Energieminimierung eindimensionaler Arrays gekoppelter Spins in einem Antiferromagneten

Bachelorarbeit

(korrigierte Version vom 23.03.2020)

Bearbeitungsbeginn: 25. Juni 2019

FernUniversität in Hagen
Fakultät für Mathematik und Informatik
Lehrgebiet Diskrete Mathematik und Optimierung

Betreuer: Prof. Dr. Winfried Hochstättler

Name:	Mario Quick
Matrikelnummer:	7564163
Studiengang:	B.Sc. Mathematik

Inhaltsverzeichnis

1. Einleitung **3**

2. Grundlagen und Problembeschreibung **5**
2.1. Das Ising-Modell 5
2.2. Die Problembeschreibung 6
2.3. Das Binary-Paintshop-Maximization-Problem 6
2.4. Konventionen bezüglich Notation 7
2.5. Definitionen 8

3. Komplexitätstheoretische Einordnung des Binary Paintshop Maximization Problems **11**
3.1. Das Binary-Paintshop-Maximization-Problem als Entscheidungsproblem 11
3.2. Das Binary-Paintshop-Maximization-Problem als Optimierungsproblem 13
3.2.1. Der Greedy-Algorithmus 13
3.2.2. Abschätzungen für die Anzahl der maximal erreichbaren Farbwechsel . . 15
3.2.3. Abschätzungen für die Anzahl der mindestens erreichbaren Farbwechsel 18
3.2.4. Das Binary Paintshop Maximization Problem und $\mathcal{APX}$-hardness 35

4. Approximationen für das Binary-Paintshop-Maximization-Problem **37**
4.1. Binary Integer Programming (0-1) 37
4.2. Formulierung von BPMP als Quadratisches Programm 38
4.3. Grundlagen der Semidefiniten Programmierung 40
4.4. Semidefinite Relaxierung 42
4.5. Ermittlung einer zulässigen Lösung für BPMP 44
4.6. Praktische Umsetzung 47

5. Zusammenfassung und Ausblick **50**

A. Anhang **52**
A.1. Quellcode zum Programm Paintmax 52

Erklärung **58**

1. Einleitung

Im Jahre 1924 erhielt der junge Physiker Ernst Ising von seinem Doktorvater Wilhelm Lenz als Aufgabe gestellt, ein neues Modell zu untersuchen, welches den Ferromagnetismus beschreibt [1]. Zunächst wenig beachtet, ist dieses heute als Ising-Modell bekannt und gehört zu den am häufigsten untersuchten Modellen der statistischen Physik. Charakteristisch für das Modell ist, dass die Spins, welche die magnetischen Momente bilden und so die Energie des Systems bestimmen, nur die Werte -1 oder 1 annehmen. In unserer Arbeit wollen wir nun eine Kette solcher Spins untersuchen. Da wir Antiferromagnetismus voraussetzen wollen, wird die Energie minimiert, wenn jeweils benachbarte Spins entgegen gerichtet sind. Außerdem nehmen wir in unserem Modell weiterhin an, dass jeweils zwei Spins entlang der Kette durch eine Wechselwirkung fest gekoppelt sind, so dass sie sich immer entgegengesetzt ausrichten. Dies führt nun dazu, dass im Allgemeinen nicht alle benachbarten Spins entgegengesetzt ausgerichtet sein können. Wir wollen in dieser Arbeit nun untersuchen, wie sich eine Ausrichtung der Spins finden lässt, so dass die Gesamtenergie des Systems minimiert wird. Diese Aufgabe wollen wir mit Hilfe der Kombinatorischen Optimierung bearbeiten.

Dies führt uns zum Binary-Paintshop-Maximization-Problem (BPMP). Das Binary-Paintshop-Maximization-Problem lässt sich recht einfach wie folgend beschreiben: Gegeben ist eine Liste der Länge 2n, in der n verschiedene Buchstaben jeweils genau zweimal enthalten sind. Diese Liste soll nun mit zwei Farben so gefärbt werden, dass jeweils die zwei gleichen Vorkommen jedes Buchstabes unterschiedliche Farben erhalten. Gesucht ist nun eine Färbung, sodass die Anzahl der Farbwechsel zwischen benachbarten Buchstaben maximal wird. Das Binary-Paintshop-Maximization-Problem ist vom Binary Paintshop Minimization Problem (oft auch nur Binary-Paint-Shop-Problem genannt) abgeleitet. Dies ist ein Spezialfall des allgemeinen Paint-Shop-Problems, welches 2004 aus einer praktischen Fragestellung einer Lackiererei (Paint-Shop) eines Automobilherstellers heraus formuliert wurde [4]. Der Unterschied vom Binary-Paint-Shop-Problem zum Binary-Paintshop-Maximization-Problem besteht genau darin, dass wir bei ersterem versuchen, möglichst wenige Farbwechsel zwischen den einzelnen Nachbarn zu erzielen. Für das Binary-Paint-Shop-Problem konnten Bonsma, Epping und Hochstättler 2006 zeigen, dass es $\mathcal{NP}$-vollständig und sogar $\mathcal{APX}$-hart ist [5].

Im Jahr 1995 eregte eine Arbeit von Goemans und Williamson [3] die Aufmerksamkeit der Fachwelt, in der ein 0.878-Algorithmus polynomieller Laufzeit für das Max-Cut Problem vorgestellt wurde. Bis dahin hatte das beste bekannte Verfahren eine Leistungsgarantie von kaum mehr als 0.5 besessen. Das Max-Cut Problem ist eines der meist untersuchten Probleme der Komplexitätstheorie und gehörte schon zu Karps Liste 21 NP-vollständiger Probleme. Beim Max-Cut Problem geht es darum, die Knotenmenge eines kantengewichteten Graphen in zwei Teilmengen zu zerlegen, so dass das Gesamtgewicht der Kanten, welche inzidente Knoten in jeweils beiden Teilmengen besitzen, maximiert wird. Erreicht haben Goemans und Williamson ihr Ergebnis mit Hilfe der Semidefiniten Programmierung (SDP). Die Semidefinite Programmierung, die als Sonderfall auch die lineare Programmierung mit einschließt, behandelt Optimierungsprobleme, bei denen die gesuchten Variablen symmetrische Matrizen sind. Wir werden versuchen, die SDP auch auf unser Problem anzuwenden.

Die Arbeit ist so gegliedert, dass wir in Kapitel 2 einen Einblick in die physikalische Motivation für die Arbeit geben und das Binary-Paintshop-Maximization-Problem (BPMP) als

dahinterliegendes kombinatorische Problem vorstellen. Des Weiteren geben wir einige für diese Arbeit wichtige Definitionen aus der Komplexitätstheorie an, die für die Bearbeitung unseres kombinatorischen Problems wichtig sind. Dann werden wir in Kapitel 3 die komplexitätstheoretischen Eigenschaften des Binary-Paintshop-Maximization-Problems untersuchen. Schließlich werden wir in Kapitel 4 versuchen, das BPMP zu approximieren, wobei der Schwerpunkt hier auf der Semidefiniten Programmierung liegt.

2. Grundlagen und Problembeschreibung

2.1. Das Ising-Modell

Grundlage für unsere Betrachtungen ist das sogenannte Ising-Modell [1]. Dieses Modell dient der Beschreibung von Ferromagnetismus und Antiferromagnetismus von Festkörpern. Im Ising-Modell werden, wie auch in vielen anderen Modellen, die Ferromagnetismus beziehungsweise Antiferromagnetismus beschreiben, diese als die Folge der Spins der Atome des Materials betrachtet. Einen Spin können wir uns dabei als Drehimpuls von Teilchen vorstellen. Die Spins von Atomen setzen sich aus dem Elektronenspin, also dem Eigendrehimpuls der Elektronen, welcher in der Regel den größten Anteil liefert, dem Bahnimpuls der Elektronen und dem Kernspin zusammen.

Im Ising-Modell wird nun das Material als ein Gitter angenommen und auf den Gitterplätzen sitzen die Träger der atomaren Spins, welche ein permanentes magnetisches Moment des Atoms oder Ions bilden. Während in dem allgemeineren Heisenberg-Modell die Spins als Vektoroperatoren abgebildet werden, wird im Ising-Modell angenommen, dass die Spins nur zwei diskrete Zustände annehmen können (Spinwert $s_i = \pm 1$, Spin-up und Spin-down oder $\uparrow$ und $\downarrow$) [2]. Liegt kein äußeres Magnetfeld an, können wir die Energie des Systems berechnen durch

$$H = -\frac{1}{2} \sum_{i,j} J_{ij} s_i s_j. \tag{1}$$

Dabei bezeichnet

- s_i den Spinwert -1 oder 1 des Atoms an der Gitterstelle i,
- J_{ij} die Austauschkonstante (Stärke der Austauschkopplungs-Wechselwirkung) zwischen den Spins an den Gitterstellen i und j.

Häufig nimmt man an, dass J_{ij} nur für benachbarte Spins ungleich Null ist. Es gibt aber auch Modelle, bei denen die Wechselwirkung mit der Entfernung abnimmt oder aber auch mit Wechselwirkungen unendlicher Reichweite. Wenn die Austauschwechselwirkung positiv ist, nennt man sie ferromagnetisch, bei einer negativen bezeichnet man sie als antiferromagnetisch. Ein Ferromagnet beziehungsweise Antiferromagnet zeichnet sich dadurch aus, dass die jeweilige Wechselwirkung dominiert. Besitzen die einzelnen Spins mehr als zwei Wechselwirkungen, können sie sich unter Umständen nicht mehr so ausrichten, dass alle Paare von Spins mit negativer Wechselwirkung entgegen gerichtet sind. Dieses Unvermögen, sich nach seiner eigentlichen Wechselwirkung auszurichten, bezeichnet man auch als Frustration.

Eine sehr wichtige Rolle spielt das Ising-Modell in der statistischen Physik bei der Betrachtung von Phasenübergängen. Ein Phasenübergang ist ein plötzliches oder kontinuierliches Verschwinden oder Entstehen einer magnetischen Ordnung wie beispielsweise Ferromagnetismus oder Antiferromagnetismus. Solche Phänomene können so erklärt werden, dass mit steigender Temperatur die thermische Energie, also die kinetische Energie der Spins, größer wird als die Energie der Austauschwechselwirkung. Phasenübergänge spielen aber in dieser Arbeit keine Rolle.

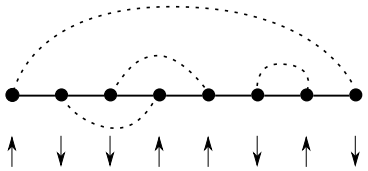

Abbildung 1: Ein Array von 8 Spins. Die Pfeile geben die Spinorientierung und die gestrichelten Linien die feste antiferromagnetische Kopplung mit erzwungener gegensätzlicher Orientierung wieder. Das hier gezeigte System der Spinorientierungen ist optimal bezüglich der minimalen Energie, wenn wir eine antiferromagnetisches Wechselwirkung zwischen benachbarten Spins annehmen.

2.2. Die Problembeschreibung

In dieser Arbeit wollen wir ein spezielles Problem eines eindimensionalen Ising-Modells bearbeiten. Wir nehmen unserer Material als eine Kette (Array) von Spins beliebiger Länge an. Zwischen den jeweils benachbarten Spins liegt eine antiferromagnetische Wechselwirkung gleicher Stärke vor, die wir ohne Einschränkung mit $J_{ij} = -1$ annehmen können. Somit ergibt sich die Energie H eines Arrays der Länge m zu

$$H = \sum_{1 \leq i \leq m-1} s_i s_{i+1}. \tag{2}$$

Des Weiteren sollen alle Spins genau mit einem anderen Spin fest antiferromagnetisch gekoppelt sein, dass heißt die dadurch entstehenden Paare sollen in jedem Fall entgegengesetzt ausgerichtet sein. Die Paarbildung soll unabhängig von der Position der beiden Spins zufällig erfolgen. Durch die feste Kopplung können sich nun im Allgemeinen nicht mehr alle Spins so ausrichten, dass sie energetisch günstig entgegengesetzt ihrer Nachbarn orientiert sind. Die Frage, die wir in dieser Arbeit diskutieren wollen, lautet:

Problem 2.1. *Wie hoch ist die minimale Energie eines Array jeweils paarweise fest gekoppelter Spins und wie lässt sich eine Anordnung von Spinorientierungen finden, so dass die Energie H minimiert wird?*

Diese Fragestellung wollen wir mit Hilfe der Kombinatorischen Optimierung bearbeiten.

2.3. Das Binary-Paintshop-Maximization-Problem

Wir werden in diesem Abschnitt das Binary-Paintshop-Maximization-Problem (BPMP) vorstellen und zeigen, dass wir unterschiedliche Konfigurationen der paarweisen festen Kopplungen der Spins unseres antiferromagnetischen Arrays als Instanzen von BPMP auffassen können.

Dazu geben wir zuerst folgende formelle Definitionen an:

Definition 2.2. *(zulässige Färbung für binäre Wortfärbungsprobleme) Sei ein endliches Alphabet Σ aus n Buchstaben vorgeben. Sei $w = (a_1, ..., a_{2n})$ ein Wort der Länge $2n$, indem jeder Buchstabe aus Σ genau zweimal enthalten ist. Sei weiterhin F eine Menge zweier Farben. Als Färbung $f = (f_1, ...f_{2n})$ von w bezeichnen wir eine Abbildung, bei der wir jedem a_i aus w ein $f_i \in F$ zuordnen.*

Als zulässig bezeichnen wir eine Färbung von w, bei der jeder Buchstabe jeweils genau einmal mit jeder der beiden Farben gefärbt wird, das heißt, $f_i \neq f_j$, falls $a_i = a_j$.

Definition 2.3. *(Binary-Paintshop-Maximization-Problem (BPMP)) Sei $w = (a_1, ..., a_{2n})$ ein Wort der Länge $2n$, in dem jeder Buchstabe aus Σ genau zweimal enthalten ist. Finde eine zulässige Färbung $f = (f_1, ...f_{2n})$, so dass die Anzahl der Farbwechsel maximal wird. Als Farbwechsel bezeichnen wir ein $i \in \{1, \ldots, 2n-1\}$, falls $f_i \neq f_{i+1}$.*

Ist nun ein antiferromagnetisches Array der Länge $2n$ mit einer bestimmten, jeweils paarweise fest antiferromagnetisch gekoppelten Anordnung von Spins K gegeben, so wollen wir jeweils ein gekoppeltes Paar an Spins mit einem Buchstaben identifizieren und ein Wort w bilden, indem wir die Buchstaben in der Reihenfolgen der Spins im Array auflisten. Ist f eine Färbung von einem solchen w mit den zwei Farben -1 und 1, so wollen wir ein System von Spinorientierungen $S = (s_1, \ldots, s_n)$ gewinnen, indem wir die Spinorientierungen s_i mit der Farbe der korrespondierenden Buchstaben a_i identifizieren. So können wir das Array aus Abbildung 1 als Wort $w = (a, b, c, b, c, d, d, a)$ auffassen.

Bezeichnet nun $\phi(f)$ die Anzahl der Farbwechsel einer Färbung f, so ergibt sich die Energie eines Array der Länge $2n$ mit der Konfiguration K und den Spinorientierungen S zu

$$H(K, S) = -2\phi(f) + 2n - 1. \tag{3}$$

Mit $\gamma(w)$ als die Anzahl der Farbwechsel einer optimalen Färbung eines Wortes w folgt dann für die minimale Energie des Arrays

$$H(K)_{min} = -2\gamma(w) + 2n - 1. \tag{4}$$

2.4. Konventionen bezüglich Notation

Wir wollen noch folgende Festlegung treffen. Sprechen wir im weiteren Text von *Buchstaben*, so meinen wir direkt die n Elemente aus Σ. Beziehen wir uns auf die $2n$ Elemente, die als Zeichenfolge das Wort $w = (a_1, ...a_{2n})$ bilden, wollen wir diese als *Lettern* bezeichnen, dass heißt, zu jedem Buchstaben gehören zwei Lettern in w. Diese Unterscheidung mag auf den ersten Blick merkwürdig oder unnötig erscheinen, vereinfacht und präzisiert aber die weiteren wörtlichen Ausführungen und trägt so hoffentlich auch zu einem einfacheren Verständnis des Geschriebenen bei.
Die zwei Farben aus F werden wir in der Regel mit 0 und 1 oder -1 und 1 identifizieren. Der Grund dafür ist, dass die Ergebnisse einiger der in der Arbeit vorgestellten Approximationen Vektoren aus $\{0, 1\}^{2n}$ oder $\{-1, 1\}^{2n}$ sind.

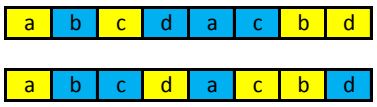

Abbildung 2: Zwei Färbungen $f = (0, 1, 0, 1, 1, 1, 0, 0)$ und $f^* = (0, 1, 1, 0, 1, 0, 0, 1)$ für das Wort $w = (a, b, c, d, a, c, b, d)$ mit $\phi(f) = 4$ und $\phi(f^*) = 5$. Die Färbung f^* ist optimal.

Wenn wir eine Färbung oder Teilfärbung f im Fließtext darstellen, wollen wir eingefärbte Buchstaben mit einem Strich oder einen Punkt über der Letter darstellen, also zum Beispiel eine Färbung f für das Wort $w = (a, b, a, b)$ mit $f = (0, 1, 1, 0) = (\dot{a}, \overline{b}, \overline{a}, \dot{b})$.
Im Gegensatz zu einem Farbwechsel wollen wir als Farbwechselfehler ein $i \in \{1, \ldots, 2n-1\}$ bezeichnen falls $f_i = f_{i+1}$.
Als optimal wollen wir eine zulässige Färbung für ein w bezeichnen, falls es keine zulässige Färbung für w gibt, die mehr Farbwechsel besitzt.

2.5. Definitionen

Bevor wir die Komplexität das Binary Paintshop Minimization Problems untersuchen, wollen wir noch die dazu benötigten grundlegenden Definitionen aus der Komplexitätstheorie angeben. Die folgenden drei Definitionen sind [9] entnommen. Eine für die Komplexitätstheorie fundamentale Komplexitätsklasse ist die Klasse $\mathcal{NP}$.

Definition 2.4. *(Klasse $\mathcal{NP}$) Sei Σ^* ein Alphabet. Die Klasse $\mathcal{NP}$ besteht aus der Menge aller Sprachen $L \subseteq \Sigma^*$ bei der es einen polynomialen Algorithmus M gibt, so dass für jedes $x \in \Sigma^*$ gilt:*
$(\exists y \in \Sigma^* : \ M$ akzeptiert$(x, y)) \Longleftrightarrow x \in L$.

Informell gesagt, handelt es sich um eine Klasse von Problemen, deren Lösung man nicht zwingend in Polynomialzeit findet, aber eine gegebene Lösung sich in Polynomialzeit verifizieren lässt. Sind nun zwei Probleme L und L' gegeben und man möchte entscheiden, ob Problem L mindestens so schwierig ist wie Problem L', kann man versuchen Problem L auf Problem L' abzubilden, dieses zu lösen und daraus eine Lösung für L abzuleiten:

Definition 2.5. *(Polynomialzeitreduktion) Sind Σ^* und Σ'^* Alphabete und $L \subseteq \Sigma^*$, $L' \subseteq \Sigma'^*$ zwei Sprachen und ist $f : L \to L'$ eine Abbildung mit $\forall x \in \Sigma^* : x \in L \Longleftrightarrow f(x) \in L'$, so nennen wir f eine Polynomialzeitreduktion (auch polynomielle Reduktion) von L auf L'. Gibt es einen Algorithmus M, der f in Polynomialzeit berechnet, so nennen wir f eine Polynomialzeitreduktion von L auf L' und schreiben $\Pi_L \preceq_P \Pi_{L'}$.*

Probleme, die in $\mathcal{NP}$ liegen und mindestens so schwer sind wie alle anderen Probleme in $\mathcal{NP}$, gehören zur Klasse der $\mathcal{NP}$- vollständigen Probleme.

Definition 2.6. *(Klasse $\mathcal{NP}$-vollständig) Ein Problem (genauer: ein Entscheidungsproblem)* Π *heißt $\mathcal{NP}$-vollständig genau dann, wenn:*

1. $\Pi \in \mathcal{NP}$.
2. $\forall \Pi' \in \mathcal{NP} : \Pi' \preceq_P \Pi$.

Die zweite Bedingung ist gleichbedeutend damit, dass jedes Problem aus $\mathcal{NP}$ in Polynomialzeit auf Π reduziert werden kann,

Die Frage, ob ein Problem in $\mathcal{NP}$ liegt ist ein Entscheidungsproblem. Wir sind hier gezwungen, eine optimale Lösung zu berechnen und zu beweisen. Möchte man eine Lösung berechnen, die dem Optimalwert unter Beachtung der eingesetzten Ressourcen möglichst nahe kommt, führt das auf Optimierungsprobleme. Wir wollen hier Optimierungsprobleme so verstehen, dass es für diese Probleme eine bestimmte Menge an Eingaben (Instanzen) gibt, für jede Eingabe x eine nichtleere Menge zulässiger Lösungen $S(x)$ existiert und jeder Lösung s ein bezogen auf x positiver Wert $c(x, s)$ zugeordnet ist. Außerdem soll noch das Ziel festgelegt sein, also ob wir $c(x, s)$ minimieren oder maximieren wollen. Eine sehr wichtige Rolle spielt die Klasse $\mathcal{NPO}$.

Definition 2.7. *(Klasse $\mathcal{NPO}$) Ein Optimierungsproblem gehört zur Komplexitätsklasse $\mathcal{NPO}$ (nondeterministic polynomial- time optimization problem), wenn es für eine Eingabe (x, s) in polynomieller Zeit möglich ist, zu überprüfen, ob s eine für x zulässige Lösung ist, und im positiven Fall die Kostenfunktion $c(x, s)$ zu berechnen [10].*

Auch innerhalb von $\mathcal{NPO}$ lassen sich verschiedene Klassen definieren, je nachdem ob für die in diesen Klassen enthaltenen Probleme Algorithmen existieren, die eine bestimmte Approximationsgüte in einer bestimmten Laufzeit garantieren. Als Approximationsgüte eines gegebenen Algorithmus E für die Eingabe x wollen wir das Verhältnis $r_E(x) = \frac{c(x,s)}{OPT_E(x)}$ definieren, wobei $OPT_E(x)$ den Optimalwert des Problems zur Eingabe x darstellt. Eine wichtige Klasse von Problemen innerhalb $\mathcal{NPO}$ ist die Klasse $\mathcal{APX}$.

Definition 2.8. *(Klasse $\mathcal{APX}$) Ein Optimierungsproblem liegt in der Klasse $\mathcal{APX}$ (approximable), wenn es eine Zahl c und einen polynomiellen Algorithmus gibt, so dass der Algorithmus bei jeder zulässigen Eingabe x eine Lösung mit einer Güte $\geq c$ für Maximumprobleme bzw. $\leq c$ für Minimumprobleme liefert.*

Man bezeichnet eine entsprechende Approximation eines Problems aus $\mathcal{APX}$ auch als c-Approximation. Für manche Probleme existieren Algorithmen, die noch bessere Ergebnisse ermöglichen:

Definition 2.9. *(PTAS) Ein PTAS (polynomial-time approximation scheme) für ein Approximationsproblem ist ein Algorithmus, der für eine Eingabe x und einen Parameter $\epsilon > 0$ eine Lösung für x mit einer Approximationsgüte von $1 + \epsilon$ für Minimumprobleme bzw. $1 - \epsilon$ für Maximumprobleme berechnet [10].*

Um die Schwere von Optimierungsproblemen zu überprüfen, können wir wieder Reduktionen verwenden. Für Optimierungsprobleme gibt es eine Reihe unterschiedlicher Reduktionen, die jeweils auch unterschiedliche Eigenschaften durch die Reduktion erhalten. Eine wichtige

Rolle spielen beispielsweise L- Reduktionen für praktische Anwendungen und für theoretische Überlegungen die $PTAS$- Reduktion. Wir wollen deshalb hier beide Definition angeben.

Damit können wir nun auch die Menge der schwierigsten Probleme in $\mathcal{APX}$ beschreiben:

Definition 2.10. *(Klasse $\mathcal{APX}$- vollständig) Ein Problem (genauer: ein Optimierungsproblem) A heißt $\mathcal{APX}$-vollständig genau dann wenn:*

1. $A \in \mathcal{APX}$.
2. $\forall A' \in \mathcal{APX} : A' \preceq_{PTAS} A$.

Die zweite Bedingung ist gleichbedeutend damit, dass es für jedes Problem aus $\mathcal{APX}$ eine $PTAS$- Reduktion auf A gibt.

