe);
655 if(temp_vorgaenger_farbe.contains("1"))
656 {
657 faerbung_buffer.append("0");
658 }
659 else
660 {
661 faerbung_buffer.append("1");
662 }
663 farbwechsel_anzahl = farbwechsel_anzahl +1;
664 }
665 else
666 {
667 if(farbwechsel_anzahl %2 == 1)
668 {
669 String temp_vorgaenger_farbe = new String(faerbung_buffer.
    substring(index_erstesZ_w_strich_m-1,
    index_erstesZ_w_strich_m));
```

```
670 String temp_nachfolger_farbe = new String(faerbung_buffer.
        substring(index_erstesZ_w_strich_m,
        index_erstesZ_w_strich_m +1));
671
672 if(temp_vorgaenger_farbe.contains("1") &&
        temp_nachfolger_farbe.contains("1"))
673 {
674 faerbung_buffer.insert(index_erstesZ_w_strich_m, "1");
675 faerbung_buffer.append("0");
676 farbwechsel_anzahl = farbwechsel_anzahl +1;
677 }
678 else
679 {
680 faerbung_buffer.insert(index_erstesZ_w_strich_m, "0");
681 faerbung_buffer.append("1");
682 }}
683 else
684 {
685 String temp_vorgaenger_farbe = new String(faerbung_buffer.
        substring(index_erstesZ_w_strich_m-1,
        index_erstesZ_w_strich_m));
686 String temp_nachfolger_farbe = new String(faerbung_buffer.
        substring(index_erstesZ_w_strich_m,
        index_erstesZ_w_strich_m +1));
687 if(temp_vorgaenger_farbe.contains("0") &&
        temp_nachfolger_farbe.contains("0"))
688 {
689 faerbung_buffer.insert(index_erstesZ_w_strich_m, "0");
690 faerbung_buffer.append("1");
691 farbwechsel_anzahl = farbwechsel_anzahl +1;
692 }
693 else
694 {
695 faerbung_buffer.insert(index_erstesZ_w_strich_m, "1");
696 faerbung_buffer.append("0");
697 }}}}
698 letzter_index = letzter_index -1;
699 }
700 int farbwechsel_anzahl_real = 0;
701 for (int p = 0; p < faerbung_buffer.length()-1; p++)
702 {
703 char vorgaenger_farbe = faerbung_buffer.charAt(p);
704 char nachfolger_farbe = faerbung_buffer.charAt(p+1);
705
706 if(vorgaenger_farbe != nachfolger_farbe)
707 {
708 farbwechsel_anzahl_real = farbwechsel_anzahl_real +1;
709 }
710 else;
```

```
711
712 }
713 farbwechsel_real.add(farbwechsel_anzahl_real);
714
715 String w_gefaerbt = faerbung_buffer.toString();
716
717 faerbung.add(w_gefaerbt);
718 farbwechsel.add(farbwechsel_anzahl);
719 }
720 verbotene_woerter(farbwechsel_real, farbwechsel_min);
721 for(int s = 0; s < verbotene_woerter.size(); s++)
722 {
723 String temp_wort = verbotene_woerter.get(s);
724 StringBuffer temp_wort_rueckwaerts = new StringBuffer();
725
726 for(int d = temp_wort.length() -1; d >= 0; d--)
727 {
728 temp_wort_rueckwaerts.append(temp_wort.charAt(d));
729 }
730 farbwechsel_minimal(temp_wort_rueckwaerts.toString(), "
    rueckwaerts");
731 verbotene_wortliste_rueckwaerts.add(temp_wort_rueckwaerts.
    toString());
732 }
733 bekannte_verbotene_woerter(verbotene_woerter);
734
735 try{
736 BufferedWriter bw = new BufferedWriter (new FileWriter ("C:/
    Users/c/Documents/Bachelor-Arbeit/woerter10.txt"));
737 for(int x = 0; x < anzahl_kombi; x++)
738 {
739 bw.write(kombinationen.get(x));
740 bw.newLine();
741 }
742 bw.close();
743 }
744 catch(IOException ex){}
745
746 try{
747 BufferedWriter bw = new BufferedWriter (new FileWriter ("C:/
    Users/c/Documents/Bachelor-Arbeit/faerbung10.txt"));
748 for(int o = 0; o < anzahl_kombi; o++)
749 {
750 bw.write(faerbung.get(o));
751 bw.newLine();
752 }
753 bw.close();
754 }
755 catch(IOException ex){}
```

