A cutting plane algorithm for graph coloring

Ein Schnittebenenverfahren zur Färbung von Graphen

Bachelor-Thesis von Lena Maria Schwan aus Hachenburg

Oktober 2010

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Mathematik
AG Optimierung

A cutting plane algorithm for graph coloring
Ein Schnittebenenverfahren zur Färbung von Graphen

Vorgelegte Bachelor-Thesis von Lena Maria Schwan aus Hachenburg

1. Gutachten: Dr. habil. Marko Lübbecke
2. Gutachten: Prof. Dr. Stefan Ulbrich

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 14. Oktober 2010

(Lena Maria Schwan)

Inhaltsverzeichnis

1 Das Graphenfärbungsproblem

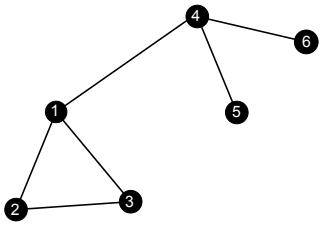
1.1 Definitionen

Das Graphenfärbungsproblem ist eines der berühmtesten Probleme der diskreten Mathematik. Dabei sollen die Knoten eines ungerichteten Graphen $G = (V, E)$ mit möglichst wenigen Farben so gefärbt werden, dass adjazente Knoten nie die gleiche Farbe zugeordnet bekommen. Das Graphenfärbungsproblem ist eine Knotenmakierung, das heißt, es gibt eine Abbildung $f : V \to S$, die jedem Knoten $v_i \in V$ eine Farbe aus der Farbenmenge S zuordnet.

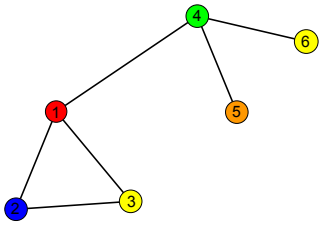
Definition:
*Eine Abbildung $f : V \to S$ heißt **Färbung** des Graphen $G = (V, E)$, wenn gilt:*
$f(v_i) \neq f(v_j)\ \forall\ v_i, v_j \in V$ *mit* $v_i \neq v_j$ *und* $(v_i, v_j) \in E$ *[1]*

Definition:
*Ein Graph $G = (V, E)$ heißt **k-färbbar**, wenn eine Färbung mit einer k-elementigen Farbenmenge S existiert. Das kleinste k, für das der Graph G eine Färbung besitzt, nennt man die **chromatische Zahl** $\chi(G)$ von G [2]*

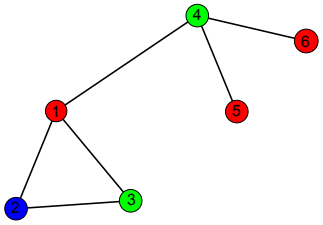
Beispiel 1:
Sei S ={blau, grün, rot, gelb, orange, pink, lila, braun} die Farbmenge und $G = (V, E)$ der folgende Graph:



Der Graph ist 5-färbbar, denn man kann ihn mit der 5-elementigen Farbmenge S={blau, grün, rot, gelb, orange} einfärben:



Man kann ihn aber auch mit weniger Farben färben, denn er hat die chromatische Zahl $\chi(G) = 3$. Es müssen also mindestens 3 Farben verwendet werden, um eine gültige Färbung zu erlangen. Mit minimaler Anzahl an Farben könnte man ihn zum Beispiel so färben:



Definition:
*$U \subseteq V$ ist eine **unabhängige Knotenmenge** genau dann, wenn für alle $i, j \in U$ gilt: $(i, j) \notin E$. Das heißt, keine zwei Knoten in U sind mit einer Kante verbunden. [2]*

Eine k-Färbung entspricht einer Aufteilung des Graphen in k unabhängige Knotenmengen, wobei jeder Knotenmenge eine Farbe zugeordnet wird.

Eine obere Schranke für die chromatische Zahl eines Graphen $G = (V, E)$ ist die Anzahl der Knoten n. Es gibt zwar viele Graphen, die sich mit weniger als n Farben färben lassen, aber bei vollständigen Graphen benötigt man genau n Farben, da alle Knoten adjazent zueinander sind. Eine untere Schranke lässt sich also anhand der vollständigen Teilgraphen $G' = (V', E')$, den sogenannten Cliquen, ableiten. Die Anzahl der Knoten der größten Clique des Graphen ist eine untere Schranke für die chromatische Zahl von G.

Im Beispiel 1 hat die größte Clique 3 Knoten, das heißt in diesem Beispiel wird die untere Schranke angenommen.

Das Färbungsproblem gehört zu der Klasse der NP-schweren Probleme. Durch die Frage:“Gibt es zu dieser Instanz eine Lösung, die besser als x ist?“, kann man jedes Optimierungsproblem in polynomialer Zeit in ein Entscheidungsproblem umwandeln. Ein Optimierungsproblem heißt NP-schwer, falls das zugehörige, NP-vollständige Entscheidungsproblem Π in polynomialer Zeit gelöst werden kann, wenn das Optimierungsproblem in polynomialer Zeit gelöst werden kann. Da das zugehörige Entscheidungsproblem in der Klasse der NP-vollständigen Probleme liegt, ist bisher kein Algorithmus bekannt, der das Problem in polynomieller Zeit lösen kann. [3]

Weitere nützliche Definitionen:

Definition:
*Eine Menge $C \subset \mathbb{R}^n$ heißt **konvex**, wenn mit je zwei Punkten aus C auch die gesamte Verbindungsstrecke zwischen den Punkten in C liegt, das heißt wenn aus $x_1, x_2 \in C$ und $\lambda \in [0, 1]$ folgt: $\lambda x_1 + (1 - \lambda)x_2 \in C$. [6]*