Definition 2.11. *($PTAS$-Reduktion) Eine $PTAS$-Reduktion eines Optimierungsproblems A auf ein Optimierungsproblems B, besteht aus einem Tripel (f, g, σ) mit folgenden Eigenschaften [10]:*

1. *f bildet Eingaben x des Problems A auf Eingaben $f(x)$ des Problems B ab und ist in polynomieller Zeit berechenbar.*
2. *g bildet aus Eingaben x des Problems A, Lösungen $y \in S_B(f(x))$ und Zahlen $\epsilon > 0$ auf Lösungen $g(x, y, \epsilon) \in S_A(x)$ ab und ist in polynomieller Zeit berechenbar.*
3. *$\forall \epsilon > 0, \exists \delta$ mit $\delta(\epsilon) > 0$, so dass falls $r_B(f(x), y) \geq 1 - \delta(\epsilon)$ (B ist Maximumproblem) bzw. $r_B(f(x), y) \leq 1 + \delta(\epsilon)$ (B ist Minimumproblem) ist, gilt $r_A(x, g(x, y, \epsilon)) \geq 1 - \epsilon$ (A ist Maximumproblem) bzw. $r_A(x, g(x, y, \epsilon)) \leq 1 + \epsilon$ (A ist Minimumproblem).*

Als sehr nützliches Werkzeug haben sich die sogenannten L- Reduktionen herausgestellt, wobei L hier für linear steht. Aus einer L-Reduktion folgt, falls B einen $PTAS$ besitzt, dass es auch einen $PTAS$ für A gibt.

Definition 2.12. *(L-Reduktion) Seien A und B Optimierungsprobleme und c_A und c_B die entsprechenden Kostenfunktionen. Das Paar der Funktionen f und g nennen wir L-Reduktion, falls folgende Merkmale zutreffen:*

1. *f und g sind in polynomieller Zeit berechenbar.*
2. *Ist x eine Instanz von Problem A, dann ist f eine Instanz von Problem B.*
3. *Ist y' eine Lösung zu $f(x)$, dann ist $g(y')$ eine Lösung zu x.*
4. *Es gibt eine positive Konstante α so dass $OPT_B(f(x)) \leq \alpha OPT_A(x)$.*
5. *Es gibt eine positive Konstante β so dass für jede Lösung y' von $f(x)$ gilt $|OPT_A(x) - c_A(g(y'))| \leq \beta |OPT_B(f(x)) - c_B(y')|$.*

3. Komplexitätstheoretische Einordnung des Binary Paintshop Maximization Problems

3.1. Das Binary-Paintshop-Maximization-Problem als Entscheidungsproblem

Zunächst wollen wir zeigen, dass das Binary Paintshop Maximization Problem NP-vollständig ist. Dazu benötigen wir das Binary-Paintshop-Minimization-Problem:

Definition 3.1. *(Binary-Paintshop-Minimization-Problem) Sei $w = (a_1, ..., a_{2n})$ ein Wort der Länge $2n$, indem jeder Buchstabe aus Σ genau zweimal enthalten ist. Finde eine zulässige Färbung $f_=(f_1, ...f_{2n})$ von w, so dass die Anzahl der Farbwechsel minimal wird.*

Dazu beweisen wir folgenden Satz [6]:

Satz 3.2. *Sei $w_1 = (a_1, \dots a_{2n})$ eine Instanz für das Binary Paintshop Minimization Problem unter der Benutzung der Buchstaben $x_1, \dots x_n$. Sei $w_2 = (b_1, \dots, b_{4n})$ das Wort, dass wir erhalten, wenn wir in w_1 jeden Buchstaben x_i durch das Paar x_i, z_i ersetzen. Genau dann gibt es für w_2 eine zulässige Färbung mit mindestens $4n - k - 1$ Farbwechseln, wenn w_1 eine zulässige Färbung mit höchstens k Farbwechseln besitzt.*

Beweis. a) Sei f^1 eine zulässige Färbung von w_1 für das Minimization Problem mit höchstens k Farbwechseln. Wir konstruieren eine Färbung f^2 für w_2, indem wir einerseits für die x_i die Färbung von f^1 übernehmen, also die erste beziehungsweise zweite Letter x_i in f^2 genau wie die erste beziehungsweise zweite Letter x_i in f^2 einfärben. Außerdem färben wir die z_i entgegengesetzt zu den davor befindlichen x_i. Seien x_i und x_j zwei Lettern in w_1 an den Stellen m und $m+1$. Dann befinden sich in w_2 an der Stelle $2m-1$ ein x_i, an der Stelle $2m$ ein z_i und an der Stelle $2m+1$ ein x_j. Dann entsteht in f^2 zwischen z_i und x_j genau kein Farbwechsel, wenn sich in f^1 zwischen x_i und x_j ein Farbwechsel befindet. Zwischen x_i und dem Nachfolger z_i findet immer ein Farbwechsel statt. In w_2 gibt es insgesamt $4n-1$ Letternpaare b_ib_{i+1}. So erhalten wir $4n - k - 1$ Farbwechsel in f^2.

b) Sei nun f^2 eine zulässige Färbung von w_2 für das Maximization Problem mit mindestens $4n-k-1$ Farbwechseln. Wir konstruieren nun eine Färbung f^{2*} aus f^2, indem wir alle Lettern z_i umfärben, die gleich gefärbt sind wie ihre Vorgänger x_i. Da so beide Lettern z_i umgefärbt werden, ist f^{2*} eine gültige Färbung von w_2. Da für jedes so umgefärbte z_i ein zusätzlicher Farbwechsel zwischen x_i und z_i entsteht und höchstens ein Farbwechsel zwischen z_i und der nachfolgenden Letter von z_i verschwindet, besitzt auch f^{2*} mindestens $4n-k-1$ Farbwechsel. Nun konstruieren wir eine Färbung f^1 für w_1 aus f^{2*}, indem wir für die x_i die Färbung der entsprechenden x_i aus f^{2*} übernehmen. Da die beiden Lettern des Buchstaben x_i unterschiedlich gefärbt werden, ist f^1 eine zulässige Färbung für w_1. Sei nun x_i eine Letter in w_1 und x_j der Nachfolger von x_i. Genau wenn es zwischen z_i und x_j in f^{2*} einen Farbwechsel gibt, gibt es keinen Farbwechsel zwischen x_i und x_j in f^1. In w_2 gibt es insgesamt $4n-1$ Letternpaare b_ib_{i+1}. Gibt es nun $4n-k-1$ Farbwechsel in f^2, enthält die Färbung f^1 höchstens k Farbwechsel.

Aus a) und b) folgt nun die Behauptung.

□

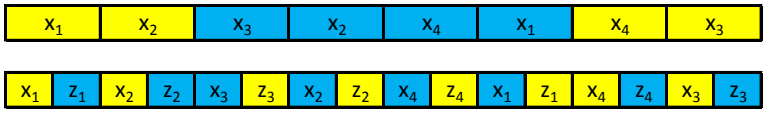

Abbildung 3: oben: gegebenes Wort w_1 mit Färbung f^1
unten: w_2 aus w_1 abgeleitet durch ersetzen von x_i mit x_i, z_i mit Färbung f^2

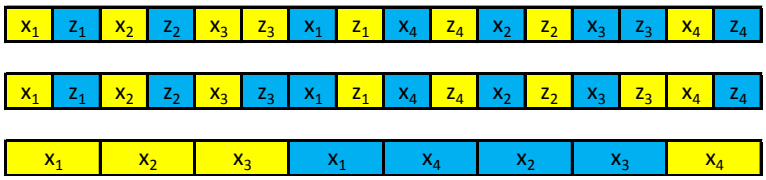

Abbildung 4: oben: Gegebenes Wort w_2 mit Färbung f^2
Mitte: Färbung f^{2*} aus f^2 durch umfärben der z_i mit gleicher Farbe wie x_i
unten: w_1 aus w_2 abgeleitet durch weglassen z_i mit Färbung f^1

Satz 3.3. *Das Binary-Paintshop-Maximization-Problem ist $\mathcal{NP}$- vollständig.*

Beweis. Der eben bewiesene Satz ist eine Polynomialzeitreduktion vom klassischen Binary Paintshop Minimization Problem auf das Binary Paintshop Maximization Problem. In [5] wurde gezeigt, dass ersteres $\mathcal{NP}$- vollständig ist. Somit ist BPMP $\mathcal{NP}$- schwer und da das Problem in $\mathcal{NP}$ liegt, ist BPMP auch $\mathcal{NP}$- vollständig.

□

Hier wollen wir außerdem noch eine Reduktion in die entgegengesetzte Richtung, also vom Maximization Problem auf das Minimization Problem, angeben.

Satz 3.4. *Sei $w_1 = (a_1, \ldots, a_{2n})$ eine Instanz für das Binary Paintshop Maximization Problem unter der Benutzung der Buchstaben $u_1, \ldots, u_n$. Sei $w_2 = (b_1, \ldots, b_{8n})$ das Wort, dass wir erhalten, wenn wir in w_1 jeden Buchstaben u_i beim erstmaligen Auftreten im Wort durch das Quadrupel u_i, x_i, y_i, x_i und beim zweiten Auftreten durch das Quadrupel u_i, z_i, y_i, z_i ersetzen. Genau dann gibt es für w_1 eine Färbung mit mindestens k Farbwechseln, wenn w_2 eine zulässige Färbung mit höchstens $4n - k - 1$ Farbwechseln besitzt.*

Beweis. a) Sei mit f^1 eine Färbung von w_1 für das Maximization Problems mit mindestens k Farbwechsel. Man färbe nun w_2 mit einer Färbung f^2 so, dass folgendes gilt:

- Die u_i werden gefärbt, wie die korrespondierenden u_i durch f^1 in w_1 gefärbt sind.
- Das erste x_i bzw. erste z_i werden gefärbt, wie die sich davor befindlichen u_i.

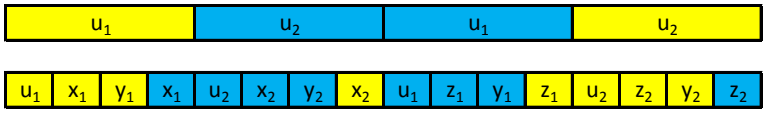

Abbildung 5: oben: Gegebenes Wort w_1 mit Färbung f^1, $k = 2$ Farbwechsel
unten: w_2 mit Färbung f^2, $4n - 1 - k = 5$ Farbwechsel

- Die y_i werden gefärbt, wie die sich davor befindlichen x_i bzw. erste z_i.
- Das zweite x_i bzw. zweite z_i werden entgegengesetzt gefärbt zu den davor befindlichen y_i.

Dann ist f^2 eine gültige Färbung von w_2. Innerhalb jedes der $2n$ Quadrupel u_i, x_i, y_i, x_i bzw. u_i, z_i, y_i, z_i besitzt f^2 genau einen Farbwechsel. Sei nun u_j die Letter, die in w_1 auf ein u_i folgt. Da u_i und das zweite x_i bzw. zweite z_i unterschiedlich gefärbt sind, gibt es genau keinen Farbwechsel zwischen dem zweiten x_i bzw. zweiten z_i und dem nachfolgenden u_j durch f^2, wenn u_i und u_j durch f^1 unterschiedlich gefärbt sind. Mit $\epsilon(f^1)$ als der Anzahl der Farbwechselfehler von f^1 gilt $k \leq 2n - \epsilon(f^1) - 1$, also $\epsilon(f^1) \leq 2n - k - 1$. Weiterhin besitzt f^2 insgesamt $2n + \epsilon(f^1)$ Farbwechsel. Daraus folgt, dass es eine Färbung von w_2 mit höchstens $4n - k - 1$ Farbwechseln gibt, falls w_1 eine Färbung mit mindestens k Farbwechseln besitzt.

b) Sei nun f^2 eine Färbung von w_2 mit höchstens $4n - k - 1$ Farbwechsel. Sei nun f^{2*} eine Färbung von w_2, die aus f^2 entsteht, wenn man jeden Farbwechsel zwischen benachbarten u_i und x_i beziehungsweise u_i und z_i entfernt, indem man die Färbung der beiden x_i bzw. z_i tauscht. Dann besitzt auch f^{2*} höchstens $4n - k - 1$ Farbwechsel, denn für jeden möglichen neu hinzukommenden Farbwechsel zwischen den umgefärbten zweiten x_i bzw. z_i und einem nachfolgenden u_j fällt jeweils der Farbwechsel zwischen u_i und dem ersten x_i bzw. z_i weg. Dann gibt es innerhalb jedes Teilwortes u_i, x_i, y_i, x_i bzw. u_i, z_i, y_i, z_i genau einen Farbwechsel und u_i und das zweite x_i beziehungsweise u_i und das zweite z_i sind unterschiedlich gefärbt. Sei nun f^1 eine Färbung von w_1, die dadurch entsteht, dass man die u_i so einfärbt wie die jeweils korrespondierenden u_i in w_2 durch f^{2*} eingefärbt werden. Sei u_j ein Nachfolger von u_i in w_1. Genau dann gibt es einen Farbwechsel zwischen u_i und u_j in f^1, wenn es keinen Farbwechsel durch f^{2*} zwischen dem zweiten x_i und u_j beziehungsweise dem zweiten z_i und u_j in w_2 gibt. Daraus folgt, dass es eine Färbung von w_1 mit mindestens k Farbwechseln gibt, falls w_2 eine Färbung mit höchstens $4n - k - 1$ Farbwechseln besitzt.

Aus a) und b) folgt nun die Behauptung.

□

3.2. Das Binary-Paintshop-Maximization-Problem als Optimierungsproblem

3.2.1. Der Greedy-Algorithmus

Wir wollen zunächst eine einfache Heuristik vorstellen, um eine Färbung mit möglichst vielen Farbwechseln für ein gegebenes Wort zu erhalten. Wir färben dazu das Wort von links nach

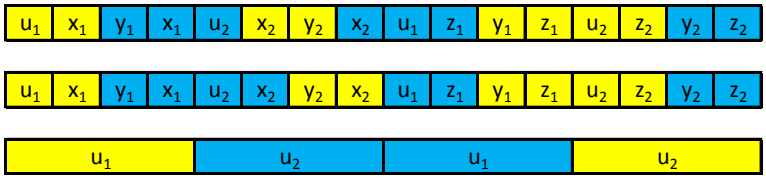

Abbildung 6: oben: Wort w_2 mit Färbung f^2, $4n-1-k=5$ Farbwechsel
Mitte: Wort w_2 mit Färbung f^{2*}, $4n-1-k=5$ Farbwechsel
unten: w_1 mit Färbung f^1, $k=2$ Farbwechsel

rechts so ein, dass die gerade zu färbende Letter immer entgegengesetzt zum Vorgänger gefärbt wird, soweit dies zulässig ist. Solche Algorithmen, die Probleme dadurch lösen, in dem sie für einzelne Teilschritte diejenige Lösung auswählen, die zu diesem Zeitpunkt den größtmöglichen Zuwachs an Gewinn liefert, werden auch Greedy-Algorithmen genannt. Wir wollen die eben beschriebene Heuristik im weiteren Verlauf der Arbeit als Greedy-Algorithmus bezeichnen. Kurz wollen wir hier einige Eigenschaften des Greedy-Algorithmus beschreiben.
Dass der Greedy-Algorithmus das BPMP nicht optimal löst, lässt sich beispielsweise am Wort $w = (a, b, c, b, a, c)$ sehen. Der Greedy-Algorithmus liefert $(\dot{a}, \overline{b}, \dot{c}, \dot{b}, \overline{a}, \overline{c})$. Man kann das Wort aber auch mit $(\dot{a}, \overline{b}, \overline{c}, \dot{b}, \overline{a}, \dot{c})$ einfärben, also einem Farbwechsel mehr. Das folgendes Ergebnis wird aber vom Greedy-Algorithmus garantiert.

Satz 3.5. *Sei w ein Wort der Länge $2n$. Der Greedy-Algorithmus färbt w mit mindestens n Farbwechseln.*

Beweis. Sei x_i die erste Letter der zwei Lettern eines Buchstabens von links aus gesehen und stehe x_i nicht an erster Stelle im Wort. Dann ergibt die Greedy-Färbung immer einen Farbwechsel zwischen dem Vorgänger von x_i und x_i. Sei x_j der Buchstabe, dessen erste Letter als letzte erste Letter aller Buchstaben im Wort vorkommt. Somit gibt es mindestens $n-1$ Farbwechsel bis einschließlich x_j und mindestens einen Farbwechsel zwischen den beiden Lettern x_j. □

Das Ergebnis kann nicht verbessert werden. Für $n=1$, $n=2$, $n=3$ und $n=4$ erhalten wir beispielsweise die Greedy-Färbungen $(\dot{x}_1, \overline{x}_1)$, $(\dot{x}_1, \overline{x}_2, \overline{x}_1, \dot{x}_2)$, $(\dot{x}_1, \overline{x}_2, \dot{x}_3, \dot{x}_2, \overline{x}_1, \overline{x}_3)$ und $(\dot{x}_1, \overline{x}_2, \dot{x}_3, \dot{x}_2, \overline{x}_4, \overline{x}_3, \overline{x}_1, \dot{x}_4)$, also n Farbwechsel für die entsprechenden Wörter. Für allgemeines ungerades $n > 3$ erhalten wir beispielsweise das eingefärbte Wort
$(\dot{x}_1, \overline{x}_2, \dot{x}_3, \dot{x}_2, \overline{x}_4, \overline{x}_3, \dot{x}_5, \dot{x}_4, \overline{x}_6, \overline{x}_5, \ldots, \dot{x}_i, \dot{x}_{i-1}, \overline{x}_{i+1}, \overline{x}_i, \ldots, \dot{x}_{2n}, \dot{x}_{2n-1}, \overline{x}_1, \overline{x}_{2n})$. Ein entsprechendes Beispiel für ein gerades $n > 4$ ist
$(\dot{x}_1, \overline{x}_2, \dot{x}_3, \dot{x}_2, \overline{x}_4, \overline{x}_3, \dot{x}_5, \dot{x}_4, \overline{x}_6, \overline{x}_5, \ldots, \dot{x}_i, \dot{x}_{i-1}, \overline{x}_{i+1}, \overline{x}_i, \ldots, \overline{x}_{2n}, \overline{x}_{2n-1}, \overline{x}_1, \dot{x}_{2n})$. Die Frage, wie viele Farbwechsel der Greedy-Algorithmus zu einem gegebenen Wort im Vergleich zu einer optimalen Lösung garantiert, wollen wir mit dem folgendem Satz beantworten:

Satz 3.6. *Für jedes $n > 1$ gibt es ein Wort der Länge $2n$, so dass der Greedy-Algorithmus nicht mehr als $n/(2n-2)$ der möglichen Farbwechsel erreicht.*

Abbildung 7: oben: Wort w der Länge $2n = 16$ mit Greedy-Färbung und n=8 Farbwechsel
unten: w mit optimaler Färbung und $2n - 2 = 14$ Farbwechsel

Beweis. Der Fall $n = 2$ ist trivial. Für unsere eben vorgestellten Beispielwörter, welche der Greedy-Algorithmus mit n Farbwechseln färbt, lassen sich in den Fällen $n = 3$ und $n = 4$ mit $(\dot{x}_1, \overline{x}_2, \overline{x}_3, \dot{x}_2, \overline{x}_1, \dot{x}_3)$ und $(\dot{x}_1, \overline{x}_2, \overline{x}_3, \dot{x}_2, \overline{x}_4, \dot{x}_3, \overline{x}_1, \dot{x}_4)$ Lösungen mit 4 beziehungsweise 6 Farbwechseln finden. Die Beispiele für ungerade $n > 3$ und gerade $n > 4$ lassen sich beide mit $(\dot{x}_1, \overline{x}_2, \overline{x}_3, \dot{x}_2, \overline{x}_4, \dot{x}_3, \overline{x}_5, \dot{x}_4, \overline{x}_6, \dot{x}_5, \ldots, \dot{x}_i, \overline{x}_{i-1}, \dot{x}_{i+1}, \overline{x}_i, \ldots, \overline{x}_{2n}, \dot{x}_{2n-1}, \overline{x}_1, \dot{x}_{2n})$ einfärben, so dass wir auch hier $2n - 2$ Farbwechsel erhalten. □

Der Greedy-Algorithmus ist also ein 0.5-Algorithmus, der in polynomieller Zeit läuft.

Korollar 3.7. *Das Binary Paintshop Maximization Problem liegt in $\mathcal{APX}$.*

Wir wollen unsere Betrachtungen über den Greedy-Algorithmus mit einer Vermutung über das im Durchschnitt zu erwartende Ergebnis abschließen:

Vermutung 3.8. *Sei w ein Wort der Länge $2n$ und seien in w die Lettern der n Buchstaben zufällig verteilt. Dann beträgt der Erwartungswert der erzielten Farbwechsel durch den Greedy-Algorithmus*

$$\mathbb{E}_n(g) = 2n - 1 - \sum_{k=1}^{n-1} \frac{2k^2 - 1}{4k^2 - 1} = \frac{3n^2 - 2n}{2n - 1}. \tag{5}$$

Nach dieser Vermutung werden also im Durchschnitt ungefähr 3/4 der Letternpaare $a_i a_{i+1}$ genügend langer Wörter durch den Greedy-Algorithmus mit einem Farbwechsel gefärbt. Wir werden hier die Vermutung nicht beweisen. Eventuell kann ein Beweis mit ähnlichen Überlegungen wie in [7], wo der Erwartungswert für das Minimumproblem bewiesen wird, erreicht werden.

3.2.2. Abschätzungen für die Anzahl der maximal erreichbaren Farbwechsel

Nun wollen wir die Anzahl der Farbwechsel, die durch eine Färbung eines gegebenen Wortes maximal erreicht werden können, abschätzen. Dazu wollen wir noch folgende Definition voranstellen:

Definition 3.9. *Sei $w = (a_1, \ldots, a_{2n})$. Sind a_i und a_j die zwei Lettern des Buchstaben x, so wollen wir die Menge $I_x := \{i, i+1, ..., j-1, j\}$ als Intervall von x bezeichnen. Als Kurzform*

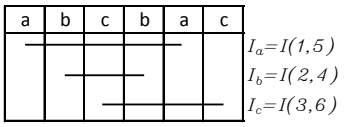

Abbildung 8: Wort w=(a,b,c,b,a,c) mit seinen Intervallen

werden wir auch $I_x = I(i, j)$ benutzen. Mit $|I_x| := j - i + 1$ wollen wir die Länge des Intervalls bezeichnen.

Somit lässt sich eine Instanz w des Binary Paintshop Maximization Problems auch durch die Menge $I(w) = \{I_x : x \in \Sigma\}$ beschreiben.
Sei a_k eine Letter an der Stelle k in einem Wort. Wir wollen sagen, a_k in liegt in einem Intervall I_z oder in einer Vereinigung $I_v \bigcup \cdots \bigcup I_z$ von Intervallen, falls $k \in I_z$ beziehungsweise $k \in I_v \bigcup \cdots \bigcup I_z$.
Damit können wir nun auch folgende Aussage treffen:

Satz 3.10. *Genau dann ist $\gamma(w) = 2n - 1$, wenn $|I_x|$ gerade ist für alle $I_x \in I(w)$.*

Beweis. Sei $|I_x|$ gerade für alle $I_x \in I(w)$. Dann befinden sich die Lettern jedes $x \in \Sigma$ im Wort einmal auf einer ungeraden und einmal auf einer geraden Position. Färbe alle Buchstaben auf den ungeraden Positionen mit 0 und die auf den geraden Positionen mit 1 ein. Diese Färbung ist gültig und besitzt $2n - 1$ Farbwechsel. Sei f eine Färbung von w mit $2n - 1$ Farbwechseln. Dann gibt es kein Paar benachbarter Lettern ohne Farbwechsel, d. h. alle Lettern auf ungeraden Positionen sind zum Beispiel mit 0 eingefärbt und alle auf geraden mit 1. Dann muss für sich jedes $x \in \Sigma$ jeweils eine Letter auf gerader und eine auf ungerader Position befinden. Damit muss dann für alle Intervalle I_x gelten, dass $|I_x|$ ungerade ist. □

Die gerade im Beweis verwendete Färbung ist eine Greedy-Färbung. Wir hatten schon gezeigt, dass der Greedy-Algorithmus 0.5 garantiert. Da der Greedy-Algorithmus nun immer optimal färbt, wenn $\gamma(w) = 2n-1$ gilt, sind die im Beweis 3.6 verwendeten Beispiele eine untere Grenze der Leistungsgarantie. Wir können nochmal präzisieren:

Korollar 3.11. *Sei w ein Wort der Länge $2n$ und $n > 0$. Dann garantiert der Greedy-Algorithmus mindestens $n/(2n - 2)$ der Farbwechsel einer optimalen Färbung.*

Unmittelbar einzusehen ist, dass wenn w mindestens k Intervalle ungerader Länge besitzt und für jeweils zwei dieser Intervalle I_x und I_y stets $I_x \cap I_y = \emptyset$ gilt, dann eine Färbung von w höchstens $2n-1-k$ Farbwechsel besitzen kann. Wir können diese Aussage aber noch verbessern:

Definition 3.12. *Wir wollen eine Menge $Q(w)$, $Q(w) \subset I(w)$, als eine Menge unabhängiger ungerader Intervalle bezeichnen und durch das Erfüllen folgender Bedingungen definieren:*

- *Gilt $I_x \in Q(w)$ und $I_y \in Q(w)$ so folgt $I_x \subset I_y$ oder $I_x \supset I_y$ oder $I_x \cap I_y = \emptyset$.*

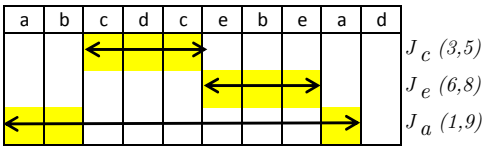

Abbildung 9: Wort w mit Menge unabhängiger ungerader Intervalle $Q(w) = \{I_a(1,9), I_c(3,5), I_e(6,8)\}$

– *Ist* $I_x \in Q(w)$*, so folgt* $|I_x \backslash P(I_x)|$ *ist ungerade, wobei* $P(I_x) = \bigcup_{I_y \in Q(w), I_y \subset I_x} I_y$.