```
756
757 try{
758 BufferedWriter bw = new BufferedWriter (new FileWriter ("C:/
       Users/c/Documents/Bachelor-Arbeit/farbwechsel_real10.txt
       "));
759 for(int u = 0; u < anzahl_kombi; u++)
760 {
761 bw.write(String.valueOf(farbwechsel_real.get(u)));
762 bw.newLine();
763 }
764 bw.close();
765 }
766 catch(IOException ex){}
767
768 try{
769 BufferedWriter bw = new BufferedWriter (new FileWriter ("C:/
       Users/c/Documents/Bachelor-Arbeit/farbwechsel_min10.txt"
       ));
770 for(int i = 0; i < anzahl_kombi; i++)
771 {
772 bw.write(String.valueOf(farbwechsel_min.get(i)));
773 bw.newLine();
774 }
775 bw.close();
776 }
777 catch(IOException ex){}
778
779 try{
780 BufferedWriter bw = new BufferedWriter (new FileWriter ("C:/
       Users/c/Documents/Bachelor-Arbeit/
       farbwechsel_min_rueck10.txt"));
781 for(int p = 0; p < verbotene_woerter.size(); p++)
782 {
783 bw.write(String.valueOf(farbwechsel_min_rueck.get(p)));
784 bw.newLine();
785 }
786 bw.close();
787 }
788 catch(IOException ex){}
789
790 try{
791 BufferedWriter bw = new BufferedWriter (new FileWriter ("C:/
       Users/c/Documents/Bachelor-Arbeit/verbotene_woerter10.
       txt"));
792 for(int c = 0; c < verbotene_woerter.size(); c++)
793 {
794 bw.write(verbotene_woerter.get(c));
795 bw.newLine();
796 }
```

```
797 bw.close();
798 }
799 catch(IOException ex){}
800
801 try{
802 BufferedWriter bw = new BufferedWriter (new FileWriter ("C:/
      Users/c/Documents/Bachelor-Arbeit/
      verbotene_woerter_rueckwaerts_10.txt"));
803 for(int e = 0; e < verbotene_woerter.size(); e++)
804 {
805 bw.write(verbotene_wortliste_rueckwaerts.get(e));
806 bw.newLine();
807 }
808 bw.close();
809 }
810 catch(IOException ex){}
811 try{
812 BufferedWriter bw = new BufferedWriter (new FileWriter ("C:/
      Users/c/Documents/Bachelor-Arbeit/
      verbotene_woerter10_neu.txt"));
813 for(int f = 0; f < verbotene_woerter_neu.size(); f++)
814 {
815 bw.write(verbotene_woerter_neu.get(f));
816 bw.newLine();
817 }
818 bw.close();
819 }
820 catch(IOException ex){}
821 }
822 }
```

A.2 Quelltext Methode 'laminar_hyper'

```
1  import java.util.ArrayList;
2
3  import java.util.List;
4  import java.io.*;
5
6
7
8  public class laminar_hyper_wort
9  {
10
11          static ArrayList<Integer> anz_laminar_int = new
                ArrayList();
12
13
14 static void laminar_hyper (String w)
15 {
16 int laenge_w = w.length();
17 StringBuffer w_buffer = new StringBuffer();
18 w_buffer.append(new String(w));
19 ArrayList<String> int_wort = new ArrayList();
20 ArrayList<String> lam_intervalls = new ArrayList();
21 StringBuffer bekannt = new StringBuffer();
22 String bekannt_string = new String();
23 int i = 0;
24
25 while(i < (laenge_w))
26 {
27 String buchstabe = w_buffer.substring(i, i+1);
28 bekannt_string = bekannt.toString();
29 StringBuffer intervall = new StringBuffer();
30
31 if(bekannt_string.contains(buchstabe))
32 { i = i+1;}
33 else
34 {
35 bekannt.append(buchstabe);
36 int index_zweit = w_buffer.lastIndexOf(buchstabe);
37 for(int q = 1; q <= index_zweit; q++)
38 {intervall.append(q);}
39 int_wort.add(intervall.toString());
40 i = i+1;
41 }}
42 String i1 = int_wort.get(0);
43 String i2 = int_wort.get(1);
44 String i3 = int_wort.get(2);
45 String i4 = int_wort.get(3);
46 String i5 = int_wort.get(4);
```