Satz 3.13. *Für jedes Wort* w *gilt* $\gamma(w) \leq 2n - 1 - |Q(w)|$.

Beweis. Falls $Q(w) = \emptyset$ folgt |Q(w)|=0 und somit $\gamma(w) \leq 2n-1$, also gilt die Behauptung. Sei nun $Q(w) \neq \emptyset$. Wir wollen die Aussage als Induktion über die maximale Länge der Kette $I_u \subset I_v \subset ... \subset I_z$ mit I_u, I_v, ...,$I_z \in Q(w)$, die sich für $Q(w)$ bilden lässt, beweisen. Induktionsanfang: Sei die maximale Länge der Kette 1, also für alle $I_u, I_v \in Q(w)$ gilt $I_u \cap I_v = \emptyset$. Dann enthält w mindestens |Q(w)| Intervalle ungerader Länge, die sich nicht überlappen. Innerhalb jedes dieser Intervalle muss es für jede Färbung eine Stelle ohne Farbwechsel geben. Daraus folgt $\gamma \leq 2n - 1 - |Q(w)|$. Sei nun ein $Q(w)$ mit einer maximalen Kettenlänge vom $m+1$ gegeben. Gelte nun die Behauptung für eine Kettenlänge m. Wir wollen nun die Menge $Q^*(w)$ aus $Q(w)$ so ableiten, indem wir in allen Intervallketten $I_u \subset I_v \subset ... \subset I_z$ der Länge $m+1$ das Intervall I_z, also das, welches die anderen Intervalle als Untermenge enthält, entfernen. Dann besitzt $Q^*(w)$ eine maximale Kettenlänge m und auch $Q^*(w)$ ist eine Menge unabhängiger ungerader Intervalle. Wir können damit $\gamma(w) \leq 2n - 1 - |Q^*(w)|$ annehmen. Sei nun I_z ein Intervall mit $I_z \in Q(w)$ und $I_z \notin Q^*(w)$. Für alle Intervalle I_x gilt, dass die erste und zweite Letter von x unterschiedlich eingefärbt werden müssen. Die Menge $P(I_z)$ lässt sich auch aus einer Vereinigung von Intervallen konstruieren, die sich nicht überlappen. Da $|I_x \backslash P(I_x)|$ ungerade ist, muss es innerhalb von I_z eine Stelle ohne Farbwechsel geben, die außerhalb von $I_x \backslash P(I_x)$ liegt. Dies trifft alle $(|Q(w)| - |Q^*(w)|)$ Intervalle zu, die aus $Q^*(w)$ entfernt wurden. Daraus folgt $\gamma(w) \leq 2n - 1 - |Q^*(w)| - (|Q(w)| - |Q^*(w)|) = 2n - 1 - |Q(w)|$ und die Behauptung ist bewiesen. □

Ist 3 ein Teiler von n, können wir Wörter w konstruieren, so dass sich ein $Q(w)$ finden lässt, für das $|Q(w)| = \frac{2}{3}n$ gilt. Beispiele hierfür sind
$w = (x_1, x_2, x_1, x_3, x_2, x_3, \ldots, x_{n-2}, x_{n-1}, x_{n-2}, x_n, x_{n-1}, x_n)$ oder
$w = (x_1, x_2, x_3, x_2, x_4, x_5, x_6, x_5, \ldots, x_{n-1}, x_{n-2}, x_n, x_{n-2}, x_n, x_{n-1}, \ldots, x_6, x_4, x_3, x_1)$.
Somit können wir folgern:

Korollar 3.14. *Ist* 3 *ein Teiler von* n*, so gibt es Wörter* w *der Länge* $2n$ *mit* $\gamma(w) = 2n - 1 - \frac{2}{3}n = \frac{4}{3}n - 1$*. Ist* 3 *ein Teiler von* $n-1$*, gibt es* w *mit* $\gamma(w) = 2n - 1 - \frac{2}{3}(n-1) = \frac{4}{3}(n-1) + 1$ *und für* 3 *Teiler von* $n-2$ *existieren* w *mit* $\gamma(w) = 2n - 1 - (\frac{2}{3}(n-2) + 1) = \frac{4}{3}(n-2) + 2$.

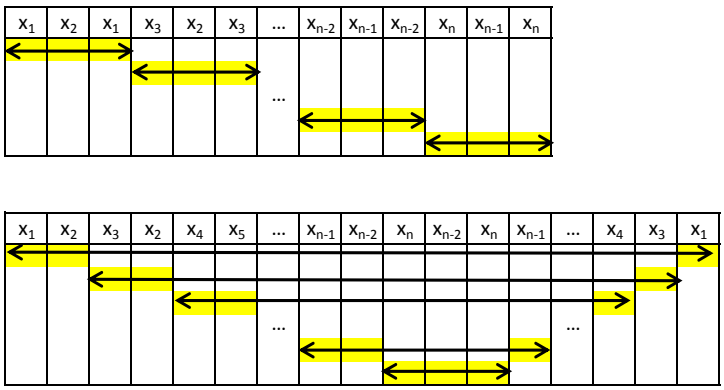

Abbildung 10: Zwei Wörter mit $\gamma(w) = \frac{4}{3}n - 1$. Die Pfeile stellen Intervalle dar. Die gelb markierten Bereiche der Intervalle sind ungerade lang und nicht durch gelb markierte Bereiche anderer Intervalle überdeckt. Hier muss es also ein Paar benachbarter Lettern ohne Farbwechsel geben.

Für $n > 1$ können wir also Wörter der Länge $2n$ finden, bei denen wir nicht mehr als $\frac{2}{3}$ der benachbarten Lettern mit einem Farbwechsel färben können.

3.2.3. Abschätzungen für die Anzahl der mindestens erreichbaren Farbwechsel

Nachdem wir eben eine Abschätzung von $\gamma(w)$ nach oben für ein gegebenes w gemacht haben, wollen wir für $\gamma(w)$ nun eine untere Grenze angeben. Dabei werden wir zeigen, dass die in Korollar 3.14 angegebenen Werte diese untere Grenze bilden, wir also jedes Wort der Länge $2n$ mit mindestens $\frac{4}{3}n - 1$ Farbwechseln einfärben können:

Satz 3.15. *Sei w ein Wort der Länge $2n$. Ist 3 ein Teiler von n, so lässt sich das Wort mit $\gamma(w) \geq \frac{4}{3}n - 1$ Farbwechsel einfärben. Ist 3 ein Teiler von $(n-1)$, so gilt $\gamma(w) \geq \frac{4}{3}(n-1) + 1$ und für 3 Teiler von $(n-2)$ gilt $\gamma(w) \geq \frac{4}{3}(n-2) + 2$.*

Wir wollen hierbei konstruktiv vorgehen, indem wir einen Algorithmus entwickeln, der gegebene Wörter entsprechend der Vermutung einfärbt. Dabei werden wir die einzelnen Lettern nicht mehr von links nach rechts einfärben, sondern wollen beide Lettern eines Buchstaben in einem Schritt färben. Ob ein Buchstabe schon eingefärbt werden kann, hängt nun davon ab, ob eine Färbung der Nachbarn der beiden Lettern des Buchstaben bereits erfolgt ist. Hierbei spielt es keine Rolle, mit welcher Farbe die Nachbarn gefärbt worden sind, sondern allein die Tatsache, dass gefärbt wurde. Die Idee dabei ist, dass wir Buchstaben dann färben wollen, wenn das Verhältnis zwischen den Farbwechselfehlern, die wir begehen müssen und den zu berücksichtigenden Randbedingungen, also den schon eingefärbten Nachbarn, möglichst günstig

ist. Wir wollen das an einem Beispiel veranschaulichen. Gegeben sei das teileingefärbte Wort $(\dots \widehat{a}_i, x, a_{i+2}, \dots, \widehat{a}_j, x, a_{j+2}, \dots)$. Das Dach über den Lettern bedeutet, dass hier eine Färbung vorhanden ist, ohne konkret die Farbe anzugeben. Wir wollen nun den Buchstaben x einfärben. Unabhängig von der konkreten Färbung der Nachbarn von x können wir x mit höchstens einem Fehler einfärben. Im Fall, dass beide gefärbte Nachbarn die selbe Farbe besitzen, ist ein Farbwechselfehler nicht zu vermeiden. Wir befriedigen also 2 Randbedingungen (schon eingefärbte Nachbarn) und müssen mit einem Fehler rechnen. Sei nun ein weiterer Nachbar von x eingefärbt und $(\dots \widehat{a}_i, x, \widehat{a}_{i+2}, \dots, \widehat{a}_j, x, a_{j+2}, \dots)$ das Wort mit der entsprechenden Teilfärbung. Auch hier lässt sich x unabhängig von der eigentlichen Färbung mit höchstens einem zusätzlichen Fehler färben, wie wir in den drei möglichen Fällen sehen können: $(\dots \dot{a}_i, \overline{x}, \dot{a}_{i+2}, \dots, \dot{a}_j, \dot{x}, a_{j+2}, \dots)$, $(\dots \dot{a}_i, \overline{x}, \dot{a}_{i+2}, \dots, \overline{a}_j, \dot{x}, a_{j+2},)$ und $(\dots \dot{a}_i, \overline{x}, \overline{a}_{i+2}, \dots, \overline{a}_j, \dot{x}, a_{j+2},)$. Dass heißt, wir befriedigen nun 3 Randbedingungen bei ebenfalls höchstens einem neuen Fehler.

In Tabelle 1 sind mögliche Umgebungen eines Buchstabens dargestellt und benannt. Dabei gibt die erste der zwei Ziffern im Namen des Typs die Anzahl der Randbedingen an und die zweite die maximal zu erwartende Anzahl von Farbwechselfehlern. So haben Buchstaben des Typs $a10$ einen schon eingefärbten Nachbarn und lassen sich immer mit null zusätzlichen Farbwechselfehlern einfärben. Bei der Definition der Umgebungstypen spielt es keine Rolle, ob an der ersten oder zweiten Letter oder ob vor oder nach der Letter die entsprechende Randbedingung anliegt. So liegt x in den beiden Teilfärbungen $(\dots \widehat{u}, x, \widehat{v}, \dots, \widehat{y}, x, z, \dots)$ und $(\dots u, x, \widehat{v}, \dots, \widehat{y}, x, \widehat{z}, \dots)$ auf dem Umgebungstyp $b31$. Die Typen sind in Tabelle 1 in drei verschiedene Kategorien eingeteilt. Die Typen $a10$, $b21$ und $c42$ befriedigen für die jeweils zu erwartende Anzahl an Fehlern maximal viele Randbedingungen, im Gegensatz zu den Typen $e00$, $f21$ und $g21$. Die Typen $s10$ und $t20$ sind Sonderfälle ohne Farbwechselfehler.
Die Angabe der maximalen Anzahl der Farbwechselfehler in der Tabelle bezieht sich darauf, dass im ungünstigsten Fall (im Bezug auf eine konkrete Färbung der Nachbarn) der Buchstabe x mit höchstens dem angegebenen Wert eingefärbt werden kann. In Tabelle 2 werden durch Fallunterscheidung die angegebenen Werte bewiesen. Als freien Rand wollen wir die Nachbarschaft von einer gefärbten und ungefärbten Letter definieren. Die Anzahl der freien Ränder wird später für einen Beweis gebraucht. In Tabelle 1 sind die Änderungen in der Anzahl der freien Ränder für den jeweiligen Umgebungstyp angegeben.
Mit den so definierten Umgebungstypen wollen wir einen Algorithmus erstellen, der zum Färben nur die Umgebungstypen $a10$, $b31$, c42, $s10$ und $t20$. verwendet. Ohne ihn hier spezifisch anzugeben, wollen wir so einen Algorithmus Gucote-Algorithmus (Greedy under consideration of the environment) nennen. Für die Funktionsweise des Algorithmus müssen wir noch spezifizieren, wie die Ränder des einzufärbenden Wortes zu behandeln sind. Hierzu wollen wir zwei zusätzliche Lettern a^* und z^* einführen. Dabei setzen wir a^* vor die erste reguläre Letter a_1 eines Wortes und z^* hinter die letzten Letter a_{2n}. Bevor wir nun mit der eigentlichen Färbung unseres Wortes beginnen, färben wir nun a^*. Die Letter z^* bleibt dagegen immer ungefärbt. In Tabelle 3 ist beispielhaft das Färben des Wortes $(a, b, c, b, d, c, e, d, f, e, a, f)$ durch so einen Algorithmus dargestellt. Das Wort wird mit einen Farbwechselfehler eingefärbt. Diese Färbung ist optimal, da zwischen den b genau eine Letter liegt und so ein Farbwechselfehler nicht vermeidbar ist. Die Färbung des Wortes mit dem Greedy-Algorithmus ergibt $(\dot{a}, \overline{b}, \dot{c}, \dot{b}, \overline{d}, \overline{c}, \dot{e}, \dot{d}, \overline{f}, \overline{e}, \overline{a}, \dot{f})$ und somit 5 Farbwechselfehler.

Tabelle 1: Übersicht Umgebungstypen

Typ	Teilwort vor Färbung von x	Anzahl Randbedingungen	Maximale Anzahl Farbwechselfehler bei optimaler Färbung x	Änderung Anzahl freier Ränder
$a10$	$(\dots \widehat{u}, x, v, \dots, y, x, z, \dots)$	1	0	+2
$b31$	$(\dots \widehat{u}, x, \widehat{v}, \dots, \widehat{y}, x, z, \dots)$	3	1	-2
$c42$	$(\dots \widehat{u}, x, \widehat{v}, \dots, \widehat{y}, x, \widehat{z}, \dots)$	4	2	-4
$e00$	$(\dots u, x, v, \dots, y, x, z, \dots)$	0	0	+4
$f21$	$(\dots \widehat{u}, x, v, \dots, \widehat{y}, x, z, \dots)$	2	1	+0
$g21$	$(\dots \widehat{u}, x, \widehat{v}, \dots, y, x, z, \dots)$	2	1	+0
$s10$	$(\dots \widehat{u}, x, x, v, \dots)$	1	0	+0
$t20$	$(\dots \widehat{u}, x, v, \dots, \widehat{u}, x, z, \dots)$	2	0	+0

Wir werden nun beweisen, dass wenn wir ein Wort mit einem Gucote-Algorithmus einfärben, dann die Anzahl der Farbwechsel aus der Vermutung erreicht wird.

Lemma 3.16. *Sei w ein Wort der Länge $2n$. Lässt sich das Wort nur mit $a10$-, $b31$-, $c42$-, $s10$- und $t20$-Färbungen einfärben, dann gilt: Ist 3 ein Teiler von n, so lässt sich das Wort mit $\gamma(w) \geq \frac{4}{3}n - 1$ Farbwechsel einfärben. Ist 3 ein Teiler von $(n-1)$, so folgt $\gamma(w) \geq \frac{4}{3}(n-1)+1$ und für 3 Teiler von $(n-2)$ gilt $\gamma(w) \geq \frac{4}{3}(n-2)+2$.*

Beweis. Wir wollen, wie eben schon besprochen, uns im zu färbenden Wort vor der ersten Letter eine bereits eingefärbte Letter (Stelle 0) und nach der letzten Letter eine nicht gefärbte Letter (Stelle $n+1$) hinzudenken, die auch nicht eingefärbt werden soll. Dieses Wort mit der Anfangsfärbung besitzt also einen freien Rand zwischen der nullten und ersten Letter. Sind alle n Buchstaben eingefärbt, besitzt das fertig gefärbte Wort wieder genau einen freien Rand, und zwar zwischen der letzten Letter des gegeben Wortes und der dahinter hinzugefügten Letter. Siehe Tabelle 1, kann sich je nach Umgebungstyp der Färbung die Anzahl der freien Ränder durch die Färbung eines Buchstabes ändern. Summiert man nun die Summe aller Änderungen der Anzahl der freien Ränder auf, muss man also null erhalten. Aus der Tabelle 1 ist ersichtlich, dass diejenigen Typen, die Fehler produzieren könnten, $b31$ und $c42$, die Typen sind, bei deren Färbung sich die Anzahl der freien Ränder verringert. Der einzige Typ, der die Summe der freien Ränder erhöht, ist $a10$. Um die Änderung der freien Ränder einer $b31$-Färbung auszugleichen, benötigen wir genau eine $a10$-Färbung, für den Ausgleich einer $c42$-Färbung dann genau zwei $a10$-Färbungen.

Bezeichnet n_a die Anzahl der $a10$-Färbungen, n_b die Anzahl der $b31$-Färbungen und n_c die Anzahl der $c42$-Färbungen, folgt damit $n_a = n_b + 2n_c$ und mit $n \geq n_a + n_b + n_c$ dann $n \geq 2n_b + 3n_c$. Mit E als obere Grenze der Wechselfehler beim Färben des gesamten Wortes erhalten wir weiterhin $E = n_b + 2n_c$, da bei der $b31$-Färbung maximal ein und bei der $c42$-Färbung höchstens zwei Farbwechselfehler entstehen.

Tabelle 2: Ermittelung Anzahl Färbungsfehler für Umgebungstypen

Typ	Teilwort vor Färbung von x	Optimale Färbung x im konkreten Fall	Anzahl Färbungsfehler
$a10$	$(\dots \widehat{u}, x, v, \dots, y, x, z, \dots)$	$(\dots \dot{u}, \overline{x}, v, \dots, y, \dot{x}, z, \dots)$	0
$b31$	$(\dots \widehat{u}, x, \widehat{v}, \dots, \widehat{y}, x, z, \dots)$	$(\dots \dot{u}, \overline{x}, \dot{v}, \dots, \dot{y}, \dot{x}, z, \dots)$	1
		$(\dots \dot{u}, \overline{x}, \dot{v}, \dots, \overline{y}, \dot{x}, z, \dots)$	0
		$(\dots \dot{u}, \overline{x}, \overline{v}, \dots, \overline{y}, \dot{x}, z, \dots)$	1
$c42$	$(\dots \widehat{u}, x, \widehat{v}, \dots, \widehat{y}, x, \widehat{z}, \dots)$	$(\dots \dot{u}, \overline{x}, \dot{v}, \dots, \dot{y}, \dot{x}, \dot{z}, \dots)$	2
		$(\dots \dot{u}, \overline{x}, \dot{v}, \dots, \dot{y}, \dot{x}, \overline{z}, \dots)$	1
		$(\dots \dot{u}, \overline{x}, \dot{v}, \dots, \overline{y}, \dot{x}, \overline{z}, \dots)$	0
		$(\dots \dot{u}, \overline{x}, \overline{v}, \dots, \dot{y}, \dot{x}, \overline{z}, \dots)$	1
$e00$	$(\dots u, x, v, \dots, y, x, z, \dots)$	$(\dots u, \overline{x}, v, \dots, y, \dot{x}, z, \dots)$	0
$f21$	$(\dots \widehat{u}, x, v, \dots, \widehat{y}, x, z, \dots)$	$(\dots \dot{u}, \overline{x}, v, \dots, \dot{y}, \dot{x}, z, \dots)$	1
		$(\dots \dot{u}, \overline{x}, v, \dots, \overline{y}, \dot{x}, z, \dots)$	0
$g21$	$(\dots \widehat{u}, x, \widehat{v}, \dots, y, x, z, \dots)$	$(\dots \dot{u}, \overline{x}, \dot{v}, \dots, y, \dot{x}, z, \dots)$	0
		$(\dots \dot{u}, \overline{x}, \overline{v}, \dots, y, \dot{x}, z, \dots)$	1
$s10$	$(\dots \widehat{u}, x, x, v, \dots)$	$(\dots \dot{u}, \overline{x}, \dot{x}, v, \dots)$	0
$t20$	$(\dots \widehat{u}, x, v, \dots, \widehat{u}, x, z, \dots)$	$(\dots \dot{u}, \overline{x}, v, \dots, \overline{u}, \dot{x}, z, \dots)$	0

Tabelle 3: Färbung des Wortes w=(a,b,c,b,d,c,e,d,f,e,a,f) mit einem Gucote-Algorithmus

$(a, b, c, b, d, c, e, d, f, e, a, f)$	ungefärbtes Wort
$(a^* a, b, c, b, d, c, e, d, f, e, a, f, z^*)$	Hinzufügen von a^* und z^*
$(\overline{a^*}, a, b, c, b, d, c, e, d, f, e, a, f, z^*)$	Färben von a^*
$(\overline{a^*}, \dot{a}, b, c, b, d, c, e, d, f, e, \overline{a}, f, z^*)$	$a(a10)$
$(\overline{a^*}, \dot{a}, \overline{b}, c, \dot{b}, d, c, e, d, f, e, \overline{a}, f, z^*)$	$b(a10)$
$(\overline{a^*}, \dot{a}, \overline{b}, c, \dot{b}, \overline{d}, c, e, \dot{d}, f, e, \overline{a}, f, z^*)$	$d(a10)$
$(\overline{a^*}, \dot{a}, \overline{b}, \overline{c}, \dot{b}, \overline{d}, \dot{c}, e, \dot{d}, f, e, \overline{a}, f, z^*)$	$c(b31)$
$(\overline{a^*}, \dot{a}, \overline{b}, \overline{c}, \dot{b}, \overline{d}, \dot{c}, \overline{e}, \dot{d}, f, \dot{e}, \overline{a}, f, z^*)$	$e(b31)$
$(\overline{a^*}, \dot{a}, \overline{b}, \overline{c}, \dot{b}, \overline{d}, \dot{c}, \overline{e}, \dot{d}, \overline{f}, \dot{e}, \overline{a}, \dot{f}, z^*)$	$f(b31)$
$(\dot{a}, \overline{b}, \overline{c}, \dot{b}, \overline{d}, \dot{c}, \overline{e}, \dot{d}, \overline{f}, \dot{e}, \overline{a}, \dot{f})$	Entfernen von a^* und z^*

Wollen wir nun E bei einem gegebenen n maximieren, sehen wir, dass wir dazu die Anzahl der $c42$-Färbungen maximieren müssen. Wir können das so erklären, dass wir hier für zwei potentielle Fehler drei Buchstaben benötigen (eine $c42$-Färbung und zwei $a10$-Färbungen zum Ausgleich der freien Ränder, also 1.5 Buchstaben pro Fehler), während ein potentieller Fehler durch eine $b31$-Färbung zwei Buchstaben benötigt. Damit folgt $n_a = \frac{2}{3}n$, $n_b = 0$ und $n_c = \frac{1}{3}n$ und damit $E = \frac{2}{3}n$, falls 3 ein Teiler von n ist, $n_a = \frac{2}{3}(n-1)$, $n_b = 0$ und $n_c = \frac{1}{3}(n-1)$ und damit $E = \frac{2}{3}(n-1)$, falls 3 ein Teiler von $n-1$ ist, und schließlich $n_a = \frac{2}{3}(n-2)+1$, $n_b = 1$ und $n_c = \frac{1}{3}(n-2)$, falls 3 ein Teiler von $n-2$ ist. Mit $\gamma(w) \geq$ 2n-1-E folgt die Behauptung. □

Nun ist noch zu zeigen, dass wir mit so einem Algorithmus tatsächlich alle Instanzen unseres Problems behandeln können. Hierzu werden wir zunächst eine schwächere Aussage beweisen und dazu die Wortmenge W' konstruieren.

Definition 3.17. *Sei W' die Menge der Wörter w' mit folgenden Eigenschaften:*

- *w' enthält kein Teilwort der Form (x, x).*
- *Gibt es ein Teilwort der Form (x, y, x) in w', so gibt es nur dann ein zweites dieser Form, falls es das Teilwort (y, x, y) ist.*
- *Es gibt keine Teilwortkombination der Form $(\ldots, a_i, v, y, a_{i+3}, x, y, a_{i+6}, \ldots, a_k, v, x)$, wobei zu beachten ist, dass das zweite x die letzte Letter im Wort ist.*

So liegt (a, b, b, a) nicht in W', da es das Teilwort (b, b) enthält. Das Wort $(a, b, c, b, d, e, d, c, e, a)$ liegt ebenfalls nicht in W', da es die Teilwörter (b, c, b) und (d, e, d) enthält. Dagegen ist (a, b, a, b) in W'.

Für Wörter aus W' wollen wir nun zeigen, dass ein wie eben skizzierten Algorithmus die Wörter komplett einfärben kann, ohne vorher zu stoppen.

Lemma 3.18. *Sei w' ein Wort aus W'. Dann lässt sich w' durch $a10$-, $b31$-, $c42$- und $t20$-Färbungen einfärben.*

Beweis. Durch den Ausschluss der Teilwörter (x, x) gibt es für Wörter aus W' keine Buchstaben auf den Umgebungstypen $s10$. Daher folgt die Vorüberlegung: Der Algorithmus würde stoppen, wenn alle noch nicht eingefärbten Buchstaben nicht mit Färbungen des Typs $a10$, $b31$, $c42$ oder $t20$ gefärbt werden können. Dies ist nur möglich, wenn alle noch nicht eingefärbten Buchstaben genau 0 oder 2 bereits eingefärbte Nachbarlettern (Randbedingungen) besitzen. Denn wenn eine Randbedingung vorliegt, können wir $a10$ färben, bei drei, $b31$, und bei vier, $c42$.