```
47 String i6 = int_wort.get(5);
48
49 int last_i1 = 200;
50 if((i1.substring(i1.length()-2, i1.length()).contains("10"))
       )
51 { last_i1 = 10; }
52 else
53 {
54 if((i1.substring(i1.length()-2, i1.length()).contains("11"))
       )
55 { last_i1 = 11; }
56 else
57 {
58 last_i1 = i1.charAt(i1.length()-1);
59 last_i1 = last_i1 + 48;
60 }
61 }
62 int first_i1 = i1.charAt(0);
63 first_i1 = last_i1 - 48;
64
65 int last_i2 = 200;
66 if((i2.substring(i2.length()-2, i2.length()).contains("10"))
       )
67 { last_i2 = 10; }
68 else
69 {
70 if((i2.substring(i2.length()-2, i2.length()).contains("11"))
       )
71 { last_i2 = 11; }
72 else
73 {
74 last_i2 = i2.charAt(i2.length()-1);
75 last_i2 = last_i2 - 48;
76 }
77 }
78 int first_i2 = i2.charAt(0);
79 first_i2 = last_i2 - 48;
80
81 int last_i3 = 200;
82 if((i3.substring(i3.length()-2, i3.length()).contains("10"))
       )
83 { last_i3 = 10; }
84 else
85 {
86 if((i3.substring(i3.length()-2, i3.length()).contains("11"))
       )
87 { last_i3 = 11; }
88 else
89 {
```

```
90  last_i3 = i3.charAt(i3.length()-1);
91  last_i3 = last_i3 - 48;
92  }
93  }
94  int first_i3 = i3.charAt(0);
95  first_i3 = last_i3 - 48;
96
97  int last_i4 = 200;
98  if((i4.substring(i4.length()-2, i4.length()).contains("10"))
        )
99  { last_i4 = 10; }
100 else
101 {
102 if((i4.substring(i4.length()-2, i4.length()).contains("11"))
        )
103 { last_i4 = 11; }
104 else
105 {
106 last_i4 = i4.charAt(i4.length()-1);
107 last_i4 = last_i4 - 48;
108 }
109 }
110 int first_i4 = i4.charAt(0);
111 first_i4 = last_i4 - 48;
112
113 int last_i5 = 200;
114 if((i5.substring(i5.length()-2, i5.length()).contains("10"))
        )
115 { last_i5 = 10; }
116 else
117 {
118 if((i5.substring(i5.length()-2, i5.length()).contains("11"))
        )
119 { last_i5 = 11; }
120 else
121 {
122 last_i5 = i5.charAt(i5.length()-1);
123 last_i5 = last_i5 - 48;
124 }
125 }
126 int first_i5 = i5.charAt(0);
127 first_i5 = last_i5 - 48;
128
129 int last_i6 = 200;
130 if((i6.substring(i6.length()-2, i6.length()).contains("10"))
        )
131 { last_i6 = 10; }
132 else
133 {
```

```
134 if((i6.substring(i6.length()-2, i6.length()).contains("11"))
        )
135 { last_i6 = 11; }
136 else
137 {
138 last_i6 = i6.charAt(i6.length()-1);
139 last_i6 = last_i6 - 48;
140 }
141 }
142 int first_i6 = i6.charAt(0);
143 first_i6 = last_i6 - 48;
144
145 lam_intervalls.add(i1);
146 boolean i2_disjunkt = true;
147 boolean i3_disjunkt = true;
148 boolean i4_disjunkt = true;
149 boolean i5_disjunkt = true;
150 boolean i6_disjunkt = true;
151
152 for(int h = 0; h < lam_intervalls.size(); h++)
153 {
154 String temp_int = lam_intervalls.get(h);
155 int first_temp = 200;
156 int last_temp = 200;
157
158 if((temp_int.substring(0,  2).contains("10")))
159 {first_temp = 10; }
160 else
161 {
162 if((temp_int.substring(0,  2).contains("11")))
163 {first_temp = 11; }
164 else
165 {
166 first_temp = temp_int.charAt(0);
167 first_temp = first_temp - 48;
168 }}
169 if((temp_int.substring(temp_int.length()-2, temp_int.length
        ()).contains("10")))
170 { last_temp = 10; }
171 else
172 {
173 if((temp_int.substring(temp_int.length()-2, temp_int.length
        ()).contains("11")))
174 { last_temp = 11; }
175 else
176 {
177 last_temp = temp_int.charAt(temp_int.length()-1);
178 last_temp = last_temp -48;
179 }}
```