Dass bedeutet, bei einem Stopp liegen alle nicht eingefärbten Buchstaben auf den Umgebungstypen $e00$, $f21$ oder $g21$ (siehe Tabelle 1) und es gibt mindestens einen Buchstaben auf $f21$ oder $g21$.
Das Wort $(\overline{a}, \dot{b}, c, \overline{b}, d, c, d, \dot{a})$ mit Teilfärbung lässt sich beispielsweise nicht mit den im Algorithmus benutzten Färbungstypen weiter einfärben (wir hatten allerdings dieses Wort für W' ausgeschlossen, es enthält zwei Teilwörter der Form (x, y, x)). Hier liegt der Buchstabe c auf einer $g21$-Position und d auf einer $f21$-Position.
Außerdem wollen wir noch folgendes festlegen: Können wir während der Färbung eines Wortes durch den Algorithmus bei der Wahl des nächsten Buchstabens zwischen Buchstaben auf

verschieden Umgebungstypen wählen, wollen wir einen Buchstaben wählen, dessen Umgebung in der Liste ($t20$, $a10$, $b31$, $c42$) möglichst weit links steht. Liegt zum Beispiel das teileingefärbte Wort $(\widehat{a}, \widehat{b}, c, \widehat{a}, d, e, c, \widehat{b}, e, d)$ vor, dann werden wir im nächsten Schritt nicht c färben, welches auf $c31$ liegt, sondern e oder d, welche auf $a10$ liegen. Eine Priorität für die Auswahl von Buchstaben des gleichen Umgebungstyps geben wir hier noch nicht an.

Wir wollen nun annehmen, dass der Algorithmus bei der Färbung eines Wortes stoppt, bevor das Wort komplett eingefärbt ist. Es ist möglich, dass unmittelbar vor dem Stoppen noch $c42$ Färbungen möglich sind. Diese haben aber auf die Nachbarn des Buchstabens keine Einfluss, da diese bereits eingefärbt sind. Nun soll die letzte Färbung vor dem Stoppen, die nicht eine $c42$-Färbung ist, betrachtet werden, also die Färbung, nach der alle noch nicht gefärbten Buchstaben auf $e00$, $f21$, $g21$ oder $c42$ liegen.

Wir können dann davon ausgehen, dass dies eine $a10$- oder $t20$-Färbung ist: Sei y ein Buchstabe, der im letzten Schritt vor dem Stoppen des Algorithmus in eine $g21$- beziehungsweise $f21$-Position gebracht wird, so dass der Algorithmus blockiert. Dann ergibt sich für y folgende Situation $(\ldots, \overline{a}_i, y, a_{i+2}, \ldots, \overline{a}_j, y, a_{j+2}, \ldots)$, falls x in einer $f21$- Position liegt, oder $(\ldots, \overline{a}_i, y, \overline{a}_{i+2}, \ldots, a_j, y, a_{j+2}, \ldots)$, falls sich x in einer $g21$- Position befindet. Dann lag vor der Färbung entweder die Situation $(\ldots, \overline{a}_i, y, a_{i+2}, \ldots, a_j, y, a_{j+2}, \ldots)$ vor, in der sich y auf einer $a10$- Position befand. Da die $a10$- Färbung gegenüber der sonst noch infrage kommenden $b31$-Färbung im Algorithmus priorisiert ist, muss hier also die letzte Färbung vor Blockade des Algorithmus, die nicht eine $c42$-Färbung ist, ebenfalls eine $a10$-Färbung sein. Die zweite Möglichkeit ist, dass die Situation $(\ldots, a_i, y, a_{i+2}, \ldots, a_j, y, a_{j+2}, \ldots)$ vorgelegen hat. In dieser Situation gehören zwei der vier Lettern a_i, a_{i+2}, a_j und a_{j+2} zum gleichen Buchstaben, der nun eingefärbt wird. Dieser Buchstabe kann vor seiner Färbung höchstens zwei Randbedingungen besessen haben, so dass er nur durch eine $a10$- oder $t20$-Färbung gefärbt hätte werden können.

Wir wollen nun ein w' aus W' mit einen Algorithmus einfärben, der nur $a10$, $b31$,$c42$ und $t20$ Färbungen benutzt und annehmen, dass der Algorithmus stoppt. Seien dazu x der Buchstabe, der zuletzt mit einer $a10$- oder $t20$-Färbung eingefärbt wird, bevor der Algorithmus stoppt, und y, z und v, die Buchstaben, deren Lettern zu x benachbart sind und nicht mehr eingefärbt werden können. Wie vorher schon beschrieben, können wir weiterhin davon ausgehen, dass y, z und v zum Zeitpunkt des Stoppens genau 2 Randbedingungen, also Ränder zu eingefärbten Lettern, besitzen.

Durch das Stoppen des Algorithmus konnten nur $m < n$ Buchstaben eingefärbt werden. Mit Hilfe einer Fallunterscheidung wollen wir nun die verschiedenen Möglichkeiten, wie y, z und, falls vorhanden, v angeordnet sind, untersuchen. Dabei werden wir die Färbung von x wieder rückgängig machen und versuchen zu zeigen, dass man y, z und, falls vorhanden, v und schließlich auch x durch Änderung der Reihenfolge der Färbung doch einfärben kann. Also könnten mindestens $m + 1$ Buchstaben eingefärbt werden. Durch Induktion würde folgen, dass wir einen Algorithmus konstruieren können, der in der Lage ist, das Wort nur mit $a10$, $b31$, $c42$ und $t20$-Färbungen komplett zu färben.

Zu beachten ist, dass manche Situationen für w' aufgrund der Definition von W' nicht auftreten können und deshalb auch hier nicht untersucht werden. So ist zum Beispiel
$(\ldots \widehat{a}, \widehat{x}, y, y, \widehat{x}, a \ldots)$ für w' nicht möglich, da das Teilwort(x, x) enthalten ist. Außerdem ist zu berücksichtigen, dass es für die einzelnen Situationen verschiedene Varianten geben kann, die aber hinsichtlich des Stoppens des Algorithmus gleichwertig sind. Gleichwertig sind die Situa-

tionen, wenn die einzelnen Buchstaben in beiden Varianten die gleichen Nachbarlettern besitzen und für diese Nachbarlettern in beiden Varianten der gleiche Status der Färbung (gefärbt bzw. nicht gefärbt) vorliegt. So sind beispielsweise die Varianten
$(\dots, \widehat{a}_i, \widehat{x}, y, \widehat{a}_{i+3}, \dots, \widehat{a}_j, v, y, z, \widehat{x}, v, z, \widehat{a}_{j+7}, \dots)$ und
$(\dots, \widehat{a}_i, \widehat{x}, y, z, \widehat{a}_{i+4}, \dots, \widehat{a}_j, y, v, z, \widehat{x}, v, \widehat{a}_{j+6}, \dots)$ gleichwertig.

Wir wollen zunächst annehmen, dass x nicht zu den Randlettern a^* und z^* benachbart ist. Wir erhalten folgende Fallunterscheidung:

1. x hat einen gefärbten und die drei ungefärbten Buchstaben y, z und v als Nachbarn.
   Für diesen Fall können wir des Weiteren die folgenden 4 Situationen unterscheiden:

   a) y, z und v sind paarweise keine Nachbarn.
   Dann ergibt sich beim Stopp des Algorithmus die Situation
   $(\dots, \widehat{a}_i, \widehat{x}, y, a_{i+3}, \dots, a_j, z, \widehat{x}, v, a_{j+4}, \dots, \widehat{a}_k, y, a_{k+2}, \dots, \widehat{a}_l, z, a_{l+2}, \dots, \widehat{a}_m, v, a_{m+2}, \dots)$.
   Hier befinden sich y, z und v auf $f21$-Positionen und können nicht gefärbt werden. Wir nehmen nun die Färbung von x zurück. Somit erhalten wir
   $(\dots, \widehat{a}_i, x, y, a_{i+3}, \dots, a_j, z, x, v, a_{j+4}, \dots, \widehat{a}_k, y, a_{k+2}, \dots, \widehat{a}_l, z, a_{l+2}, \dots, \widehat{a}_m, v, a_{m+2}, \dots)$.
   Nun können unabhängig von einander y, z und v mit $a10$-Färbungen gefärbt werden, so dass wir schließlich
   $(\dots, \widehat{a}_i, x, \widehat{y}, a_{i+3}, \dots, a_j, \widehat{z}, x, \widehat{v}, a_{j+4}, \dots, \widehat{a}_k, \widehat{y}, a_{k+2}, \dots, \widehat{a}_l, \widehat{z}, a_{l+2}, \dots, \widehat{a}_m, \widehat{v}, a_{m+2}, \dots)$ erhalten. Hier kann dann auch x durch $c42$-Färbung gefärbt werden. Somit können wir mehr als m Buchstaben einfärben.

   b) y und z sind einfache Nachbarn, y und v beziehungsweise z und v sind keine Nachbarn.
   Es liegt hier die Situation
   $(\dots, \widehat{a}_i, \widehat{x}, y, a_{i+3}, \dots, a_j, z, \widehat{x}, v, a_{j+4}, \dots, \widehat{a}_k, y, z, \widehat{a}_{k+3}, \dots, \widehat{a}_m, v, a_{m+2}, \dots)$
   vor. Durch Zurücknehmen der Färbung von x erhalten wir
   $(\dots, \widehat{a}_i, x, y, a_{i+3}, \dots, a_j, z, x, v, a_{j+4}, \dots, \widehat{a}_k, y, z, \widehat{a}_{k+3}, \dots, \widehat{a}_m, v, a_{m+2}, \dots)$.
   Wir färben zunächst y und v mit $a10$ und es ergibt sich so
   $(\dots, \widehat{a}_i, x, \widehat{y}, a_{i+3}, \dots, a_j, z, x, \widehat{v}, a_{j+4}, \dots, \widehat{a}_k, \widehat{y}, z, \widehat{a}_{k+3}, \dots, \widehat{a}_m, \widehat{v}, a_{m+2}, \dots)$.
   Nun können wir x mit $b31$ einfärben für
   $(\dots, \widehat{a}_i, \widehat{x}, \widehat{y}, a_{i+3}, \dots, a_j, z, \widehat{x}, \widehat{v}, a_{j+4}, \dots, \widehat{a}_k, \widehat{y}, z, \widehat{a}_{k+3}, \dots, \widehat{a}_m, \widehat{v}, a_{m+2}, \dots)$.
   Dann kann z mit $b31$ gefärbt werden.

   c) y und z beziehungsweise y und v sind einfache Nachbarn, z und v sind keine Nachbarn.
   Dann erhalten wir die Situation
   $(\dots, \widehat{a}_i, \widehat{x}, y, \widehat{a}_{i+3}, \dots, a_j, z, \widehat{x}, v, a_{j+4}, \dots, \widehat{a}_k, z, y, v, \widehat{a}_{k+4}, \dots)$.
   Entfernen der Färbung von x ergibt
   $(\dots, \widehat{a}_i, x, y, \widehat{a}_{i+3}, \dots, a_j, z, x, v, a_{j+4}, \dots, \widehat{a}_k, z, y, v, \widehat{a}_{k+4}, \dots)$.
   Wir färben z und v mit $a10$ ein und erhalten
   $(\dots, \widehat{a}_i, x, y, \widehat{a}_{i+3}, \dots, a_j, \widehat{z}, x, \widehat{v}, a_{j+4}, \dots, \widehat{a}_k, \widehat{z}, y, \widehat{v}, \widehat{a}_{k+4}, \dots)$.
   Schließlich lässt sich nun x mit $b31$ einfärben zu
   $(\dots, \widehat{a}_i, \widehat{x}, y, \widehat{a}_{i+3}, \dots, a_j, \widehat{z}, \widehat{x}, \widehat{v}, a_{j+4}, \dots, \widehat{a}_k, \widehat{z}, y, \widehat{v}, \widehat{a}_{k+4}, \dots)$
   und anschließend y mit $c42$.

d) y, z und v sind paarweise Nachbarn.
Es liegt hier bei Blockade die Situation
$(\dots, \widehat{a}_i, \widehat{x}, y, \widehat{a}_{i+3}, \dots, \widehat{a}_j, v, y, z, \widehat{x}, v, z, \widehat{a}_{j+7}, \dots)$
vor. In der Situation 1.d) können wir auch nach dem Entfernen der Färbung von x nicht mehr als einen der Buchstaben x, y, z und v mit $a10$, $b31$ oder $c42$ einfärben. Wir können aber hier folgendermaßen argumentieren: Färben wir nun ein Wort, können wir bevor wir den Buchstaben wählen, der als nächstes gefärbt werden soll, alle Quadrupel von Buchstaben ermitteln, die sich in einer Konstellation nach Situation 1.d) für x, y, z und v befinden, höchstens zwei eingefärbte Buchstaben besitzen und bei denen mindestens schon eine Nachbarletter zu den vier Buchstaben des Quadrupels eingefärbt ist. Ein solches Quadrupel wollen wir als gefährliches Quadrupel bezeichnen. Ein Beispiel für ein gefährliches Quadrupel ist $\{x, y, z, v\}$ mit $(\dots, \widehat{a}_i, \widehat{x}, y, \widehat{a}_{i+3}, \dots, a_j, v, y, z, \widehat{x}, v, z, a_{j+7}, \dots)$.
Mit x ist hier ein Buchstabe (also weniger als zwei) eingefärbt und es gibt mit $\widehat{a}_i$ und $\widehat{a}_{i+3}$ schon zwei eingefärbte Nachbarlettern zu den vier Buchstaben.
Wollen wir einen Buchstaben färben und existieren gefährliche Quadrupel, wählen wir dasjenige gefährliche Quadrupel, welches in Summe die meisten schon eingefärbten Nachbarlettern besitzt und färben einen Buchstaben daraus mit $a10$ ein. Dies ist immer möglich, solange noch nicht alle 4 Nachbarlettern zu x, y, z und v eingefärbt sind: Falls noch kein Buchstabe aus dem Quadrupel eingefärbt ist, färbt man einen ein, der eine gefärbte Nachbarletter besitzt, ist schon ein Buchstabe eingefärbt, färbt man einen Buchstaben, dessen Nachbarletter außerhalb des Quadrupels noch nicht gefärbt ist. Ist beispielsweise also $(\dots, \widehat{a}_i, x, y, \widehat{a}_{i+3}, \dots, a_j, v, y, z, x, v, z, a_{j+7}, \dots)$ gegeben, so können wir x zum Färben auswählen, da die Nachbarletter a_i bereits gefärbt ist. Wir erhalten $(\dots, \widehat{a}_i, \widehat{x}, y, \widehat{a}_{i+3}, \dots, a_j, v, y, z, \widehat{x}, v, z, a_{j+7}, \dots)$. Hier ist nun ein Buchstabe des Quadrupels eingefärbt. Wir sehen, dass die Nachbarletter a_{j+7} von z noch nicht gefärbt ist, so dass wir im nächsten Schritt z mit $a10$ färben können.

Wir müssen noch zeigen, dass wir mindestens zwei Buchstaben aus jedem gefährlichen Quadrupel zum Färben auswählen können, bevor alle Nachbarlettern des Quadrupels gefärbt sind. Dies wird durch folgende Überlegungen gezeigt: Sei während des Färbens eines Wortes die Situation gegeben, dass kein gefährliches Quadrupel vorliegt. Dies ist zum Beispiel vor dem ersten Färben eines Buchstabens eines Worts der Fall. Wird im weiteren Verlauf der Färbung des Wortes nun erstmals ein Buchstabe gefärbt, der zu Buchstaben innerhalb gefährlicher Quadrupel benachbart ist, können höchstens 3 der 4 Nachbarn desselben Quadrupels gefärbt werden. Als Summe können 4 Nachbarlettern zu gefährlichen Quadrupeln gefärbt werden. Als Beispiel können wir die Konstellation
$(\dots, a_i, e, f, a_{i+3}, \dots, a_j, h, f, g, e, h, g, \widehat{a}_{j+7}, x, y, \widehat{a}_{j+10}, v, y, z, x, v, z, a_{j+17}, \dots)$
betrachten, bei der gerade der Buchstabe $a_{j+7} = a_{j+10}$ gefärbt wurde. Es wurde ein Nachbar des Quadrupels $\{e, f, g, h\}$ und 3 des Quadrupels $\{x, y, z, v\}$ gefärbt.

Bei der ersten Färbung eines Buchstaben eines gefährlichen Quadrupels entsteht keine Färbung eines Nachbarn außerhalb des eigenen Quadrupels, bei der Färbung eines zweiten Buchstabens höchstens eine gefärbte Nachbarletter zu einem andern gefährlichen Quadrupel. Gleichzeitig ist das Quadrupel des gefärbten Buchstabens nun nicht mehr gefährlich. Daraus folgt, dass die Summe aller eingefärbten Nachbarn zu gefährlichen Quadrupeln

auch im Weiteren nicht größer als vier werden kann. Damit alle 4 Nachbarn zu einem gefährlichen Quadrupel gefärbt werden, müsste es aber einen Schritt geben, bei dem 2 gefährliche Quadrupel gleichzeitig jeweils 3 gefärbte Nachbarn besitzen, dass heißt, man würde insgesamt 6 gefärbte Nachbarn zu gefährlichen Quadrupeln benötigen. Denn liegt nur ein gefährliches Quadrupel mit 3 gefärbten Nachbarn vor, können wir dieses durch einfärben beseitigen. Daraus folgt, dass wir durch geeignete Auswahl der als nächstes zu färbenden Buchstaben Situation 1.d) immer vermeiden können.

e) y und z sind doppelte Nachbarn.
Dann ergibt sich beim Stopp des Algorithmus die Situation
$(\dots, \widehat{a}_i, \widehat{x}, y, z, \widehat{a}_{i+4}, \dots, \widehat{a}_j, y, z, \widehat{x}, v, a_{j+5}, \dots, \widehat{a}_k, v, a_{k+2}, \dots)$.
Wieder nehmen wir die Färbung von x zurück und erhalten
$(\dots, \widehat{a}_i, x, y, z, \widehat{a}_{i+4}, \dots, \widehat{a}_j, y, z, x, v, a_{j+5}, \dots, \widehat{a}_k, v, a_{k+2}, \dots)$.
Nun färben wir y und v mit $a10$ für
$(\dots, \widehat{a}_i, x, \widehat{y}, z, \widehat{a}_{i+4}, \dots, \widehat{a}_j, \widehat{y}, z, x, \widehat{v}, a_{j+5}, \dots, \widehat{a}_k, \widehat{v}, a_{k+2}, \dots)$
und dann x mit $b31$ für
$(\dots, \widehat{a}_i, \widehat{x}, \widehat{y}, z, \widehat{a}_{i+4}, \dots, \widehat{a}_j, \widehat{y}, z, \widehat{x}, \widehat{v}, a_{j+5}, \dots, \widehat{a}_k, \widehat{v}, a_{k+2}, \dots)$.
Schließlich können wir z mit $c42$ färben.

2. x hat einen gefärbten und die zwei ungefärbten Buchstaben y und z als Nachbarn. Dabei ist eine Letter y doppelter Nachbar zu x.
Für diesen Fall können wir die folgenden 2 Situationen unterscheiden:

   a) y und z sind keine Nachbarn. Es liegt die Situation
   $(\dots, \widehat{a}_i, \widehat{x}, y, \widehat{x}, z, \widehat{a}_{i+5}, \dots, a_j, y, a_{j+2}, \dots, a_k, z, a_{k+2}, \dots)$ vor.
   Tatsächlich können wir hier auch nach Entfernen der Färbung von x zunächst nicht weiter einfärben. Wir werden die Situation aber im Anschluss der Fallunterscheidung diskutieren, wollen aber noch festhalten, dass es innerhalb a_{i+6} bis a_j eine Letter a_l geben muss, die einen eingefärbten Nachbarn besitzt, aber der zu a_l gehörige Buchstabe nicht eingefärbt werden kann.

   b) y und z sind Nachbarn. Dies entspricht der Situation
   $(\dots, \widehat{a}_i, \widehat{x}, y, \widehat{x}, z, \widehat{a}_{i+5}, \dots, a_j, y, z, a_{j+3}, \dots)$.
   Wir entfernen die Färbung von x und erhalten
   $(\dots, \widehat{a}_i, x, y, x, z, \widehat{a}_{i+5}, \dots, a_j, y, z, a_{j+3}, \dots)$.
   Nun können wir z mit $a10$ färben und bekommen
   $(\dots, \widehat{a}_i, x, y, x, \widehat{z}, \widehat{a}_{i+5}, \dots, a_j, y, \widehat{z}, a_{j+3}, \dots)$.
   Jetzt kann y mit $a10$ eingefärbt werden zu
   $(\dots, \widehat{a}_i, x, \widehat{y}, x, \widehat{z}, \widehat{a}_{i+5}, \dots, a_j, \widehat{y}, \widehat{z}, a_{j+3}, \dots)$.
   Hier kann dann x mit $c42$ gefärbt werden.

3. x hat einen gefärbten und die zwei ungefärbten Buchstaben y und z als Nachbarn. Dabei ist eine Letter x doppelter Nachbar zu z.
Hier lassen sich die folgenden 2 Situationen unterscheiden:

   a) y und z sind keine Nachbarn.

Es liegt die Situation
$(\ldots, \widehat{a}_i, \widehat{x}, y, \widehat{a}_{i+4}, \ldots, a_j, z, \widehat{x}, z, a_{j+2}, \ldots, a_k, y, a_{k+2}, \ldots)$ vor.
Nach dem Entfernen der Färbung von x können wir nur y, aber keinen weiteren Buchstaben einfärben. Auch hier werden wir die Situation im Anschluss der Fallunterscheidung diskutieren, wollen aber noch festhalten, dass es innerhalb a_{i+5} bis a_j eine Letter a_l geben muss, die einen eingefärbten Nachbarn besitzt, aber der zu a_l gehörige Buchstabe nicht eingefärbt werden kann.

b) y und z sind Nachbarn.
Dann ergibt sich die Situation
$(\ldots, \widehat{a}_i, \widehat{x}, y, \widehat{a}_{i+4}, \ldots, a_j, z, \widehat{x}, z, y, a_{j+5}, \ldots)$.
Wir entfernen die Färbung von x und erhalten
$(\ldots, \widehat{a}_i, x, y, \widehat{a}_{i+4}, \ldots, a_j, z, x, z, y, a_{j+5}, \ldots)$.
Jetzt färben wir y ein, so dass sich
$(\ldots, \widehat{a}_i, x, \widehat{y}, \widehat{a}_{i+4}, \ldots, a_j, z, x, z, \widehat{y}, a_{j+5}, \ldots)$ ergibt.
Nun können wir z einfärben zu
$(\ldots, \widehat{a}_i, x, \widehat{y}, \widehat{a}_{i+4}, \ldots, a_j, \widehat{z}, x, \widehat{z}, \widehat{y}, a_{j+5}, \ldots)$.
Schließlich kann x mit $c42$ gefärbt werden.

4. x hat zwei gefärbte Lettern und die zwei ungefärbten Buchstaben y und z als Nachbarn. Aufgrund der zwei gefärbten Nachbarlettern muss die Färbung von x eine $t20$-Färbung sein. Hier lassen sich die folgenden 3 Situationen unterscheiden:

a) y und z sind keine Nachbarn.
Es ergibt sich die Situation
$(\ldots, \widehat{a}_i, \widehat{x}, y, \widehat{a}_{i+3}, \ldots, \widehat{a}_j, \widehat{x}, z, \widehat{a}_{j+3}, \ldots, a_k, y, a_{k+2}, \ldots, a_l, z, a_{l+2}, \ldots)$.
Hier entfernen wir die Färbung von x und bekommen
$(\ldots, \widehat{a}_i, x, y, \widehat{a}_{i+3}, \ldots, \widehat{a}_j, x, z, \widehat{a}_{j+3}, \ldots, a_k, y, a_{k+2}, \ldots, a_l, z, a_{l+2}, \ldots)$.
Nun können wir y und z mit $a10$ einfärben und wir erhalten
$(\ldots, \widehat{a}_i, x, \widehat{y}, \widehat{a}_{i+3}, \ldots, \widehat{a}_j, x, \widehat{z}, \widehat{a}_{j+3}, \ldots, a_k, \widehat{y}, a_{k+2}, \ldots, a_l, \widehat{z}, a_{l+2}, \ldots)$.
Jetzt kann x mit $c42$ gefärbt werden.

b) y und z sind einfache Nachbarn.
Dann ergibt sich die Situation
$(\ldots, \widehat{a}_i, \widehat{x}, y, \widehat{a}_{i+3}, \ldots, \widehat{a}_j, \widehat{x}, z, \widehat{a}_{j+3}, \ldots, a_k, y, z, a_{k+3}, \ldots)$.
Wir entfernen die Färbung von x und erhalten
$(\ldots, \widehat{a}_i, x, y, \widehat{a}_{i+3}, \ldots, \widehat{a}_j, x, z, \widehat{a}_{j+3}, \ldots, a_k, y, z, a_{k+3}, \ldots)$.
Jetzt färben wir y mit $a10$ ein, so dass sich
$(\ldots, \widehat{a}_i, x, \widehat{y}, \widehat{a}_{i+3}, \ldots, \widehat{a}_j, x, z, \widehat{a}_{j+3}, \ldots, a_k, \widehat{y}, z, a_{k+3}, \ldots)$ ergibt.
Nun kann x mit $b31$ gefärbt werden:
$(\ldots, \widehat{a}_i, \widehat{x}, \widehat{y}, \widehat{a}_{i+3}, \ldots, \widehat{a}_j, \widehat{x}, z, \widehat{a}_{j+3}, \ldots, a_k, \widehat{y}, z, a_{k+3}, \ldots)$.
Hier können wir z mit $b31$ färben.

c) y und z sind doppelte Nachbarn.
Es liegt die Situation
$(\ldots, \widehat{a}_i, \widehat{x}, y, z, \widehat{a}_{i+3}, \ldots, \widehat{a}_j, \widehat{x}, z, y, \widehat{a}_{j+4}, \ldots)$ vor.
Wir entfernen die Färbung von x und es ergibt sich

$(\ldots, \widehat{a}_i, x, y, z, \widehat{a}_{i+3}, \ldots, \widehat{a}_j, x, z, y, \widehat{a}_{j+4}, \ldots)$.
Jetzt färben wir y ein, so dass sich wir
$(\ldots, \widehat{a}_i, x, \widehat{y}, z, \widehat{a}_{i+3}, \ldots, \widehat{a}_j, x, z, \widehat{y}, \widehat{a}_{j+4}, \ldots)$ erhalten.
Nun besteht die Möglichkeit x mit $b31$ zu färben:
$(\ldots, \widehat{a}_i, \widehat{x}, \widehat{y}, z, \widehat{a}_{i+3}, \ldots, \widehat{a}_j, \widehat{x}, z, \widehat{y}, \widehat{a}_{j+4}, \ldots)$.
Hier können wir z mit $c42$ färben.

Bisher hatten wir vorausgesetzt, dass x nicht zu einer der beiden Randlettern a^* und z^* benachbart ist. Die Letter a^* am Anfang des Wortes ist schon zu Beginn eingefärbt und wir können danach mindestens eine weitere Färbung vornehmen, so dass hier nichts weiter zu zeigen ist. Die Letter z^* am Ende des Wortes wird nicht eingefärbt und kann zu x benachbart sein. Dadurch dass wir z^* während der kompletten Färbung des Wortes im Gegensatz zu den bisher betrachteten Nachbarn von x nicht färben, müssen nun noch weitere Fälle untersucht werden.

5. x hat einen gefärbten und die drei ungefärbten Buchstaben y, v und z^* als Nachbarn.
   Für diesen Fall können wir des Weiteren die folgenden 3 Situationen unterscheiden:

   a) y und v sind keine Nachbarn.
   Dann ergibt sich beim Stopp des Algorithmus die Situation
   $(\ldots, \widehat{a}_i, \widehat{x}, y, a_{i+3}, \ldots, \widehat{a}_j, y, a_{j+2}, \ldots, \widehat{a}_k, v, a_{k+2} \ldots, a_l, v, \widehat{x}, z^*)$.
   Hier befinden sich y und v auf $f21$-Positionen und können nicht gefärbt werden. Wir nehmen nun die Färbung von x zurück. Somit erhalten wir
   $(\ldots, \widehat{a}_i, x, y, a_{i+3}, \ldots, \widehat{a}_j, y, a_{j+2}, \ldots, \widehat{a}_k, v, a_{k+2} \ldots, a_l, v, x, z^*)$.
   Nun können unabhängig von einander y und v mit $a10$-Färbungen gefärbt werden, so dass wir schließlich
   $(\ldots, \widehat{a}_i, x, \widehat{y}, a_{i+3}, \ldots, \widehat{a}_j, \widehat{y}, a_{j+2}, \ldots, \widehat{a}_k, \widehat{v}, a_{k+2} \ldots, a_l, \widehat{v}, x, z^*)$.
   erhalten. Hier kann dann auch x durch $b31$-Färbung gefärbt werden.

   b) y und v sind einfache Nachbarn.
   Es liegt hier die Situation
   $(\ldots, \widehat{a}_i, \widehat{x}, y, a_{i+3}, \ldots, \widehat{a}_j, y, v, \widehat{a}_{j+3} \ldots, a_k, v, \widehat{x}, z^*)$ vor.
   Zunächst bemerken wir, dass a_i und a_j oder a_i und a_{j+3} nicht zusammenfallen können, da wir dies bei der Definition von W' ausgeschlossen hatten. Durch Zurücknehmen der Färbung von x erhalten wir
   $(\ldots, \widehat{a}_i, x, y, a_{i+3}, \ldots, \widehat{a}_j, y, v, \widehat{a}_{j+3} \ldots, a_k, v, x, z^*)$.
   In dieser Situation können wir auch nach dem Entfernen der Färbung von x nicht mehr als einen der Buchstaben x, y, oder v mit $a10$, $b31$ oder $c42$ einfärben. Wir können aber wie in Situation 1.d) argumentieren. Sobald nun durch die Färbung einer der Nachbarlettern von x, y oder v eine Randbedingung hinzu kommt, wollen wir $\{x, y, v, z^*\}$ wie ein gefährliches Quadrupel nach Situation 1.d) behandeln. Sei zunächst keine der Nachbarlettern zu den Buchstaben des Quadrupels eingefärbt. Durch eine Färbung eines Buchstaben können höchstens 2 der äußeren Nachbarlettern des Quadrupels in einem Schritt eingefärbt werden. Wir färben dann wieder einen der Buchstaben aus $\{x, y, z\}$, zunächst einen, der schon einen eingefärbten Nachbarn besitzt, und danach einen ohne äußeren eingefärbten Nachbarn. Nachdem 2 Buchstaben aus $\{x, y, v\}$ eingefärbt sind, ist das Quadrupel $\{x, y, v, z^*\}$ nicht mehr gefährlich. Wir wollen dies an einem Beispiel demonstrieren. Sei

dazu
$(\ldots, \widehat{a}_i, x, y, a_{i+3}, \ldots, \widehat{a}_j, y, v, a_{j+3}, \ldots, a_l, v, x, z^*)$ gegeben. Hier wurde der Buchstabe $a_i = a_j$ eingefärbt und damit das Quadrupel gefährlich. Zum Färben wählen wir x als einen Buchstaben mit schon eingefärbten Nachbarn und erhalten
$(\ldots, \widehat{a}_i, \widehat{x}, y, a_{i+3}, \ldots, \widehat{a}_j, y, v, a_{j+3}, \ldots, a_l, v, \widehat{x}, z^*)$. Nun können wir v als Buchstaben ohne äußere Randbedingung zu
$(\ldots, \widehat{a}_i, \widehat{x}, y, a_{i+3}, \ldots, \widehat{a}_j, y, \widehat{v}, a_{j+3}, \ldots, a_l, \widehat{v}, \widehat{x}, z^*)$ färben und das Quadrupel ist nicht mehr gefährlich. Somit bleibt die Argumentation von 1.d) erhalten und wir können die Situation 5.b) immer vermeiden.

c) y und v sind doppelte Nachbarn.
Dann ergibt sich beim Stopp des Algorithmus die Situation
$(\ldots, \widehat{a}_i, \widehat{x}, y, v, \widehat{a}_{i+4}, \ldots, \widehat{a}_j, y, v, \widehat{x}, z^*)$.
Wieder nehmen wir die Färbung von x zurück und erhalten
$(\ldots, \widehat{a}_i, x, y, v, \widehat{a}_{i+4}, \ldots, \widehat{a}_j, y, v, x, z^*)$.
Nun färben wir y mit $a10$ für
$(\ldots, \widehat{a}_i, x, \widehat{y}, v, \widehat{a}_{i+4}, \ldots, \widehat{a}_j, \widehat{y}, v, x, z^*)$
und dann v mit $b31$ für
$(\ldots, \widehat{a}_i, x, \widehat{y}, \widehat{v}, \widehat{a}_{i+4}, \ldots, \widehat{a}_j, \widehat{y}, \widehat{v}, x, z^*)$.
Schließlich können wir x mit $b31$ färben.

6. x hat einen gefärbten und die zwei ungefärbten Buchstaben y und z^* als Nachbarn. Dabei ist eine Letter y doppelter Nachbar zu x.
Für diesen Fall ergibt sich die folgende Situation.

   a) y und z^* sind keine Nachbarn. Es liegt die Situation
   $(\ldots, a_i, y, a_{i+2}, \ldots, \widehat{a}_j, \widehat{x}, y, \widehat{x}, z^*)$ vor.
   Tatsächlich können wir hier auch nach Entfernen der Färbung von x zunächst nicht weiter einfärben. Wir werden die Situation wieder im Anschluss der Fallunterscheidung diskutieren, wollen aber noch festhalten, dass es innerhalb a_{i+2} bis a_{j-1} eine Letter a_l geben muss, die einen eingefärbten Nachbarn besitzt, aber der zu a_l gehörige Buchstabe nicht eingefärbt werden kann.

7. x hat einen gefärbten und die zwei ungefärbten Buchstaben y und z^* als Nachbarn. Dabei ist eine Letter x doppelter Nachbar zu y.

   a) y und z sind keine Nachbarn.
   Es liegt die Situation
   $(\ldots, a_i, y, \widehat{x}, y, a_{i+4}, \ldots, \widehat{a}_j, \widehat{x}, z^*)$ vor.
   Nach dem Entfernen der Färbung von x können wir keinen weiteren Buchstaben einfärben. Auch hier werden wir die Situation im Anschluss an die Fallunterscheidung diskutieren, wollen aber noch festhalten, dass es innerhalb a_{i+4} bis a_{j-1} eine Letter a_l geben muss, die einen eingefärbten Nachbarn besitzt, aber der zu a_l gehörige Buchstabe nicht eingefärbt werden kann.

8. x hat zwei gefärbte Lettern und die zwei ungefärbten Buchstaben y und z^* als Nachbarn.

Aufgrund der zwei gefärbten Nachbarlettern muss die Färbung von x eine $t20$-Färbung sein. Die folgende Situation ist möglich:

a) y und z^* sind keine Nachbarn.
Dann ergibt sich
$(\ldots, \widehat{a}_i, \widehat{x}, y, \widehat{a}_{i+3}, \ldots, a_j, y, a_{j+2}, \ldots, \widehat{a}_k, \widehat{x}, z^*)$.
Hier entfernen wir die Färbung von x und bekommen
$(\ldots, \widehat{a}_i, x, y, \widehat{a}_{i+3}, \ldots, a_j, y, a_{j+2}, \ldots, \widehat{a}_k, x, z^*)$.
Nun können wir y einfärben und wir erhalten
$(\ldots, \widehat{a}_i, x, \widehat{y}, \widehat{a}_{i+3}, \ldots, a_j, \widehat{y}, a_{j+2}, \ldots, \widehat{a}_k, x, z^*)$.
Jetzt kann x mit $b31$ gefärbt werden.

Damit ist die Fallunterscheidung abgeschlossen. Wir müssen aber noch die Situationen 2.a), 3.a), 6.a) und 7.a) diskutieren. Wir hatten festgestellt, dass es in beiden Situationen auch außerhalb der in der Situation betrachteten Buchstaben weitere Buchstaben gibt, die gefärbte Nachbarn besitzen, aber selbst nicht gefärbt sind. Wir wollen diese Buchstaben in der Menge A zusammenfassen. Die Buchstaben der Menge A können aber auch nicht eingefärbt werden, da wir ja vorausgesetzt hatten, dass unser Algorithmus gestoppt hat. Sie müssen sich also in einer der Situationen befinden, die wir eben in der Fallunterscheidung betrachtet hatten. Da wir für alle Situationen bis auf 2.a), 3.a), 6.a) und 7.a) zeigen konnten, dass die jeweils beteiligten Buchstaben eingefärbt werden können, müssen sich die Buchstaben der Menge A ebenfalls in den Situationen 2.a), 3.a), 6.a) oder 7.a) befinden. Dies ist allerdings nicht möglich, denn die 4 Situationen beinhalten ein Teilwort des Typs (x, y, x), so dass im Wort mindestens zwei solcher Teilwörter enthalten sein müssten, die sich nicht überlappen. Das hatten wir allerdings bei der Definition von W' ausgeschlossen. Daraus folgt, dass es keine Blockade des Algorithmus durch die Situationen 2.a), 3.a), 6.a) oder 7.a) geben kann.
Damit ist gezeigt, dass wir in allen Fällen eine Blockade beim Einfärben eines Wortes aus W' durch geeignete Wahl des als nächstes zu färbenden Buchstabens vermeiden können. Somit lässt sich jedes Wort aus W' durch einen Algorithmus einfärben, der nur $a10$-, $b31$-, $c42$- und $t20$-Färbungen benutzt. □

Mit Lemma 3.16 folgt sich unmittelbar:

Korollar 3.19. *Sei w' ein Wort der Länge $2n$ und $w' \in W'$. Ist 3 ein Teiler von n, so lässt sich das Wort mit $\gamma(w') \geq \frac{4}{3}n - 1$ Farbwechsel einfärben. Ist 3 ein Teiler von $(n-1)$, so folgt $\gamma(w') \geq \frac{4}{3}(n-1) + 1$ und für 3 Teiler von $(n-2)$ gilt $\gamma(w') \geq \frac{4}{3}(n-2) + 2$.*

Definition 3.20. *Sei W'' die Menge der Wörter w'' mit folgenden Eigenschaften:*

- *w'' enthält kein Teilwort der Form (x, x).*
- *Gibt es ein Teilwort der Form (x, y, x) in w'', so gibt es nur dann ein zweites dieser Form, falls es das Teilwort (y, x, y) ist.*

Wir bemerken, dass die Menge W'' alle Wörter der Menge W' umfasst, zuzüglich der Wörter, die in W' nicht enthalten sind, weil sie eine Teilwortkombination der Form
$(\ldots, a_i, v, y, a_{i+3}, x, y, a_{i+6}, \ldots, a_k, v, x)$ enthalten.

Lemma 3.21. *Sei w'' ein Wort der Länge $2n$ und $w'' \in W''$. Ist 3 ein Teiler von n, so lässt sich das Wort mit $\gamma(w'') \geq \frac{4}{3}n - 1$ Farbwechsel einfärben. Ist 3 ein Teiler von $(n-1)$, so folgt $\gamma(w'') \geq \frac{4}{3}(n-1) + 1$ und für 3 Teiler von $(n-2)$ gilt $\gamma(w'') \geq \frac{4}{3}(n-2) + 2$.*

Beweis. Wir können wie im Beweis für Lemma 3.18 argumentieren, nur die Situation 5.b) müssen wir neu beurteilen, so dass sich bei Blockade nun die Situation
$(\ldots, \widehat{a}_i, v, y, \widehat{a}_{i+3}, \widehat{x}, y, a_{i+6}, \ldots, a_k, v, \widehat{x}, z^*)$ ergibt, die wir für Wörter aus W'' ausgeschlossen hatten. Durch Zurücknehmen der Färbung von x erhalten wir
$(\ldots, \widehat{a}_i, v, y, \widehat{a}_{i+3}, x, y, a_{i+6}, \ldots, a_k, v, x, z^*)$. Wir wollen nun wieder den Zustand vor der ersten Färbung von a_i oder a_{i+3} annehmen. Gilt $a_i \neq a_{i+3}$, können a_i und a_{i+3} nicht gleichzeitig gefärbt werden und wir können die Situation wie im Beweis für Lemma 3.18, Situation 5.b), lösen, indem wir $\{x, y, v, z^*\}$ als gefährliches Quadrupel betrachten, nachdem die erste dieser beiden Lettern gefärbt wurde. Wird so beispielsweise a_i als erste der beiden Lettern a_i und a_{i+3} gefärbt, können wir v und danach x mit $a10$ färben und das Quadrupel ist nicht mehr gefährlich.
Als letztes müssen wir noch die Variante untersuchen, dass $a_i = a_{i+3}$ gilt. Dann sind a_i und a_{i+3} unterschiedlich gefärbt und bei Blockade liegt
$(\ldots, \dot{a}_i, v, y, \overline{a}_{i+3}, \dot{x}, y, a_{i+6}, \ldots, a_k, v, \overline{x}, z^*)$ vor.
Wir färben v ohne Farbwechselfehler und erhalten
$(\ldots, \dot{a}_i, \overline{v}, y, \overline{a}_{i+3}, \dot{x}, y, a_{i+6}, \ldots, a_k, \dot{v}, \overline{x}, z^*)$. Damit wurde die Blockade ohne Farbwechselfehler aufgelöst, das Wort kann nun weiter mit $a10$-, $b31$, $c42$ und $t20$-Färbungen gefärbt und die Behauptung ist bewiesen. □

Wie wollen nun zunächst zeigen, dass Korollar 3.19 auch für Wörter gilt, die ein Teilwort (x, x) oder zwei Teilwörter des Typs (x, y, x) enthalten. Dazu die folgenden drei Aussagen.

Lemma 3.22. *Sei ein Wort w der Länge $2n$, $n > 1$, welches ein Teilwort (x, x) enthält und sei w^* das Wort, das wir erhalten, wenn wir das Teilwort (x, x) entfernen. Dann folgt $\gamma(w) \geq \gamma(w^*) + 2$.*

Beweis. Wir färben w^* optimal und übernehmen diese Färbung als Teilfärbung für
$w = (a_1, \ldots, a_{2n})$. Für eine komplette Färbung von w müssen nur noch die zwei Lettern a_i und a_{i+1} gefärbt werden, die für die Erstellung von w^* entfernt wurden. Gilt $i = 1$, befinden sich die entfernten Lettern also am Anfang des Wortes, so färben wir $(\overline{x}, \dot{x}, \overline{a}_3, \ldots)$. Ist $i = 2n - 1$, dass heißt die entfernten Lettern stehen am Ende des Wortes, so färben wir $(\ldots, \overline{a}_{2n-2}, \dot{x}, \overline{x})$ In beiden Fällen erhalten wir zwei zusätzliche Farbwechsel. Sei nun $1 < i < 2n - 1$. Dann gibt es die beiden Möglichkeiten, dass a_{i-1} und a_{i+2} gleich oder entgegengesetzt gefärbt sind. Mit $(\ldots, \dot{a}_{i-1}, \overline{x}, \dot{x}, \dot{a}_{i+2}, \ldots)$ und $(\ldots, \dot{a}_{i+1}, \overline{x}, \dot{x}, \overline{a}_{i+2}, \ldots)$ erhalten wir jeweils zwei zusätzliche Farbwechsel gegenüber der optimalen Färbung von w^*.

□

Lemma 3.23. *Sei ein Wort w der Länge $2n$, $n > 3$, welches die zwei Teilwörter (x, y, x) und (z, y, z) enthält. Sei w^* das Wort, das wir erhalten, wenn wir die beiden Teilwörter (x, y, x) und (z, y, z) entfernen. Dann folgt $\gamma(w) \geq \gamma(w^*) + 4$.*

Beweis. Wir färben w^* optimal und übernehmen wieder diese Färbung als Teilfärbung für $w = (a_1, \ldots, a_{2n})$. Nun müssen noch die 6 Lettern $a_{(i-1)} = x, a_i = y, a_{(i+1)} = x, a_{(j-1)} = z, a_j = y, a_{(j+1)} = z$ gefärbt werden. Gilt $i = 2$, so befinden sich die entfernten Lettern $a_{(i-1)}, a_i, a_{(i+1)}$ am Anfang des Wortes, und wir färben $(\overline{x}, \dot{y}, \dot{x}, \overline{a}_3, \ldots)$. Ist $i = 2n-1$, dass heißt die entfernten Lettern stehen am Ende des Wortes, so färben wir $(\ldots, \overline{a}_{2n-3}, \dot{x}, \overline{y}, \overline{x})$. In beiden Fällen erhalten wir zwei zusätzliche Farbwechsel durch die Färbung des Teilwortes (x, y, x). Sei nun $2 < i < 2n-1$. Dann gibt es die beiden Möglichkeiten, dass a_{i-2} und a_{i+2} gleich oder entgegengesetzt gefärbt sind. Mit $(\ldots, \dot{a}_{i-2}, \overline{x}, \dot{y}, \dot{x}, \dot{a}_{i+2}, \ldots)$ und $(\ldots, \dot{a}_{i-2}, \overline{x}, \dot{y}, \dot{x}, \overline{a}_{i+2}, \ldots)$ erhalten wir jeweils zwei zusätzliche Farbwechsel. Wir bemerken, dass die Farbe von y keinen Einfluss auf die Anzahl der Farbwechsel hat. Mit den gleichen Argumenten können wir auch (z, y, z) einfärben, wobei wir hier y entgegen dem y im Teilwort (x, y, x) färben. So erhalten wir weitere zwei zusätzliche Farbwechsel und die Behauptung ist damit bewiesen. □

Lemma 3.24. *Sei ein Wort w der Länge $2n$, $n > 3$, welches die zwei Teilwörter (x, y, x) und (z, v, z) enthält. Sei w^* das Wort, welches wir erhalten, wenn wir die beiden Teilwörter (x, y, x) und (z, v, z) entfernen und die Letter v außerhalb des Teilworts durch y ersetzen. Dann folgt $\gamma(w) \geq \gamma(w^*) + 4$.*

Beweis. Auch hier färben wir w^* optimal und übernehmen wieder diese Färbung als Teilfärbung für $w = (a_1, \ldots, a_{2n})$, wobei wir das v in w außerhalb z, v, z so färben wie das y in w^*, welches durch die Substitution von v entstanden ist. Nun müssen noch die Teilwörter (x, y, x) und (z, v, z) eingefärbt werden. Dabei gehen wir wie in Lemma 3.23 vor, unter der Beachtung, dass y und z innerhalb (x, y, x) und (z, v, z) entgegen den schon gefärbte Lettern y und v eingefärbt werden. Unabhängig von der Farbe der Lettern y und v innerhalb der Teilwörter (x, y, x) und (z, v, z) erhalten wir so jeweils zwei zusätzliche Farbwechsel pro eingefärbten Teilwort und die Behauptung ist damit bewiesen. □

Wir wollen Lemma 3.24 noch durch ein Beispiel veranschaulichen.
Sei $w = (a, b, c, d, c, a, d, f, g, f, b, g)$ gegeben. Mit (c, d, c) und (f, g, f) gibt es zwei Teilwörter des Typs (x, y, x). Um das Wort w^* zu erzeugen, entfernen wir (c, d, c) und (f, g, f) und ersetzen das g außerhalb (f, g, f) durch d. Damit erhalten wir $w^* = (a, b, a, d, b, d)$. Wir sehen, dass $(\dot{a}, \overline{b}, \overline{a}, \dot{d}, \dot{b}, \overline{d})$ eine optimale Färbung für w ist, also gilt $\gamma(w^*) = 3$. Für eine Teilfärbung von w übernehmen die Färbung von w^*, wobei wir das g außerhalb (f, g, f) wie das d in w^*, welches durch Substitution von g entstanden ist, einfärben. Damit erhalten wir $(\dot{a}, \overline{b}, c, d, c, \overline{a}, \dot{d}, f, g, f, \dot{b}, \overline{g})$. Durch Einfärben der zwei Teilwörter erhalten wir
$(\dot{a}, \overline{b}, \dot{c}, \overline{d}, \overline{c}, \overline{a}, \dot{d}, \overline{f}, \dot{g}, \dot{f}, \dot{b}, \overline{g})$, also 4 Farbwechsel mehr und somit folgt $\gamma(w) \geq 7$.

Nun können wir schließlich Satz 3.15 beweisen.