```
180 if((first_i2 < first_temp && last_i2 > first_temp && last_i2
        < last_temp) ||
181 ((first_i2 > first_temp && first_i2 < last_temp) && last_i2
       > last_temp))
182 { i2_disjunkt = false;}}
183 if(i2_disjunkt == true)
184 { lam_intervalls.add(i2);}
185 else{}
186
187 for(int h = 0; h < lam_intervalls.size(); h++)
188 {
189 String temp_int = lam_intervalls.get(h);
190 int first_temp = 200;
191 int last_temp = 200;
192
193 if((temp_int.substring(0,  2).contains("10")))
194 {first_temp = 10; }
195 else
196 {
197 if((temp_int.substring(0,  2).contains("11")))
198 {first_temp = 11; }
199 else
200 {
201 first_temp = temp_int.charAt(0);
202 first_temp = first_temp - 48;
203 }}
204 if((temp_int.substring(temp_int.length()-2, temp_int.length
        ()).contains("10")))
205 { last_temp = 10; }
206 else
207 {
208 if((temp_int.substring(temp_int.length()-2, temp_int.length
        ()).contains("11")))
209 { last_temp = 11; }
210 else
211 {
212 last_temp = temp_int.charAt(temp_int.length()-1);
213 last_temp = last_temp -48;
214 }}
215 if((first_i3 < first_temp && last_i3 > first_temp && last_i3
        < last_temp) ||
216 ((first_i3 > first_temp && first_i3 < last_temp) && last_i3
       > last_temp))
217 { i3_disjunkt = false;}}
218 if(i3_disjunkt == true)
219 { lam_intervalls.add(i3);}
220 else{}
221
222 for(int h = 0; h < lam_intervalls.size(); h++)
```

```
223 {
224 String temp_int = lam_intervalls.get(h);
225 int first_temp = 200;
226 int last_temp = 200;
227
228 if((temp_int.substring(0, 2).contains("10")))
229 {first_temp = 10; }
230 else
231 {
232 if((temp_int.substring(0, 2).contains("11")))
233 {first_temp = 11; }
234 else
235 {
236 first_temp = temp_int.charAt(0);
237 first_temp = first_temp - 48;
238 }}
239 if((temp_int.substring(temp_int.length()-2, temp_int.length
      ()).contains("10")))
240 { last_temp = 10; }
241 else
242 {
243 if((temp_int.substring(temp_int.length()-2, temp_int.length
      ()).contains("11")))
244 { last_temp = 11; }
245 else
246 {
247 last_temp = temp_int.charAt(temp_int.length()-1);
248 last_temp = last_temp -48;
249 }}
250 if((first_i4 < first_temp && last_i4 > first_temp && last_i4
      < last_temp) ||
251 ((first_i4 > first_temp && first_i4 < last_temp) && last_i4
     > last_temp))
252 { i4_disjunkt = false;}}
253 if(i4_disjunkt == true)
254 { lam_intervalls.add(i4);}
255 else{}
256 for(int h = 0; h < lam_intervalls.size(); h++)
257 {
258 String temp_int = lam_intervalls.get(h);
259 int first_temp = 200;
260 int last_temp = 200;
261
262 if((temp_int.substring(0, 2).contains("10")))
263 {first_temp = 10; }
264 else
265 {
266 if((temp_int.substring(0, 2).contains("11")))
267 {first_temp = 11; }
```