Beweis. (Satz 3.15) Für den Beweis wollen wir ausnutzen, dass man die für die Beweise der Lemmata 3.22, 3.23 und 3.24 benutzte Methode auch mehrfach hintereinander auf ein Wort anwenden kann. Wir können solange pro Schritt ein Teilwort (x, x) entfernen oder ein Paar von Teilwörtern $((x, y, x), (z, y, z))$ entfernen oder ein Paar von Teilwörtern $((x, y, x), (z, v, z))$ entfernen und das verbliebene v durch y ersetzen bis das verbliebene Restwort in W'' liegt oder eines der beiden Wörter (a, a) oder (a, b, a, c, b, c) ist. Wir färben das Restwort und fügen

nun in umgekehrter Reihenfolge die zuvor entfernten Teilwörter wieder ein und färben dabei entsprechend den Beweisen der Lemmata 3.22, 3.23 und 3.24 ein. Für unseren Beweis wollen wir eine Induktion durchführen über die Anzahl solcher Schritte, die notwendig sind, bis wir ein Wort in W' oder eines der Wörter (a,a) oder (a,b,a,c,b,c) erhalten.
Für Induktionsanfang wählen wir, dass das Wort w_0 in W'' liegt oder eines der Wörter (a,a) oder (a,b,a,c,b,c) ist. Ist $w_0 \in W''$, gilt die Behauptung, dies hatten wir mit Korollar 3.21 schon bewiesen. Ist $w_0 = (a,a)$, färben wir w_0 mit einen Farbwechsel. Nach der Behauptung muss $\gamma(w_0) \geq \frac{4}{3}(n-1)+1$, also $\gamma(w_0) \geq 1$ gelten, so dass hier die Behauptung erfüllt ist. Für den Fall $w_0 = (a,b,a,c,b,c)$ können wir $(\dot{a},\overline{b},\overline{a},\dot{c},\dot{b},\overline{c})$ färben, erreichen also 3 Farbwechsel. Die Behauptung verlangt $\gamma(w_0) \geq \frac{4}{3}n - 1$, also $\gamma(w_0) \geq 3$, und so erfüllen wir auch hier die Behauptung. Somit ist der Induktionsanfang bewiesen.
Wir wollen nun annehmen, dass die Behauptungen aus Satz 3.15 für Wörter erfüllt ist, bei denen wir in k Schritten durch Entfernen von Teilwörtern ein Restwort erhalten, welches in W' liegt oder eines der Wörter (a,a) oder (a,b,a,c,b,c) ist. Sei w_{k+1} ein Wort, bei dem wir nun in $k+1$ Schritten ein solches Restwort erhalten. Sei weiterhin w_k ein Wort, welches wir durch Entfernen eines Teilwortes (x,x), zweier Teilwörter $((x,y,x),(z,y,z))$ oder zweier Teilwörter $((x,y,x),(z,v,z))$ aus w_{k+1} erhalten. Wir wollen die Länge von w_{k+1} mit $2n$ annehmen und die 6 folgenden Fälle unterscheiden:

1. 3 ist Teiler von n. Wir müssen daher $\gamma(w_{k+1}) \geq \frac{4}{3}n - 1$ zeigen.
   a) w_k entsteht aus w_{k+1} durch Entfernen eines Teilwortes (x,x).
   Damit beträgt die Länge von w_k dann $2(n-1)$ und $((n-1)-2)$ ist durch 3 teilbar. Dann gilt Satz 3.15 entsprechend $\gamma(w_k) \geq \frac{4}{3}((n-1)-2)+2$ und mit Lemma 3.22 folgt $\gamma(w_{k+1}) \geq \frac{4}{3}((n-1)-2)+2+2 = \frac{4}{3}n$. Somit gilt $\gamma(w_{k+1}) \geq \frac{4}{3}n - 1$ und die Behauptung ist in diesem Fall bewiesen.
   b) w_k entsteht aus w_{k+1} durch Entfernen von Teilwörtern (x,y,x) und (x,y,x) oder (x,y,x) und (x,z,x).
   Somit beträgt die Länge von w_k hier $2(n-3)$ und $(n-3)$ ist durch 3 teilbar. Nach Satz 3.15 gilt entsprechend $\gamma(w_k) \geq \frac{4}{3}(n-3)-1$ und durch die Lemmata 3.23 und 3.24 folgt $\gamma(w_{k+1}) \geq \frac{4}{3}(n-3)-1+4 = \frac{4}{3}n - 1$. Damit ist die Behauptung ist bewiesen.

2. 3 ist Teiler von $(n-1)$. Somit müssen wir $\gamma(w_{k+1}) \geq \frac{4}{3}(n-1)+1$ zeigen.
   a) w_k entsteht aus w_{k+1} durch Entfernen eines Teilwortes (x,x).
   Damit beträgt die Länge von w_k dann $2(n-1)$ und da $(n-1)$ durch 3 teilbar ist, folgt mit Satz 3.15 entsprechend $\gamma(w_k) \geq \frac{4}{3}(n-1)-1$ und mit Lemma 3.22 folgt $\gamma(w_{k+1}) \geq \frac{4}{3}(n-1)+1$. Somit ist die Behauptung in diesem Fall bewiesen.
   b) w_k entsteht aus w_{k+1} durch Entfernen von Teilwörtern (x,y,x) und (x,y,x) oder (x,y,x) und (x,z,x).
   Somit beträgt die Länge von w_k hier $2(n-3)$ und $((n-3)-1)$ ist durch 3 teilbar. Nach Satz 3.15 gilt entsprechend $\gamma(w_k) \geq \frac{4}{3}((n-3)-1)+1$ und durch die Lemmata 3.23 und 3.24 folgt $\gamma(w_{k+1}) \geq \frac{4}{3}((n-3)-1)+1+4 = \frac{4}{3}(n-1)+1$. Damit ist die Behauptung ist bewiesen.

Tabelle 4: Färbung des Wortes $w = (a, b, c, d, c, b, d, e, f, a, g, h, g, e, h, f)$ mit Methoden nach dem Beweis des Satzes 3.15. Die Färbung besitzt 11 Farbwechsel, nach Satz 3.15 muss $\gamma(w) \geq \frac{4}{3}(n-2)+2 = 10$ gelten.

$(a, b, c, d, c, b, d, e, f, a, g, h, g, e, h, f)$	ungefärbtes Wort
$(a, b, b, d, e, f, a, e, d, f)$	Entfernen von (c, d, c) und (g, h, g), Tauschen h nach d
(a, d, e, f, a, e, d, f)	Entfernen von (b, b)
$(\overline{a^*}, a, d, e, f, a, e, d, f, z^*)$	Hinzufügen von a^* und z^*, färben von a^*
$(\overline{a^*}, \dot{a}, d, e, f, \overline{a}, e, d, f, z^*)$	Färben von a mit $a10$
$(\overline{a^*}, \dot{a}, \overline{d}, e, f, \overline{a}, e, \dot{d}, f, z^*)$	Färben von d mit $a10$
$(\overline{a^*}, \dot{a}, \overline{d}, \dot{e}, f, \overline{a}, \overline{e}, \dot{d}, f, z^*)$	Färben von e mit $b31$
$(\overline{a^*}, \dot{a}, \overline{d}, \dot{e}, \dot{f}, \overline{a}, \overline{e}, \dot{d}, \overline{f}, z^*)$	Färben von f mit $b31$
$(\dot{a}, \overline{d}, \dot{e}, \dot{f}, \overline{a}, \overline{e}, \dot{d}, \overline{f})$	Entfernen von a^* und z^*
$(\dot{a}, b, b, \overline{d}, \dot{e}, \dot{f}, \overline{a}, \overline{e}, \dot{d}, \overline{f})$	Hinzufügen von (b, b)
$(\dot{a}, \overline{b}, \dot{b}, \overline{d}, \dot{e}, \dot{f}, \overline{a}, \overline{e}, \dot{d}, \overline{f})$	Färben von (b, b)
$(\dot{a}, \overline{b}, c, d, c, \dot{b}, \overline{d}, \dot{e}, \dot{f}, \overline{a}, \overline{e}, g, h, g, \dot{h}, \overline{f})$	Hinzufügen von (c, d, c) und (g, h, g), Tauschen d nach h
$(\dot{a}, \overline{b}, \dot{c}, \dot{d}, \overline{c}, \dot{b}, \overline{d}, \dot{e}, \dot{f}, \overline{a}, \overline{e}, \dot{g}, \overline{h}, \overline{g}, \dot{h}, \overline{f})$	Färben von (c, d, c) und (g, h, g)

3. 3 ist Teiler von $(n-2)$. Wir müssen daher $\gamma(w_{k+1}) \geq \frac{4}{3}(n-2)+2$ zeigen.

   a) w_k entsteht aus w_{k+1} durch Entfernen eines Teilwortes (x, x).
   Damit beträgt die Länge von w_k dann $2(n-1)$ und da $((n-1)-1)$ durch 3 teilbar ist, folgt mit Satz 3.15 entsprechend $\gamma(w_k) \geq \frac{4}{3}((n-1)-1)+1$ und mit Lemma 3.22 folgt $\gamma(w_{k+1}) \geq \frac{4}{3}((n-1)-1)+1+2 = \frac{4}{3}(n-2)+3$. Somit gilt $\gamma(w_{k+1}) \geq \frac{4}{3}(n-2)+2$ und die Behauptung ist in diesem Fall bewiesen.

   b) w_k entsteht aus w_{k+1} durch Entfernen von Teilwörtern (x, y, x) und (x, y, x) oder (x, y, x) und (x, z, x).
   Somit beträgt die Länge von w_k hier $2(n-3)$ und $((n-3)-2)$ ist durch 3 teilbar. Nach Satz 3.15 gilt entsprechend $\gamma(w_k) \geq \frac{4}{3}((n-3)-2)+2$ und durch die Lemmata 3.23 und 3.24 folgt folgt $\gamma(w_{k+1}) \geq \frac{4}{3}((n-3)-2)+2+4 = \frac{4}{3}(n-2)+2$. Damit ist die Behauptung ist bewiesen.

□

In Tabelle 4 wird an einem Beispiel die im Beweis genutzte Methode noch einmal gezeigt. Für $n \geq 1$ folgt aus $\frac{4}{3}(n-1)+1 > \frac{4}{3}(n-2)+2 > \frac{4}{3}n-1$ und $(\frac{4}{3}n-1)/(2n-1) = \frac{2}{3}+\frac{1}{6n-3}$, dass sich aus der hier vorgestellte Methoden ein $\frac{2}{3}$-Algorithmus entwickeln lässt. Wir wollen noch grob die Komplexität eines solchen Algorithmus abschätzen. Dabei gehen wir zunächst davon aus, dass wir keine Teilwörter entfernen. Dann müssen wir n Buchstaben einfärben. In jedem Schritt zur Färbung wollen wir die Umgebung aller Buchstaben kennen, um den nächsten für die Färbung auszuwählen. Allerdings brauchen wir nicht alle Umgebungen neu zu

ermitteln, sondern nur von den maximal 4 Nachbarn des zuletzt gefärbten Buchstabens, so dass der Aufwand hierfür konstant ist. Im Beweis für Lemma 3.18 hatten wir benutzt, dass man zur Verhinderung einer Blockade des Algorithmus die letzte Färbung entfernt und einen anderen Buchstaben wählt. Hierfür war es ausreichend, unter den Nachbarn des zuletzt vor der Blockade gefärbten Buchstaben zu wählen, so dass auch dieser Schritt unabhängig von n ist. Somit liegt der Aufwand zum Färben eines Wortes ohne Entfernen von Teilwörtern bei $\mathcal{O}(n)$. Dies ändert sich auch nicht, wenn wir beim Färben Teilwörter entfernen, denn der Aufwand für das Färben eines Teilwortes ist konstant und die maximale Anzahl von Teilwörtern in einem Wort steigt linear mit n.

3.2.4. Das Binary Paintshop Maximization Problem und $\mathcal{APX}$-hardness

Leider können wir in dieser Arbeit keinen Beweis angeben, der zeigt, dass das Binary Paintshop Maximization Problem $\mathcal{APX}$-hard ist. Trotzdem wollen wir versuchen, dass Problem hinsichtlich seiner Schwere der Approximierbarkeit einzuordnen. Dazu wollen wir zunächst eine Reduktion auf MAX-CUT angeben und definieren davor das Problem.

Definition 3.25. *Gegeben sei ein kantengewichteter Graph. Der maximale Schnitt des Graphen ist eine Zerlegung seiner Knotenmenge V in zwei Teilmengen V_1 und V_2 so, dass das Gesamtgewicht der zwischen den beiden Teilmengen verlaufenden Kanten maximal wird. Das Entscheidungsproblem MAX-CUT fragt nun, ob es für einen gegeben Graphen G mit einer Kantenwichtung u und einen vorgegeben $k > 0$ einen Schnitt gibt, der mindestens so groß wie k ist. Das Optimierungsproblem MAX-CUT besteht darin, einen Schnitt mit einem möglichst großen Kantengewicht zu finden.*

MAX-CUT als Entscheidungsproblem gehörte auch schon zu den 21 $\mathcal{NP}$-vollständigen Problemen von Karp [8]. Als Optimierungsproblem ist es $\mathcal{APX}$-vollständig [11].

Wir wollen folgend eine Reduzierung von BPMP auf MAX-CUT angeben. Sei ein Wort w gegeben. Wir erzeugen einem w zugehörigen Graphen G_w und eine Kantenwichtung u_w folgender Weise:

1. Jede Letter a_i aus $w = (a_1, .., a_{2n})$ wird mit einem Knoten v_i des Graphen G_w identifiziert.
2. Zwischen zwei Knoten v_i und v_j befindet sich genau eine Kante mit der Wichtung 1, wenn die zugehörigen Lettern x_i und x_j Nachbarn sind und nicht zum selben Buchstaben gehören.
3. Zwischen zwei Knoten v_i und v_j befindet sich genau eine Kante mit der Wichtung 2, wenn die zugehörigen Lettern a_i und a_j zum selben Buchstaben gehören und keine Nachbarn sind.
4. Zwischen zwei Knoten v_i und v_j befindet sich genau eine Kante mit der Wichtung 3, wenn die zugehörigen Lettern a_i und a_j zum selben Buchstaben gehören und Nachbarn sind.

Lemma 3.26. *Sei (V_0/V_1) ein Schnitt des zu w gehörenden Graphen G_w mit einem Gesamtgewicht der geschnittenen Kanten k. Dann gibt es einen Schnitt (V_0^*/V_1^*) von G_w mit einem*

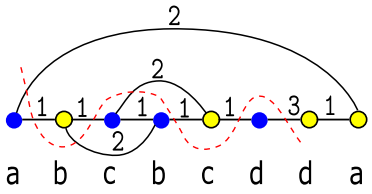

Abbildung 11: Wort $w = (a, b, c, b, c, d, d, a)$ und der korrespondierende kantengewichtete Graph G_w. Die gestrichelte Linie entspricht einem optimalen Schnitt mit Gewicht $k = 13$ und zugehöriger optimaler Färbung von w mit $\gamma(w) = k - 2n = 5$.

Gesamtgewicht der geschnittenen Kanten $k^* \geq k$ so, dass jeweils von den zu einem Buchstaben gehörenden Knoten v_i und v_j genau einer in V_1 liegt.

Beweis. Sei nun (V_0/V_1) ein Schnitt von G_w mit Gesamtgewicht k, wobei von m Buchstaben die zugehörigen Knoten nur in V_1 und von l Buchstaben die zugehörigen Knoten nur in V_2 liegen. Wir erzeugen einen Schnitt (V_0^*/V_1^*) aus (V_1/V_2) indem wir nacheinander jeweils einen Knoten in die andere Knotenmenge schieben, falls zwei Knoten v_i und v_j zu einem zugehörigen Buchstaben in der selben Teilmenge liegen. Durch das Schieben wird nun die Kante zwischen v_i und v_j mit Kantengewicht von mindestens 2 in den Schnitt aufgenommen, während höchstens zwei Kanten mit Kantengewicht 1 aus dem Schnitt entfernt werden. Für das Gesamtgewicht k^* des Schnittes (V_0^*/V_1^*) folgt $k^* \geq k$. □

Satz 3.27. *Genau wenn w eine Färbung mit $\gamma(w)$ Farbwechseln besitzt, gibt es einen Schnitt von G_w mit einem Gesamtgewicht $k = \gamma(w) + 2n$.*

Beweis. a) Sei w ein Wort der Länge $2n$, für das es eine Färbung mit $\gamma(w)$ Farbwechseln gibt. Wir erzeugen einen Schnitt (V_0/V_1) von G_w folgendermaßen: Ist die Letter zu einem Knoten mit 0 gefärbt, ordnen wir den Knoten V_0 zu, bei einer Färbung der Letter mit 1 wird der Knoten V_1 zugeordnet. So erhalten wir für je zwei benachbarte Knoten, deren Lettern unterschiedlich gefärbt sind, 1 Kantengewicht und für jeden Buchstaben 2 Kantengewichte, also insgesamt $k = \gamma + 2n$.
Sei nun (V_0/V_1) ein Schnitt von G_w mit Gesamtgewicht k. Wir erzeugen nun einen Schnitt (V_0^*/V_1^*) wie in Satz 3.26. Färbt man nun die zu den Knoten aus V_0^* zugehörigen Lettern mit 0 und die zu den Knoten aus V_1^* zugehörigen Lettern mit 1, so erhält man eine gültige Färbung von w mit $\gamma(w) = k^* - 2n \geq k - 2n$ Farbwechseln. □

Wir haben also unserer Problem auf MAX-CUT auf Graphen mit maximalem Knotengrad 3 reduziert. Da $\gamma(w) > 0.5n$ und das maximale Kantengewicht kleiner als $3n$ ist, erhalten wir somit eine L-Reduktion mit $\alpha = 6$ und $\beta = 1$. Tatsächlich ist auch MAX-CUT auf Graphen mit maximalen Knotengrad 3 noch $\mathcal{APX}$-hard [11]. Das bedeutet, falls BPMP $\mathcal{APX}$-hard wäre,

hätten wir gezeigt, das MAX-CUT auf Graphen mit maximalen Knotengrad 3 und mit einer sehr speziellen Kantenwichtung $\mathcal{APX}$-hard ist. Das Ergebnis aus [11] wäre nochmals verbessert.

Für MAX-CUT gibt es einen einfachen 0.5-Polynomialzeitalgorithmus. Goemans und Williamson [3] gaben schließlich ihren bekannten 0.878- Approximationsalgorithmus unter Benutzung Semidefiniter Programmierung an. Mit ähnlichen Argumenten wie oben beschrieben lässt sich BPMP auch auf MAX2-SAT reduzieren. MAX2-SAT ist ebenfalls $\mathcal{APX}$-vollständig und kann mit Hilfe Semidefiniter Programmierung und dem LLZ- Algorithmus beim Runden mit einer Güte von 0.940 approximiert werden[13]. Wir werden Approximationsalgorithmen, welche Semidefinite Programmierung benutzen, in den folgenden Kapiteln noch stärker beleuchten.

Hinsichtlich $\mathcal{APX}$-hardness wollen wir noch eine letzte Idee diskutieren. Wir können die Problembeschreibung von BPMP zu einem Problem auch so umformulieren, dass wir nicht die maximale Anzahl an Farbwechseln suchen, sondern die Anzahl der gleich gefärbten Nachbarn minimieren wollen. Das Optimierungsproblem bleibt an sich gleich. Optimale Lösungen des ursprünglichen Problems als auch des umformulierten Problems sind jeweils optimale Lösungen beider Probleme, nur die Art und Weise, wie wir Kosten einer Lösung ermitteln ändert sich. Sei BPMP* das umformulierte Problem.

Mit der in Abschnitt 3.1 formulierten Reduktion vom Binary Paintshop Minimation Problem auf BPMP erhalten wir so eine L-Reduktion mit $\alpha = 1$ und $\beta = 1$ auf BPMP*. In [4] wurde gezeigt, dass das Binary Paintshop Minimation Problem $\mathcal{APX}$-hard ist. Damit ist auch BPMP* $\mathcal{APX}$-hard.

4. Approximationen für das Binary-Paintshop-Maximization-Problem

In den vorherigen Kapiteln wurde mit dem Greedy- Algorithmus eine einfache Heuristik für BPMP vorgestellt. Wir wollen nun weitere Approximationsmöglichkeiten untersuchen. Schwerpunkt bildet die Lösung mit Hilfe der Semidefiniten Programmierung. Wir wollen vorher noch kurz zwei andere Ansätze vorstellen.

4.1. Binary Integer Programming (0-1)

Für ein gegebenes Wort der Länge $2n$ wollen wir die Lücken zwischen den Lettern durch die Indizes 1 bis $2n-1$ identifizieren und die Buchstaben von 1 bis n durchnummerieren. Sei nun folgendes Programm formuliert:

$$\min \quad \sum_{i=1}^{2n+1} y_i$$

unter

$$
\begin{aligned}
x_i + x_i^* + y_i &= 1, && \text{für alle Lücken } i,\ 1 \le i \le 2n-1.\\
x_j - x_j^* + x_{j+1} - x_{j+1}^* + \ldots + x_k - x_k^* + 2z_m &= 1, && \text{für alle } n \text{ Buchstaben, wobei } j\\
& && \text{die Lücke nach der ersten Letter}\\
& && \text{und } k \text{ die Lücke vor der zweiten}\\
& && \text{Letter des m-ten Buchstaben ist.}\\
x_i, x_i^*, y_i, z_m &= 0 \text{ od. } 1, && 1 \le i \le 2n-1, 1 \le m \le n.
\end{aligned}
$$

Dies ist ein Binary Integer Program, dass heißt alle Variablen können nur den Wert 0 oder 1 annehmen. Da sowohl die Zielfunktion als auch die Nebenbedingungen linear sind, ist es, noch genauer formuliert, ein Lineares Binary Integer Program. Für jedes Wort w lässt sich so ein Programm aufstellen. Aus jeder Lösung des Programms lässt sich eine zulässige Färbung für w ableiten: Die erste Letter im Wort wird mit 0 gefärbt. Ist $x_i = 1$ oder $x_i^* = 1$ wird die Farbe zwischen den beiden Lettern vor und hinter der Lücke i gewechselt, bei $x_i = x_i^* = 0$ findet kein Wechsel statt. Aufgrund der ersten Gleichung können x_i und x_i^* nicht gleichzeitig 1 sein. Durch die zweite Gleichung ist dann sichergestellt, dass die Anzahl der Farbwechsel zwischen den beiden Lettern eines Buchstabens ungerade ist, die beiden Lettern also unterschiedlich gefärbt werden. Die z_m dienen dazu, die zweite Gleichung erfüllbar zu machen, falls die Gleichung mit $x_j^* = 1$ beginnt. Die Zielfunktion soll dafür sorgen, dass möglichst viele y_i zu null werden und so durch $x_i = 1$ bzw. $x_i^* = 1$ möglichst viele Farbwechsel entstehen. Binary Integer Programming ist $\mathcal{NP}$- vollständig und das Entscheidungsproblem gehörte schon zu den 21 $\mathcal{NP}$-vollständigen Problemen von Karp [8]. Lösungsstrategien für Binary Integer Programs können beispielsweise Branch-and-Bound-Algorithmen beinhalten.

Die eben beschrieben Ansätze zur Lösung von BPMP durch Reduktion auf andere Probleme wollen wir hier nicht weiter verfolgen.

4.2. Formulierung von BPMP als Quadratisches Programm

Wir können BPMP für ein Wort $w = (a_1, .., a_{2n})$ auch wie folgend formulieren:

$$
\max \sum_{\substack{1 \le i \le 2n-1 \\ x_i \ne x_{i+1}}} 1
$$

$$
\begin{aligned}
&\text{s.t.} && && \\
&\quad x_i &&\ne x_j, && \text{falls } a_i = a_j \\
&\quad x_i &&= -1 \text{ oder } 1, && 1 \le i \le 2n-1.
\end{aligned}
\tag{6}
$$

Dabei identifizieren wir $x = (x_i, ..., x_{2n-1})$ als Färbung von w mit den Farben -1 und 1.

Definition 4.1. *(Laplace-Matrix L) Für ein Wort w der Länge $2n$ ist die Laplace-Matrix $L \in M^{2n \times 2n}$ folgend definiert:*

$$L := (l_{ij}),\ l_{ij} = \begin{cases} 1 & \{i \in \{1, 2n\} \mid i = j\} \\ 2 & \{2 \leq i \leq 2n-1 \mid i = j\} \\ -1 & \{1 \leq i \leq 2n-1 \mid j = i+1\} \text{ oder } \{2 \leq i \leq 2n \mid j = i-1\} \\ 0 & \text{sonst.} \end{cases}$$

Die Laplace- Matrix für ein Wort der Länge 6 hat also die Form

$$\text{L}= \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}.$$

Der Begriff Laplace-Matrix ist hier der Graphentheorie entlehnt. Dort entsprechen der Eintrag an der Stelle (i, i) der Anzahl der Nachbarknoten des Knoten i und der Eintrag $-a$ an der Stelle (i, j) einer Kante mit Kantengewicht a zwischen den Knoten i und j. Eine Laplace-Matrix für ein Wort der Länge $2n$ entspricht also der Laplace-Matrix eines Graphen der Form einer Kette mit $2n$ Knoten und jeweils Kantengewicht 1. Laplace-Matrizen sowohl für Wörter als auch für Graphen sind schwach diagonaldominant, dass heißt die Beträge ihrer Diagonalelemente l_{ii} sind jeweils größer oder gleich der Summe der Beträge der restlichen jeweiligen Zeileneinträge l_{ij}. Da außerdem alle Diagonalelemente nichtnegativ sind, sind Laplace-Matrizen positiv semidefinit [20].

Satz 4.2. *Sei $x = (x_i, ..x_{2n})^T$ ein Vektor mit $x_i \in \{-1, 1\}$ und $L = (l_{ij})$ die Laplace-Matrix zu einem Wort der Länge $2n$. Dann gilt*

$$\sum_{\substack{1 \leq i \leq 2n-1 \\ x_i \neq x_{i+1}}} 1 = \frac{1}{4} x^T L x.$$

Beweis. Es gilt $\sum_{i=1}^{2n} x_i l_{ii} x_i = 2n - 2$. Für jedes i mit $x_i = x_{i+1}$ gilt $x_i l_{i,i+1} x_{i+1} = x_{i+1} l_{i+1,i} x_i = -1$. Ist $x_i \neq x_{i+1}$, folgt $x_i l_{i,i+1} x_{i+1} = x_{i+1} l_{i+1,i} x_i = 1$. Da alle anderen Einträge $l_{i,j}$ der Matrix null sind, folgt für einen Vektor x, bei dem die Anzahl der i mit $x_i \neq x_{i+1}$ genau b ist:

$$\frac{1}{4} x^T L x = \frac{1}{4}(2n - 2 + 2b - 2(n - 1 - b)) = b. \tag{7}$$

□

Setzt man $X = xx^T$, können wir mit $x^T L x = \text{Spur}(Lxx^T)$ das zu Beginn des Abschnittes formulierte Programm für BPMP eines Wortes $w = (a_1, ..., a_{2n})$ in den Raum $\mathcal{S}^n$ der symmetrischen $2n \times 2n$-Matrizen überführen:

$$
\begin{array}{lll}
BPMP_2 & \max & \frac{1}{4}L \bullet X \\
& \text{unter} & X = xx^T, x \in \mathbb{R}^{2n} \\
& & X_{ii} = 1 \\
& & X_{ij} = -1, \text{ falls } a_i = a_j \text{ und } i \neq j.
\end{array}
$$

Dabei haben wir die Schreibweise $A \bullet B = \text{Spur}(A^T B)$ benutzt. Die einzelnen Komponenten des Vektors $x = (x_1, \ldots, x_{2n})$ können auch hier weiterhin nur die Werte -1 oder 1 annehmen. Dies ist ein ganzzahliges quadratisches Optimierungsproblem (engl. integer quadratic program, kurz QP). Auch dieses Problem ist $\mathcal{NP}$-vollständig. Allerdings soll diese Formulierung Ausgangspunkt für eine Semidefinite Relaxation sein, welche wir im nächsten Abschnitt genauer untersuchen wollen.