```
268 else
269 {
270 first_temp = temp_int.charAt(0);
271 first_temp = first_temp - 48;
272 }}
273 if((temp_int.substring(temp_int.length()-2, temp_int.length
        ()).contains("10")))
274 { last_temp = 10; }
275 else
276 {
277 if((temp_int.substring(temp_int.length()-2, temp_int.length
        ()).contains("11")))
278 { last_temp = 11; }
279 else
280 {
281 last_temp = temp_int.charAt(temp_int.length()-1);
282 last_temp = last_temp -48;
283 }}
284 if((first_i5 < first_temp && last_i5 > first_temp && last_i5
        < last_temp) ||
285 ((first_i5 > first_temp && first_i5 < last_temp) && last_i5
       > last_temp))
286 { i5_disjunkt = false;}}
287 if(i5_disjunkt == true)
288 { lam_intervalls.add(i5);}
289 else{}
290
291 for(int h = 0; h < lam_intervalls.size(); h++)
292 {
293 String temp_int = lam_intervalls.get(h);
294 int first_temp = 200;
295 int last_temp = 200;
296
297 if((temp_int.substring(0,  2).contains("10")))
298 {first_temp = 10; }
299 else
300 {
301 if((temp_int.substring(0,  2).contains("11")))
302 {first_temp = 11; }
303 else
304 {
305 first_temp = temp_int.charAt(0);
306 first_temp = first_temp - 48;
307 }}
308 if((temp_int.substring(temp_int.length()-2, temp_int.length
        ()).contains("10")))
309 { last_temp = 10; }
310 else
311 {
```

```
312 if((temp_int.substring(temp_int.length()-2, temp_int.length
        ()).contains("11")))
313 { last_temp = 11; }
314 else
315 {
316 last_temp = temp_int.charAt(temp_int.length()-1);
317 last_temp = last_temp -48;
318 }}
319 if((first_i6 < first_temp && last_i6 > first_temp && last_i6
        < last_temp) ||
320 ((first_i6 > first_temp && first_i6 < last_temp) && last_i6
       > last_temp))
321 { i6_disjunkt = false;}}
322 if(i6_disjunkt == true)
323 { lam_intervalls.add(i6);}
324 else{}
325
326
327 anz_laminar_int.add(lam_intervalls.size());
328 }
329         public static void main(String[] args)
330
331         {
332
333         }
334
335
336 }
```

Literatur

[1] S.D. Andres, W. Hochstättler: Some heuristics for the binary paint shop problem and their expected number of colour changes, Journal of Discrete Algorithms 9 (2011) 203-211

[2] D. Rautenbach, Z. Szigeti: Greedy colorings of words, Discrete Applied Mathematics 160(2012) 1872-1874

[3] Th. Epping, W. Hochstättler, P. Oertel: Complexity results on a paint shop problem, Discrete Applied Mathematics 136(2004) 217-226

[4] H. Amini, F. Meunier, H. Michel, A. Mohajeri: Greedy colorings for the binary paintshop problem, J. Discrete Algorithms 8 (2010) 8-14

[5] Th. Epping, W. Hochstättler: Abuse of multiple sequence alignment in a paint shop, Technical Report zaik2001-418, Center of Applied Computer Science, 2001

[6] S. Spieckermann, S. Voß: Paint shop simulation in the automotive industry, ASIM Mitt. 54 (1996) 367-380

[7] P. Bonsma, Th. Epping, W. Hochstättler: Complexity results on restricted instances of a paint shop problem for words, Discrete Appl. Math., 154 (2006), pp. 1335-1343

[8] F. Meunier, A. Sebő: Paint shop, odd cycles and splitting necklace, Discrete Appl. Math.

[9] R. Preiß: Beweisvisualisierung und -analyse mit Hypergraphen, Aachen: Shaker, 1998 (Berichte aus der Informatik) Zugl.: Karlsruhe, Univ., Diss., 1998

[10] C. H. Papadimitriou: Optimization, Approximation, and Complexity Classes, Journal of Computer and System Sciences 43, 425-440 (1991)

[11] S. Khot: On the power of unique 2-prover 1-round games. In 34th Annual ACM Symposium on the Theory of Computing, pages 767-775, July 2002.

[12] Th. Epping, W. Hochstättler, R. Nickel, P. Oertel: Order sequencing in the automobile industry, Lars Mönch, Giselher Pankratz (eds.) "Intelligente Systeme zur Entscheidungsunterstützung ", Teilkonferenz der Multikonferenz Wirtschaftsinformatik München, 26.02.2008-28.02.2008, SCS Publishing House e. V., 49-64