4.3. Grundlagen der Semidefiniten Programmierung

Semidefinite Programmierung ist ein relativ neues Gebiet der Optimierung. Seit ungefähr 1990 gibt es jedoch rasch anwachsende Aktivitäten auf dem Gebiet und die SDP ist in den Fokus von Experten verschiedener Bereiche wie Konvexe Programmierung, Kontrolltheorie, Numerische Programmierung und Kombinatorische Optimierung gerückt, so dass heutzutage die SDP einer der Bereiche mit hoher Forschungstätigkeit auf dem Gebiet der Optimierung ist[14]. Diese Entwicklung wurde angestoßen durch Entdeckungen von Anwendungsmöglichkeiten innerhalb der Kombinatorischen Optimierung und Kontrolltheorie, effizienten Innere-Punkte-Verfahren und der Eleganz der der SDP zugrundeliegenden Theorie. Einer der Meilensteine in Bezug auf die Anwendung der SDP für die Entwicklung von Approximationsalgorithmen für $\mathcal{NP}$-vollständige Maximierungsprobleme stellt die Arbeit von Goemans und Williamson aus dem Jahre 1995 dar [3]. Wegweisend wurde dort ein Näherungsverfahren für MAX-CUT mit Hilfe einer Semidefiniten Relaxierung angegeben, welches 0.879 des Optimalwertes garantiert. Bis dahin waren nur Verfahren bekannt, die mit einer Leistungsgarantie von $\frac{1}{2} + \frac{1}{2n}$ nur wenig besser waren als einfache Verfahren, die bereits $\frac{1}{2}$ garantieren [15].

Bevor wir versuchen unserer Problem mit Hilfe der Semidefiniten Programmierung (SDP) zu bearbeiten, wollen wir zunächst ein paar Grundlagen vorstellen. Dabei können wir zuerst folgende grobe Einordnung der SDP geben: Die SDP lässt sich dem Gebiet der mathematischen Optimierung zuordnen. Hierbei geht es darum, die optimale Lösung eines Problems unter Einhaltung eventuell gegebener Randbedingungen zu finden. Dies geschieht dadurch, dass man die zu optimierenden Parameter durch eine Zielfunktion beschreibt und die Extrema dieser Funktion auf einer von den Restriktionen definierten zulässigen Menge sucht. Innerhalb der mathematischen Optimierung gehört die SDP zu den skalaren Optimierungsproblemen, dass heißt die Zielfunktion ist reellwertig, im Gegensatz zur Vektoroptimierung mit vektorwertigen Zielfunktionen. Wiederum innerhalb der skalaren Optimierung ist die SDP ein sogenanntes Konvexes Programm, dass bedeutet, die zulässige Menge ist konvex und die Zielfunktion ist konvex (konkav). Dadurch ist hier ein lokales Optimum auch stets ein globales Optimum [18]. In einer weiteren Unterteilung wird die SDP den konischen Programmen zugeordnet. In konischen Programmen werden zur Formulierung der zulässigen Punkte Kegel verwendet, also Teilmengen eines Vektorraums, die abgeschlossen sind bezüglich der Multiplikation mit positiven Skalaren

[18]. Folgende Definition ist namensgebend für die SDP:

Definition 4.3. *(positiv semidefinite Matrizen) Eine reelle Matrix A wird positiv semidefinit genannt, wenn A symmetrisch ist und für alle $x \in \mathbb{R}^n$ gilt: $x^T Ax \geq 0$. Ist A positiv semidefinit schreiben wir auch $A \succcurlyeq 0$.*

Auf den Vektorraum $\mathcal{S}^n$ der symmetrischen $n \times n$-Matrizen bilden die positiv semidefiniten Matrizen einen Kegel $\mathcal{S}^n_+ := \{A \in \mathcal{S}^n \mid x^T Ax \geq 0 \text{ für alle } x \in \mathbb{R}^n\}$.
Des Weiteren wird durch $A \succcurlyeq B \iff A - B$ ist positiv semidefinit die sogenannte Loewner-Halbordnung auf $\mathcal{S}^n$ definiert.

Damit wollen wir nun die Standardform der semidefiniten Programmierung angeben [12]:

$$\begin{array}{ll} \min & C \bullet X \\ \text{unter} & A_i \bullet X = b_i, \quad i = 1, ..., m \\ & X \succcurlyeq 0, \end{array}$$

wobei C, A_i und X in $\mathcal{S}^n$ liegen.

Nun möchten wir mit $\boldsymbol{vec}(A)$ den Vektor beschreiben, den wir erhalten, indem wir die Spalten einer Matrix A untereinander schreiben, das heißt

$$\boldsymbol{vec}(A)^T = (a_{11}, \ldots, a_{m1}, \ldots, a_{1n}, \ldots, a_{mn}). \tag{8}$$

Falls mit $v \in \mathbb{R}^{mn}$ ein Vektor gegeben ist, so wollen wir mit $\boldsymbol{Mat}_{m,n}(v)$ die Matrix

$$\boldsymbol{Mat}_{m,n}(v) = \begin{pmatrix} v_1 & \cdots & v_{(m-1)n+1} \\ \vdots & \ddots & \vdots \\ v_n & \cdots & v_{mn} \end{pmatrix} \tag{9}$$

bezeichnen, und wenn aus dem Kontext die Dimension von $\boldsymbol{Mat}_{m,n}(v)$ erkennbar ist, so wollen wir nur kurz $\boldsymbol{Mat}(v)$ schreiben.
Wenn wir außerdem mit $\mathcal{A}$ die Matrix definieren, die aus den Zeilen $\boldsymbol{vec}(A_i)^T$ besteht, so lässt sich die oben formulierte Standardform der SDP auch angeben mit

$$\begin{array}{ll} \min & C \bullet X \\ \text{unter} & \mathcal{A}\boldsymbol{vec}(X) = b \\ & X \succcurlyeq 0. \end{array}$$

Dann erhalten wir als zugehöriges duales Optimierungsproblem [12]

$$\begin{array}{ll} \max & u^T b \\ \text{unter} & \boldsymbol{Mat}(u^T \mathcal{A}) + S = C \\ & S \succcurlyeq 0. \end{array}$$

Das duale Problem des dualen Problems eines SDP ist stets wieder das primale Problem. Außerdem gilt stets die schwache Dualität, dass heißt der Zielfunktionswert des dualen Problems ist

stets kleiner oder gleich dem Zielfunktionswert des primalen Problems. Die starke Dualität, also dass der Optimalwert des primalen Problems und der des dualen Problems übereinstimmen und die Dualitätslücke null ist, gilt nicht im Allgemeinen. Eine Voraussetzung, die garantiert, dass die starke Dualität gilt, ist die sogenannte Slater-Bedingung. Die Slater-Bedingung verlangt, dass es einen zulässigen Punkt gibt, der alle Ungleichungen in den Nebenbedingen strikt erfüllt [17].

Es gibt verschiedene Algorithmen, die SDP lösen. Obwohl Semidefinite Programme wesentlich allgemeiner formuliert sind als Lineare Programme, sind sie nicht wesentlich schwieriger zu lösen [19]. Die größte Relevanz besitzen heutzutage Innere-Punkte-Verfahren aufgrund ihrer Robustheit und Effizienz in Theorie und Praxis. Das bedeutet, sie können in polynomieller Zeit das Programm zu jeder vorgegebenen Genauigkeit lösen.

4.4. Semidefinite Relaxierung

Wir wollen nun aus unserem quadratischen ± 1 Programm ein semidefinites Programm ableiten. Als grundsätzliche Idee wählen wir einen ähnlichen Zugang, wie ihn auch schon Goemans und Williamson für ihre Arbeit zu MAX-CUT vorgestellt hatten [3]. Zur Erinnerung geben wir nochmal das Problem an, das wir in ein SDP überführen wollen:

Gegeben sei ein Wort $w = (a_1, \ldots, a_{2n})$, bei dem es für jedes $i \in \{1, \ldots, 2n\}$ ein $j \in \{1, \ldots, 2n\}$, $j \neq i$ gibt, so dass $a_i = a_j$ gilt. Sei L die Laplace-Matrix für Wörter der Länge $2n$.

$$
\begin{array}{lll}
(BPMP_2) & \max & \frac{1}{4} L \bullet X \\
& \text{unter} & X = xx^T, x \in \mathbb{R}^{2n} \\
& & X_{ii} = 1 \\
& & X_{ij} = -1, \text{ falls } a_i = a_j \text{ und } i \neq j.
\end{array}
$$

Die Überführung diese Problems in ein SDP geschieht durch eine sogenannte Relaxation, dass heißt, wir werden Bedingungen an die Lösung des Problems lockern. Dabei sollen für zulässige Punkte des ursprünglichen Problems die jeweils entsprechenden Punkte im relaxierten Problem weiterhin zulässig sein. Außerdem soll der Optimalwert der Relaxation eine obere Grenze des eigentlichen Problems sein. Durch die Relaxation nehmen wir allerdings in Kauf, dass unsere Lösung keinem zulässigen Punkt aus dem Original entspricht. Wir werden daher das Verfahren von Goemans und Williamson benutzen, um aus der Optimallösung des SDP eine zulässige Lösung für BPMP zu ermitteln, die eine gute Näherung zur Optimallösung von BPMP darstellt.

Nun hatten wir bisher die x_i als Skalare aus der Menge $\{-1, 1\}$ betrachtet. In einem ersten Schritt wollen wir die x_i als 1-dimensionale Vektoren der Länge 1 annehmen. Die eigentliche Relaxation geschieht nun dadurch, dass wir nun erlauben, dass die 1-dimensionalen x_i als Vektoren v_i aus $\mathbb{R}^{2n}$ der Länge 1 betrachtet werden können. Somit liegen die v_i in der Einheitssphäre S^{2n}. Weiterhin können aber die v_i so gewählt werden, dass sie alle auf einer Gerade liegen, dass heißt, dass sie sich in einem 1-dimensionalen Vektorraum befinden. In diesem Fall sind jeweils zwei v_i gleich oder sie sind entgegengerichtet. Damit unser neues Programm tatsächlich eine Relaxation ist, müssen wir sicherstellen, dass sich der Wert unserer neuen Zielfunktion wieder auf den Wert von $\frac{1}{2} \sum_{i=1}^{2n-1} (1 - x_i \cdot x_{i+1})$ reduziert, falls die Lösung unseres neuen Programms in einen

1-dimensionalen Vektorraum liegt. Wir wollen daher das Produkt $x_i x_j$ durch das Skalarprodukt $v_i \cdot v_j$ ersetzen. Dadurch erfüllen wir die Forderung, denn $v_i \cdot v_j = 1$ falls $v_i = v_j$ und $v_i \cdot v_j = -1$ für $v_i = -v_j$. Wir wollen, ähnlich wie bei dem Verfahren von Goemans und Williamson, später die v_i einer Lösung durch eine Hyperebene durch den Ursprung in zwei Mengen trennen, die jeweils einer Farbe entsprechen. Da wir a_i und a_j unterschiedlich einfärben müssen, falls a_i=a_j, ist es notwendig, dass wir sicherstellen, dass die entsprechenden Vektoren v_i und v_j auf jeden Fall getrennt werden. Daher verlangen wir, dass für eine Lösung gelten muss, dass $v_i = -v_j$ falls $a_i = a_j$ gilt. Dies führt nun zu folgenden Programm:

$$\begin{array}{lll} (RELAX) & \max & \frac{1}{2}\sum_{i=1}^{2n-1}(1 - v_i \cdot v_{i+1}) \\ & \text{unter} & v_i \in S^{2n} \\ & & v_i = -v_j, \text{ falls } a_i = a_j \quad \forall i,j \in \{1,\ldots,2n\}. \end{array}$$

Wie eben gezeigt, ist $RELAX$ eine Reduzierung von $BPMP$ und $ZF_{RELAX} \geq ZF_{BPMP}$. Für dieses Programm können wir uns folgendes „mechanisches“ Modell vorstellen: Jede Letter a_i entspricht einem „Pfeil“ in $\mathbb{R}^{2n}$, wobei zwei Pfeile genau entgegengerichtet sind, wenn die beiden Lettern zum gleichen Buchstaben gehören. Für zwei Pfeile, dessen Lettern Nachbarn in w sind, befindet sich eine Feder, die versucht, die beiden Pfeile auseinander zudrücken, also den Winkel zwischen den beiden Pfeilen möglichst groß zu machen. Einen Winkel von 180° zwischen allen Pfeilen mit Feder zu realisieren wird im Allgemeinen nicht gelingen, dafür müssten alle Pfeile auf einer Geraden liegen und zwei Pfeile genau entgegenrichtet sein, falls die Lettern Nachbarn sind. Dies ist zum Beispiel nicht möglich, wenn der linke und rechte Nachbar einer Letter zum selben Buchstaben gehören.
Wir werden nun zeigen, dass $RELAX$ ein semidefinites Programm ist. Dazu wollen wir die Matrix Y, $(y_{ij}) = v_i \cdot v_j$ einführen. Durch diese Definition ist Y positiv semidefinit. Denn folgendes gilt:
Sind $z_i, \ldots, z_n$ Vektoren in $\mathbb{R}^n$ und ist Z die Matrix, die $z_i, \ldots, z_n$ als Spalten enthält, so ist die Matrix $G := Z^T Z$ positiv semidefinit. Die Matrix G wird auch Gram-Matrix von $\{z_1, \ldots, z_n\}$ genannt. Umgekehrt ist auch jede symmetrische Matrix G darstellbar als $G = B^T B$, wobei $B \in \mathbb{R}^{m \times n}$, $m \leq n$. Für ein gegebenes G kann B beispielsweise durch eine unvollständige Cholesky-Zerlegung in $\mathcal{O}(n^3)$ [16] oder durch Hauptachsentransformation ermittelt werden. Gilt G_{ii}=1, dann liegen die Spalten $(z_1, \ldots, z_n)$ von B in der Einheitssphäre S^m. Zusammenfassend kann man also sagen, dass wir aus einer Lösung eines semidefiniten Programms, welche aus einer semidefiniten Matrix besteht, in polynomieller Zeit unsere gesuchten Einheitsvektoren ermitteln können.

Wir beweisen nun noch folgenden Zusammenhang der Zielfunktion:

Satz 4.4. *Seien $v_1, \ldots, v_{2n}$ Vektoren in S_{2n} und $L = (l_{ij})$ die Laplace-Matrix zu einem Wort der Länge $2n$. Dann gilt*

$$\frac{1}{2}\sum_{i=1}^{2n-1}(1 - v_i \cdot v_{i+1}) = \frac{1}{4} L \bullet Y$$

Beweis. Unter Beachtung von $L_{11} = L_{2n,2n} = 1$, $L_{ii} = 2$ für $2 \leq i \leq 2n-1$ und $L_{i,i+1} =$

$L_{i+1,i} = -1$ und $L_{ij} = 0$ sonst, folgt:

$$\frac{1}{4}L \bullet Y = \frac{1}{4}\sum_{i=1}^{2n}\sum_{j=1}^{2n} L_{ij}Y_{ij} = \frac{1}{4}(\sum_{i=1}^{2n} L_{ii}Y_{ii} + 2\sum_{i=1}^{2n-1} L_{i,i+1}Y_{i,i+1}) \tag{10}$$

$$= \frac{1}{4}(2(n-2) + 2 + 2\sum_{i=1}^{2n-1}(-1)Y_{i,i+1}) \tag{11}$$

$$= \frac{1}{4}(2\sum_{i=1}^{2n-1} 1 + 2\sum_{i=1}^{2n-1}(-1)v_i \cdot v_{i+1}) \tag{12}$$

$$= \frac{1}{2}\sum_{i=1}^{2n-1}(1 - v_i \cdot v_{i+1}). \tag{13}$$

□

Damit können wir folgendes semidefinite Programm aufstellen:

$$\begin{array}{lll} SDP & \max & \frac{1}{4}L \bullet Y \\ & & Y_{ii} = 1 \\ & & Y_{ij} = -1, \text{ falls } a_i = a_j \quad \forall i,j \in \{1,\ldots,2n\} \\ & & Y \succcurlyeq 0. \end{array}$$

Und wie eben gezeigt, ist dieses Programm gleichwertig zu unserem ursprünglich relaxierten Programm $RELAX$. Wir wollen der Vollständigkeit halber auch noch die Normalform angeben:

$$\begin{array}{lll} SDPn & \min & C \bullet Y \\ & & A_i \bullet Y = 1 \qquad \forall i \in \{1,\ldots,2n\} \\ & & B_{ij} \bullet Y = -1, \quad \forall i,j \text{ mit } a_i = a_j \text{ und } i < j \\ & & Y \succcurlyeq 0. \end{array}$$

Dabei ist $C = -\frac{1}{4}L$. Für die $2n$ Matrizen $A_i = (a_{rs})$ gilt, $a_{ii} = 1$, alle anderen Einträge sind null. Die n Matrizen $B_{ij} = (b_{rs})$ sind definiert durch $b_{ij} = b_{ji} = \frac{1}{2}$, falls $a_i = a_j$ und $i < j$ für das gegebene Wort $w = (a_1, \ldots, a_{2n})$, alle anderen Einträge sind null.

4.5. Ermittlung einer zulässigen Lösung für BPMP

Wir wollen hier schon einmal die Idee skizzieren, wie wir eine Lösung für $BPMP$ mit Hilfe der Semidefiniten Programmierung erhalten, die detaillierte Herleitung erfolgt in den nächsten Abschnitten:

1. Löse $RELAX$, die Lösung besteht aus $2n$ Vektoren v_i in S^m, $m \leq 2n$.
2. Ermittle einen Vektor r, der auf S^m liegt, zufällig.
3. Überprüfe, ob $r \cdot v_i \neq 0$ für alle $i \in \{1,\ldots,2n\}$. Wenn nein, gehe zurück zu 2. .

4. Setze $S := \{i \mid v_i \cdot r \geq 0\}$ und $T := \{i \mid v_i \cdot r < 0\}$.

Die Vektoren der optimalen Lösung werden also durch eine zufälligen Hyperebene in zwei Teilmengen S und T geteilt und indem wir die Lettern, die wir S zuordnen, mit der ersten Farbe färben und die Lettern, die wir T zuordnen, mit der zweiten Farbe färben, erhalten wir eine gültige Färbung für w, also eine Lösung für unserer ursprüngliches Problem. Wir wollen nun noch die Güte der so ermittelten Lösung beweisen.

Sei nun $E[W]$ der Erwartungswert des Verfahrens, welches wir gerade vorgestellt haben. Unsere Abschätzung von $E[W]$ hängt nun davon ab, mit welcher Wahrscheinlichkeit zwei Vektoren v_i und v_{i+1} durch die zufällig erzeugte Hyperebene getrennt werden. Somit gilt [15]:

$$E[W] = \sum_{i=1}^{2n-1} \Pr[i \in S, i+1 \in T \vee i+1 \in S, i \in T], \tag{14}$$

wobei wir mit Pr (probability) die Wahrscheinlichkeit bezeichnet haben. Da v_i und $v_i + 1$ genau dann getrennt werden, wenn das Skalarprodukt mit r unterschiedliche Vorzeichen liefert, ist also die Wahrscheinlichkeit $\Pr[\mathrm{sgn}(v_i^T r) \neq \mathrm{sgn}(v_{i+1}^T r)]$ zu bestimmen [15]. Wiederum hängt diese Wahrscheinlichkeit linear vom Winkel zwischen v_i und v_{i+1} ab [3]:

Satz 4.5. *Für alle $v_i, v_{i+1} \in \mathbb{R}^{2n}$ gilt*

$$\Pr[\mathrm{sgn}(v_i^T r) \neq \mathrm{sgn}(v_{i+1}^T r)] = \frac{1}{\pi} \arccos(v_i^T v_{i+1}). \tag{15}$$

Beweis. Aus Symmetriegründen gilt $\Pr[\mathrm{sgn}(v_i^T r) \neq \mathrm{sgn}(v_{i+1}^T r)] = 2\Pr[v_i^T r \geq 0, v_{i+1}^T r < 0]$. Die Menge $\{v_i^T r \geq 0, v_{i+1}^T r < 0\}$ entspricht dem Schnitt von zwei Halbräumen, die mit dem Winkel $\theta = \arccos(v_i^T v_{i+1})$ gegeneinander geneigt sind. Der Schnitt dieser Menge mit einer Kugeloberfläche ergibt ein sphärisches Zweieck mit dem Innenwinkel θ. Die Fläche des Zweiecks entspricht dem $\theta/2\pi$-fachen Wert der vollen Kugeloberfläche, das bedeutet $\Pr[v_i^T r \geq 0, v_{i+1}^T r < 0] = \theta/2\pi$ und die Behauptung ist bewiesen [15].

□

Die Linearität des Erwartungswertes liefert [3]

Satz 4.6.

$$E[W] = \frac{1}{\pi} \sum_{i=1}^{2n-1} \arccos(v_i^T v_{i+1}). \tag{16}$$

Um den Erwartungswert abzuschätzen, wollen wir nun einen Wert für das Verhältnis $\frac{1}{\pi}\arccos(v_i^T v_{i+1})$ und $\frac{1}{2}(1 - v_i^T v_{i+1})$ angeben:

Satz 4.7. *Für alle $v_i \in \mathbb{R}^{2n}$ mit $-1 \leq v_i^T v_{i+1} \leq 1, (1 \leq i \leq 2n-1)$ gilt*

$$\frac{1}{\pi} \arccos(v_i^T v_{i+1}) \geq \alpha \frac{1}{2}(1 - v_i^T v_{i+1}) \tag{17}$$

mit

$$\alpha := \min_{0 \leq \theta \leq \pi} \frac{2}{\pi} \frac{\theta}{1 - \cos\theta} \approx 0,87856. \tag{18}$$

Beweis. Wir setzen $\cos\theta = v_i^T v_{i+1}$ und definieren

$$\beta(\theta) := \frac{\frac{1}{\pi}\arccos(\cos\theta)}{\frac{1}{2}(1-\cos\theta)} = \frac{2}{\pi}\frac{\theta}{(1-\cos\theta)}. \tag{19}$$

Bildet man die Ableitung von β und setzt $\beta'(\theta) = 0$, ergibt sich $1 = \cos\theta + \theta\sin\theta$. Als nichttriviale Lösung erhält man $\theta_0 = 2,331122\ldots$ und $\beta(\theta_0)$ nimmt den Wert $0,87856$ an. Wir wollen noch zeigen, dass $0,87856$ eine untere Schranke von α ist. Dazu stellen wir als erstes fest, dass

$$\frac{2}{\pi}\frac{\theta}{1-\cos\theta} \geq 1 \tag{20}$$

für $0 < \theta \leq \pi/2$. Da $f(\theta) = 1-\cos\theta$ im Intervall $\pi/2 \leq \theta \leq \pi$ konkav ist, folgt für jedes θ^* $f(\theta) \leq f(\theta^*)+(\theta-\theta^*)f'(\theta^*)$ oder entsprechend eingesetzt $1-\cos\theta \leq 1-\cos\theta^*+(\theta-\theta^*)\sin\theta^* = \theta\sin\theta^* + (1-\cos\theta^*-\theta^*\sin\theta^*)$. Wir wählen nun $\theta^* = 2,331122$, womit $1-\cos\theta^*-\theta^*\sin\theta^* < 0$ gilt. Damit können wir den vorherigen Ausdruck abschätzen zu $1-\cos\theta < \theta\sin\theta^*$ und erhalten schließlich [3]:

$$\alpha > \frac{2}{\pi\sin\theta^*} > 0.87856. \tag{21}$$

□

Somit können wir nun zusammenfassen:

Satz 4.8. *Sei $E[W]$ der Erwartungswert unseres zu Beginn dieses Abschnittes formulierten Verfahrens, Z^*_{BPMP} der Optimalwert des Binary-Paintshop-Maximization-Problems, sei Z^*_{SDP} der Optimalwert der semidefiniten Relaxierungen RELAX, SDP oder SDPn. Dann gilt [15]:*

$$E[W] \geq \alpha\frac{1}{2}\sum_{i=1}^{2n-1}(1-v_i^T v_{i+1}) = \alpha Z^*_{SDP}. \tag{22}$$

$$Z^*_{SDP} \geq Z^*_{BPMP} \geq E[W] \geq \alpha Z^*_{SDP} \geq \alpha Z^*_{BPMP}. \tag{23}$$

Wie beim Näherungsverfahren von Goemans und Williamson [3] für das MAX-CUT-Problem erreicht unser Näherungsverfahren für das Binary-Paintshop-Maximization-Problem im Durchschnitt also mindestens das 0.87856-fache des Optimalwertes. Auf der anderen Seite liegt der Wert der SDP-Relaxierung nicht über dem (1/0.87856)-fachen des Optimalwertes des Binary-Paintshop-Maximization-Problems.

Das eben vorgestellte Verfahren garantiert aufgrund seiner Zufallskomponente zunächst nur im Mittel seine gute Näherungslösung. Eine Möglichkeit ist jetzt, die Zufallserzeugung der Schnittebenen iteriert anzuwenden. Bei MAX-CUT sind beispielsweise so um $5n$ Iterationen üblich. Da die so ermittelten Lösungen im Allgemeinen lokal nicht optimal sind, schließt man oft eine lokale Suche an [15]. Eine weitere Möglichkeit besteht darin, das Verfahren zu derandomisieren. In [21] wird eine Methode vorgestellt, die die auf SDP basierenden Verfahren zur Lösung von MAX-CUT, MAX-2SAT und MAX DICUT derandomisiert und dabei die Approximationsgüte erhält, also die $0,87856$ für MAX-CUT, wobei die Methode Polynomialzeit benötigt. In der Arbeit wird zunächst gezeigt, wie der Karger-Motwani-Sudan-Färbungs-Algorithmus zum

Einfärben von 3-färbbaren Graphen mit möglichst vielen Farben derandomisiert werden kann. Anschließend wird begründet, warum die Methode auch für den MAX-CUT-Algorithmus von Goemans und Williamson angewendet werden kann. Da unser sehr ähnlicher Algorithmus die gestellten Bedingungen ebenso erfüllt, kann er auch derandomisiert werden.

4.6. Praktische Umsetzung

Die eben beschriebenen Ergebnisse wurden in einem MATLAB-Programm umgesetzt, welches vorgegebene Wörter mit Hilfe der SDP einfärbt. Der Quellcode ist im Anhang angegeben. Als SDP-Solver wird CSDP [22] verwendet, der in MATLAB eingebunden wurde. Die Bereitstellung der Eingabedaten für den Solver erfolgt im SeDuMi-Format. Wir werden die Arbeitsweise an einem kleinen Beispiel demonstrieren. Grundlage ist die Normalform des Problems, wir geben sie deshalb hier noch mal an:

$$\begin{array}{lll} SDPn \quad \min & C \bullet Y & \\ & A_i \bullet Y = 1 & \forall i \in \{1, \ldots, 2n\} \\ & B_{ij} \bullet Y = -1, & \forall i,j \text{ mit } a_i = a_j \text{ und } i < j \\ & Y \succcurlyeq 0. & \end{array}$$

Wir wollen als Beispiel das Wort $w = (a, b, c, b, a, c)$ einfärben. Die Eingabe des zu färbenden Wortes w in MATLAB erfolgt als Vektor in $\mathbb{N}^{2n}$, für unser Beispiel als $w = [1\,2\,3\,2\,1\,3]$. Die Matrix C berechnet sich zu $C = -\frac{1}{4}L$ (siehe S. 44). Somit folgt

$$C = -\tfrac{1}{4}\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix} = \begin{pmatrix} -0.25 & 0.25 & 0 & 0 & 0 & 0 \\ 0.25 & -0.5 & 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & -0.5 & 0.25 & 0 & 0 \\ 0 & 0 & 0.25 & -0.5 & 0.25 & 0 \\ 0 & 0 & 0 & 0.25 & -0.5 & 0.25 \\ 0 & 0 & 0 & 0 & 0.25 & -0.25 \end{pmatrix}.$$

Die A_i und B_{ij} ergeben sich zu

$$A_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \ldots, A_6 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ und}$$

$$B_{15} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, B_{24} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} B_{36} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 \end{pmatrix}.$$

Seien y_{ij} die Einträge von Y und y der Vektor in $\mathbb{R}^{4n^2\times 1}$, der entsteht, wenn wir in y die Zeilen aus Y nacheinander aufreihen und die Transponierte bilden, also
$y = (y_{11}\ \ y_{12}\ \ \dots\ y_{1,2n}\ \ y_{21}\ \ \dots\ y_{2n,2n})^T$.

Für die Eingabe in den Solver im SeDuMi-Format benötigen wir $c \in \mathbb{R}^{1\times 4n^2}$, $A \in \mathbb{R}^{3n\times 4n^2}$ und $b \in \mathbb{R}^{3n\times 1}$ so, dass folgendes gilt:
Einerseits soll $C \bullet Y = c \cdot y$ sein. Das bedeutet, für die Bildung von c müssen wir die Zeilen von C hintereinander schreiben. Sind c_{ij} die Einträge in C folgt $c = (c_{11}\ \ c_{12}\ \dots\ c_{1,2n}\ \ c_{21}\ \dots\ c_{2n,2n})$.
Für unser Beispiel ergibt sich somit
$c = (-0.25\ \ 0.25\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0.25\ \ -0.5\ \ 0.25\ \ 0\ \ 0\ \ 0\ \dots\ 0\ \ 0\ \ 0\ \ 0\ \ 0.25\ \ -0.25)$.
Weiterhin soll $A \cdot y = b$ unsere Randbedingungen $A_i \bullet Y = 1$ und $B_{ij} \bullet Y = -1$ abbilden. Für die Bildung von A erstellen wir von den einzelnen A_i und B_{ij} jeweils Zeilenvektoren der Länge $4n^2$, indem wir die einzelnen Zeilen dieser Vektoren hintereinander anordnen und anschließend schreiben wir diese Zeilenvektoren untereinander. Sind a^i_{rs} und b^{ij}_{rs} die Einträge in den Matrizen A_i und B_{ij} an der Stelle (r,s), können wir A beispielsweise bilden zu

$$A = \begin{pmatrix} a^1_{11} & a^1_{12} & \cdots & a^1_{1,2n} & a^1_{21}\cdots 0 & a^1_{2n,2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a^{2n}_{11} & a^{2n}_{12} & \cdots & a^{2n}_{1,2n} & a^{2n}_{21}\cdots 0 & a^{2n}_{2n} \\ b^{1,k}_{11} & b^{1,k}_{12} & \cdots & b^{1,k}_{1,2n} & b^{1,k}_{21} c \dots 0 & b^{1,k}_{2n,2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b^{l,m}_{11} & b^{l,m}_{12} & \cdots & b^{l,m}_{1,2n} & b^{l,m}_{21}\cdots 0 & b^{l,m}_{2n,2n} \end{pmatrix}, \text{ womit für unser Beispiel}$$

$$A = \begin{array}{c} \begin{matrix} j=1 & 3 & 8 & 11 & 13 & 15 & 22 & 24 & 26 & 29 & 34 & 36 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0.5 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 \end{pmatrix} \end{array} \text{ und } b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{pmatrix}$$

folgt. Dabei haben wir für in der Darstellung von A der Übersicht halber alle Nullspalten weggelassen und die Nummerierung der Spalten über die Matrix geschrieben. Mit A, b und c als Eingabe für den Solver erhalten wir schließlich die Lösung des Semidefiniten Programmes mit

$$Y = \begin{pmatrix} 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 & -1 & 1 \end{pmatrix}$$ und durch Normieren ergeben sich die Vektoren

$$v_1 \approx \begin{pmatrix} 0.4083 \\ -0.4083 \\ -0.4083 \\ 0.4083 \\ -0.4083 \\ 0.4083 \end{pmatrix}, v_2 \approx \begin{pmatrix} -0.4083 \\ 0.4083 \\ 0.4083 \\ -0.4083 \\ 0.4083 \\ -0.4083 \end{pmatrix} v_3 \approx \begin{pmatrix} -0.4083 \\ 0.4083 \\ 0.4083 \\ -0.4083 \\ 0.4083 \\ -0.4083 \end{pmatrix} v_4 \approx \begin{pmatrix} 0.4083 \\ -0.4083 \\ -0.4083 \\ 0.4083 \\ -0.4083 \\ 0.4083 \end{pmatrix},$$

$$v_5 \approx \begin{pmatrix} -0.4083 \\ 0.4083 \\ 0.4083 \\ -0.4083 \\ 0.4083 \\ -0.4083 \end{pmatrix} \text{ und } v_6 \approx \begin{pmatrix} 0.4083 \\ -0.4083 \\ -0.4083 \\ 0.4083 \\ -0.4083 \\ 0.4083 \end{pmatrix}.$$

Nun können wir mit $Z^*_{SDPn} = -C \bullet Y = 4.0$ den Optimalwert der semidefiniten Relaxierung bestimmen. Für die Lösung unseres eigentlichen Färbungsproblems müssen wir die v_i noch in zwei Mengen teilen, wobei wir zur Separation eine zufällige Hyperebene benutzen. Bei der Lösung unseres Beispiels sehen wir allerdings schon, dass die Vektoren v_i auf einer Geraden liegen, und jede Ebene, die einen Winkel ungleich null zu den Vektoren aufspannt, diese in $\{v_1, v_4, v_6\}$ und $\{v_2, v_3, v_5\}$ aufteilt. Somit erhalten wir als Färbung unseres Beispielwortes $(\dot{a}, \overline{b}, \overline{c}, \dot{b}, \overline{a}, \dot{c})$, also 4 Farbwechsel. Wir wollen das Ergebnis noch mit den Resultaten vergleichen, die wir mit dem Greedy-Algorithmus und dem Gucote-Algorithmus erhalten. Als Greedy-Färbung erhalten wir $(\dot{a}, \overline{b}, \dot{c}, \dot{b}, \overline{a}, \overline{c})$, also nur 3 Farbwechsel. Der Gucote-Algorithmus färbt über die Schritte $(\dot{a}, b, c, b, \overline{a}, c)$, $(\dot{a}, \overline{b}, c, \dot{b}, \overline{a}, c)$ das Wort mit $(\dot{a}, \overline{b}, \overline{c}, \dot{b}, \overline{a}, \dot{c})$, also mit 4 Farbwechseln.
Bei unserem gerade vorgestellten Beispiel hatten wir gesehen, dass die Lage der zufälligen Hyperebene auf die Färbung keinen Einfluss gehabt hatte. Dies ist im Allgemeinen nicht der Fall. Für das Wort w=(a,b,a,c,b,c) erhalten wir als Lösung

$$Y = \begin{pmatrix} 1 & 0 & -1 & 1 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ -1 & 0 & 1 & -1 & 0 & 1 \\ 1 & 0 & -1 & 1 & 0 & -1 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 1 & -1 & 0 & 1 \end{pmatrix}$$ und schließlich $v_1 = \begin{pmatrix} 0.5 \\ 0 \\ -0.5 \\ 0.5 \\ 0 \\ -0.5 \end{pmatrix}, v_2 \approx \begin{pmatrix} 0 \\ 0.7071 \\ 0 \\ 0 \\ -0.7071 \\ 0 \end{pmatrix},$

$$v_3 = \begin{pmatrix} -0.5 \\ 0 \\ 0.5 \\ -0.5 \\ 0 \\ 0.5 \end{pmatrix}, v_4 = \begin{pmatrix} 0.5 \\ 0 \\ -0.5 \\ 0.5 \\ 0 \\ -0.5 \end{pmatrix}, v_5 \approx \begin{pmatrix} 0 \\ -0.7071 \\ 0 \\ 0 \\ 0.7071 \\ 0 \end{pmatrix} \text{ und } v_6 = \begin{pmatrix} -0.5 \\ 0 \\ 0.5 \\ -0.5 \\ 0 \\ 0.5 \end{pmatrix}.$$

Mit $r = \begin{pmatrix} 1.7119 & -0.1941 & -2.1384 & -0.8396 & 1.3546 & -1.0722 \end{pmatrix}$ als Normalenvektor erhalten wir die beiden Mengen $S = \{v_1, v_4, v_5\}$ und $T = \{v_2, v_3, v_6\}$. Die Färbung des Wortes ergibt sich dann zu $(\dot{a}, \overline{b}, \overline{a}, \dot{c}, \dot{b}, \overline{c})$.
Für $r = \begin{pmatrix} 2.908 & 0.8252 & 1.37894 & -1.0581 & -0.4686 & -0.2724 \end{pmatrix}$ folgt die Teilung
$S = \{v_1, v_2, v_4\}$ und $T = \{v_3, v_5, v_6\}$ und damit die Färbung $(\dot{a}, \dot{b}, \overline{a}, \dot{c}, \overline{b}, \overline{c})$.
Bei längeren Wörtern lässt sich dann auch der Effekt feststellen, dass die Anzahl der Farbwechsel mit der Lage der zufälligen Ebene variieren kann. Für das Wort w=(a,b,c,d,a,c,e,b,f,d,f,e) erhält man als Optimalwert der semidefiniten Relaxierung $Z^*_{SDPn} = -C \bullet Y = 8.35$. Es wurden 100 Durchläufe des Programms mit jeweils neuen zufällig ermittelten Hyperebenen gemacht. Dabei wurden 81-mal 8 Farbwechsel, 12-mal 7 und 7-mal 6 Farbwechsel erzielt. Zum Vergleich, der Greedy-Algorithmus färbt das Wort mit $(\dot{a}, \overline{b}, \dot{c}, \overline{d}, \overline{a}, \overline{c}, \dot{e}, \dot{b}, \overline{f}, \dot{d}, \dot{f}, \overline{e})$, also 7 Farbwechsel. Mit dem Gucote-Algorithmus färben wir der Reihe nach a, b, d, e, c, f und erhalten so die Färbung $(\dot{a}, \overline{b}, \overline{c}, \dot{d}, \overline{a}, \dot{c}, \overline{e}, \dot{b}, \dot{f}, \overline{d}, \overline{f}, \dot{e})$ mit 8 Farbwechseln.

5. Zusammenfassung und Ausblick

Zu Beginn hatten wir gezeigt, dass wir unterschiedliche Konfigurationen der Spins eines antiferromagnetischen Arrays als Instanzen des Binary-Paintshop-Maximization-Problems auffassen können und die Suche nach einem Zustand minimaler Energie in einem antiferromagnetischen Array der Suche einer Färbung maximaler Farbwechsel in einem Wort, also dem Binary-Paintshop-Maximization-Problem, entspricht.
Wir konnten zeigen, dass das Binary-Paintshop-Maximization-Problem $\mathcal{NP}$-vollständig ist. Außerdem haben wir auch eine Reduktion vom Binary-Paintshop-Maximization-Problem auf das Binary-Paintshop-Minimization-Problem angegeben.
Anschließend haben wir einen einfachen Greedy-Algorithmus untersucht und festgestellt, dass er ein 0.5-Algorithmus ist, so dass BPMP in $\mathcal{APX}$ liegt.
Wir haben dann gezeigt, dass es Wörter der Länge $2n$ gibt, die sich nur mit rund $\frac{4}{3}n$ Farbwechsel einfärben lassen und konnten anschließend eine Methode angeben, die jedes Wort mit rund $\frac{4}{3}n$ Farbwechsel einfärbt.
Die Suche nach einem Beweis, dass BPMP $\mathcal{APX}$-hard ist, war leider nicht erfolgreich. Um dennoch BPMP hinsichtlich Approximierbarkeit einzuordnen, wurde eine L-Reduktion von BPMP auf kubische Graphen einer speziellen Struktur gezeigt. Wir haben dann BPMP so umformuliert, dass man nicht mehr die Farbwechsel maximiert, sondern die Farbwechselfehler minimiert. Dann ist dieses Problem $\mathcal{APX}$-hard.
Wir haben uns anschließend der Semidefiniten Programmierung, dem Schwerpunkt dieser Arbeit, zugewandt. Nach einer Einführung haben wir aufgezeigt, dass BPMP mit Hilfe der Semi-

definiten Programmierung bearbeitet werden kann. Dabei haben wir einen Weg angegeben, wie dabei vorgegangen werden kann. Somit lässt sich die Hauptfragestellung dieser Arbeit positiv beantworten. Die erarbeite Methodik wurde anschließend in ein MATLAB-Programm überführt, welches mit Hilfe des Solvers CSDP vorgegebene Wörter mit Hilfe der SDP einfärbt, so dass für die Problemstellung auch ein praktisches Tool zur Verfügung steht.

Einige Fragestellungen blieben offen. So wurde für den Greedy-Algorithmus eine Vermutung für den Erwartungswert der durchschnittlich erreichbaren Farbwechsel angegeben, bewiesen konnte die Vermutung im Rahmen dieser Arbeit nicht. Außerdem, wie eben schon beschrieben, konnte nicht die Frage geklärt werden, ob das Binary-Paintshop-Maximization-Problem $\mathcal{APX}$-vollständig ist.

A. Anhang

A.1. Quellcode zum Programm Paintmax

Das Programm dient zum Einfärbung von Wörter als Instanzen des BPMP mit Hilfe des SDP-Solver CSDP. Damit das Programm läuft, müssen die Suchpfade in Matlab so eingestellt sein, dass auf den Installationsort von CSDP zugegriffen werden kann. Das zu färbende Wort wird als Zeilenvektor in $\mathbb{N}^{2n}$ mit dem Namen w eingegeben. Anschließend wird das Programm mit der Eingabe «Paintmax»gestartet. Als Ausgabe erfolgt ein gefärbter String.

```
%Programm faerbt Woerter w des Binary-Paintshop-Maximization-Problem.
%Zunaechst muss das Wort eingegeben werden als Zeilenvektor w,
%also zum Beipiel als w=[1 2 3 2 1 3] fuer das Wort w=(a,b,c,b,a,c).


n=0.5*length(w);    % Bestimmung Anzahl der Buchstaben

g=zeros(2,n);       % g enthaelt Positionen der Buchstaben in w
for i = 1:n         % 1. Zeile Pos. der 1. Letter, 2. Zeile Pos. der 2.
    for j = 1:2*n
      if w(1,j)==i
          if g(1,i)==0
            g(1,i)=j;
          else
            g(2,i)=j;
          end
      end
    end
end

A=zeros(3*n,4*n*n);
AA=zeros(2*n);
for i = 1:2*n
    for j = 1:2*n
      if w(1,j)==w(1,i)
        AA(i,j)=1;
      end
    end

end

for i = 1:n
    AA1=AA(:,g(1,i))*AA(g(1,i),:);
    t=1:size(AA1)+1:numel(AA1);
    AA1(t)=0;
```

```
    AA2=reshape(AA1,1,4*n*n);
    A(i+2*n,:)=0.5*AA2(1,:);   % Bedingung Lettern gleicher Buchstaben
                               % sind entgegengesetzt.
end

b=ones(1,3*n);                 % Bedingung Diagonale von X ist 1.
b(:,2*n+1:3*n)=-1;             % Bedingung Lettern gleicher Buchstaben sind
                               % entgegengesetzt.

for i = 1:2*n                  % Bedingung Diagonale von X ist 1 (A*X=b)
    A(i,t(i))=1;
end
K.s=2*n;
L=2*eye(2*n);                  % Bildung der Laplace-Matrix.
L(1,1)=1;
L(2*n,2*n)=1;
for i = 2:2*n
      L(i,i-1)=-1;
      L(i-1,i)=-1;
end

C1=0.25*L;                     % Kostenvektor
c=reshape(-1*C1,1,4*n*n);      % Kostenvektor im SeDuMi-Format
[x,y,info] = csdp(A,b,c,K);    % Loesung SDP durch CSDP
k=c*x;                         % Kosten des SDP
X1=reshape(x,2*n,2*n);         % X1 ist Loesung als Semidefinite Matrix
V = sqrtm(X1);                 % Loesungsvektoren v_i als Zeilen von V

u=zeros(2*n,1);
kq_max=1;

l=1;                           % Gibt an wie oft ein Schnitt erzeugt werden soll.
for j = 1:l                    % Bei l>1 wird Lsg. mit max. Farbwechsel gewaehlt
  r=randn(1,2*n);              % r ist eine Zufaellige Ebene in R^2n
  E=r*V;                       % Knoten mit gleichem Vorzeichen in E liegen
  for i = 1:(2*n)              % auf der selben Seite des Schnittes.
    if E(1,i)>=0
        u(i,1) = 1;            % u ist Teilungsvektor.
    else
        u(i,1) = -1;
    end
  end
  Xq=u*u.';                    % Loesungsmatrix im quadratischen Problem.
  kq=0.25*trace(L*Xq);         % Anzahl Farbwechsel nach Rundung.
```

```
   if kq>=kq_max
        kq_max=kq;
        u_max=u;
   end
end

fprintf('\n')
for i=1:2*n                 % Ausgabe eines gefaerbten Strings.
        hh=w(i);            % Letter i wird entsprechend Vorzeichen von u(i)
        s = num2str(hh); % gefaerbt.
     if u_max(i)==1;
        cprintf('red', s);
     else
        cprintf('blue', s);
     end
end
fprintf('\n')
```

Literatur

[1] E. Ising, Beitrag zur Theorie des Ferromagnetismus, Zeitschrift für Physik, Band 31 (1925), 253–258

[2] H. Kawamura, Two models of spin glasses - Ising versus Heisenberg, Journal of Physics: Conference Series 233 (2010) 012012

[3] M. Goemans, D. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, Journal of the ACM, 42 (6): 1115–1145

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

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

[6] W. Hochstättler, Note on the Complexity of the Binary Paintshop Maximization Problem (BPMP), Unveröffentlichtes Dokument (2017)

[7] 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

[8] R. M. Karp, Reducibility Among Combinatorial Problems, Proceedings of the 9th International IPCO Conference on Integer Programming and Combinatorial Optimization (2002): 67-82

[9] W. Hochstättler, Lineare Optimierung, FernUniversität in Hagen (2013)

[10] I. Wegener, Komplexitätstheorie: Grenzen der Effizienz von Algorithmen, Springer, 2003

[11] P. Alimonti, V. Kann, Some APX-completeness results for cubic graphs, Theoret. Comput. Sci. 237 (1–2) (2000) 123–134

[12] W. Hochstättler, Semidefinite Programmierung, frei nach dem Skriptum des Kurses von F. Alizadeh im RUTCOR Januar, Februar 1995, 2017

[13] M. Lewin, D. Livnar, U. Zwick, Improved Rounding Techniques for the MAX 2-SAT and MAX DI-CUT Problems, In R. E. Miller; J. W. Thatcher; J.D. Bohlinger. Complexity of Computer Computations (1972): 85-103

[14] H. Wolkowicz, R. Saigal, L. Vandenberghe, Handbook of Semidefinite Programming: Theory, Algorithms, and Applications, International Series in Operations Research and Management Science, Band 27, 2000

[15] B. Hirschfeld, Approximative Lösungen des MAX-CUT-Problems mit semidefiniten Programmen, Dissertation zur Erlangung des Doktorgrades der Mathematisch-Naturwissenschaftlichen Fakultät der Heinrich-Heine-Universität Düsseldorf (2004)

[16] P. Lancaster, M. Tismenetsky, The theory of matrices, Academic Press, 1985

[17] F. Jarre, J. Stoer, Optimierung, Springer, 2004

[18] L. Vandenberghe, S. Boyd, Convex Optimization, Cambridge University Press, 2004

[19] L. Vandenberghe, S. Boyd, Semidefinite Programming, SIAM Review Vol. 38, No. 1, (1996), 49-95

[20] Wei Chen, Ji Liu, Yongxin Chen, Sei Zhen Khong, Dan Wang, T. Basar, Li Qiu, K.-H. Johansson, Characterizing the Positive Semidefiniteness of Weighted Laplacians via Generalized Effective Resistance, Conference Paper (2016)

[21] S. Mahajan, H. Ramesh, Derandomizing Semidefinite Programming Based Approximation Algorithms., In: Proceedings of the 36th Annual Symposium on Foundations of Computer Science IEEE (1995), 162–169

[22] B. Borchers, CSDP 6.2.0 User's Guide, in: https://github.com/coin-or/Csdp/blob/master/doc/csdpuser.pdf (2017), abgerufen am 27.07.2019

Abbildungsverzeichnis

1. Ein Array von 6 gekoppelten Spins 6
2. Zwei Färbungen für das Wort $w = (a, b, c, d, a, c, b, d)$ 8
3. Gegebenes Wort w_1 mit Färbung f^1 und w_2 mit Färbung f^2 12
4. Gegebenes Wort w_2 mit Färbungen f^2 und f^{2*}, w_1 mit Färbung f^1 12
5. Gegebenes Wort w_1 mit Färbungen f^1, w_2 mit Färbung f^2 13
6. Wort w_2 mit Färbung f^2, w_2 mit f^{2*}, w_1 mit f^1 14
7. Wort w mit Greedy-Färbung und optimaler Färbung 15
8. Wort w mit Greedy-Färbung und optimaler Färbung 16
9. Wort w mit Menge unabhängiger ungerader Intervalle 17
10. Zwei Wörter mit $\gamma(w) = \frac{4}{3}n - 1$ 18
11. Wort $w = (a, b, c, b, c, d, d, a)$ und der entsprechende Graph G_w 36

Erklärung

Name:	Mario Quick
Matrikel-Nr.:	7564163
Fach:	Mathematik
Modul:	Bachelorarbeit

Ich erkläre, dass ich die vorliegende Abschlussarbeit mit dem Thema

Energieminimierung eindimensionaler Arrays gekoppelter Spins in einem Antiferromagneten

selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich, inhaltlich oder sinngemäß entnommenen Stellen als solche den wissenschaftlichen Anforderungen entsprechend kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für Zeichnungen, Skizzen oder graphische Darstellungen. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Arbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate überprüft und ausschließlich für Prüfungszwecke gespeichert wird.

Datum Unterschrift