Definition:
*Der Durchschnitt aller konvexen Mengen, die M enthalten, heißt die **konvexe Hülle** von M und wird mit conv M bezeichnet. Es gilt: Die konvexe Hülle einer Menge M ist die Menge aller Konvexkombinationen von Punkten aus M. [6]*

1.2 Anwendungen des Graphenfärbungsproblems

Die berühmteste Anwendung der Graphenfärbung ist das Vierfarbenproblem. Dabei geht es darum, zu zeigen, dass jede Landkarte mit 4 Farben so färbbar ist, dass Länder mit gemeinsamer Grenze nie die gleiche Farbe haben. Jede Landkarte lässt sich als planarer Graph darstellen, das heißt, man kann ihn so in der Ebene darstellen, dass sich keine Kanten schneiden [1]. Jedes Land entspricht dabei einem Knoten. Zwei Knoten sind durch eine Kante verbunden, wenn die zugehörigen Länder eine gemeinsame Grenze haben. Um diese Aussage zu beweisen, muss man also zeigen, dass alle planaren Graphen die chromatische Zahl 4 haben. Dies wurde bis heute aber nur „bewiesen“, indem ein Rechner die 4-Färbung für 1936 Graphen, auf die sich alle planaren Graphen reduzieren lassen, errechnet hat.

Auch das im Moment so beliebte Rätsel „Sudoku“ könnte anhand der Graphenfärbung gelöst werden. Dazu benötigt man einen Graph mit 81 Knoten (ein Knoten für jedes auszufüllende Kästchen). Zwei Knoten werden verbunden, wenn sie:

- in einer Reihe oder Spalte stehen
- beide in einem der kleineren Vierecke sind.

Gesucht ist eine 9-Färbung, wobei durch die schon zu Beginn eingetragenen Zahlen die Aufteilung in unabhängige Knotenmengen eingeschränkt wird.

Man kann die Graphenfärbung auch sehr gut benutzen, um Zeitpläne zu erstellen. Man könnte z.B. den optimalen Zeitplan für die Ausschusssitzungen eines Parlamentes erstellen, bei dem jeder Ausschuss einen Tag lang tagen will und es möglich ist, dass mehrere Personen in verschiedenen Ausschüssen sitzen. Man erstellt einen Graphen G, in dem jeder Knoten einen Ausschuss repräsentiert. Zwei Ausschüsse, die nicht gleichzeitig tagen können, da es Mitglieder gibt, die in beiden Gremien sitzen, verbindet man mit einer Kante. Sie bekommen also bei der Lösung durch einen Färbungsalgorithmus eine unterschiedliche Farbe zugeordnet. Interpretiert man nun die Farben als die unterschiedlichen Tage, ist garantiert, dass Mitglieder des Parlamentes nie gleichzeitig in mehreren Ausschüssen sitzen müssen. Die chromatische Zahl des erstellten Graphen entspricht der minimalen Zeit, die für die Ausschusstagungen angesetzt werden muss.

Es gibt noch viele weitere Anwendungsbereiche für das Färbungsproblem in der Praxis. Deshalb ist es wichtig, einen guten Algorithmus zur Lösung dieses Problems zu finden oder bestehende Algorithmen zu verbessern.

1.3 Modellierung

Für den Graphen $G = (V, E)$ mit n Knoten und m Kanten lässt sich das Problem folgendermaßen modellieren [4]:

Sei $x_{ij} \in \{0, 1\}$ mit $i \in V$ und $1 \leq j \leq n$ (da man höchstens n Farben benötigt, um G zu färben).
Es gilt: $x_{ij} = \begin{cases} 1, & \text{wenn Knoten i die Farbe j hat} \\ 0, & \text{sonst} \end{cases}$

Sei außerdem $w_j \in \{0, 1\}$ mit $1 \leq j \leq n$ und es gilt:
$w_j = \begin{cases} 1, & \text{falls mind. einem Knoten die Farbe j zugeordnet ist} \\ 0, & \text{falls keinem Knoten die Farbe j zugeordnet ist} \end{cases}$

Man kann das Problem nun als ganzzahliges Minimierungsproblem darstellen:

$$\min \sum_{j=1}^{n} w_j$$
s.t.

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i \in V \tag{1.1}$$

$$x_{ij} + x_{kj} \leq w_j \qquad \forall\, (i,k) \in E,\ 1 \leq j \leq n \tag{1.2}$$

$$x_{ij} \in \{0,1\} \quad \forall\, i \in V,\ 1 \leq j \leq n$$

$$w_j \in \{0,1\} \quad 1 \leq j \leq n$$

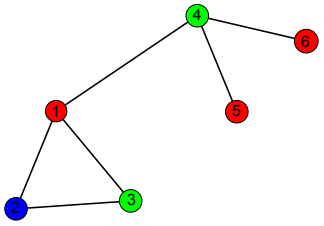
Wegen Nebenbedingung (1) wird jedem Knoten genau eine Farbe zugeordnet. Nebenbedingung (2) stellt sicher, dass benachbarte Knoten nicht die gleiche Farbe zugeordnet bekommen, denn wird die Farbe j überhaupt nicht benutzt, dann muss gelten: $x_{ij} = x_{kj} = 0$. Wird die Farbe j benutzt, muss $w_j = 1$ sein und es darf höchstens eine der beiden Variablen den Wert 1 haben. Da diese Bedingung für alle $(i,j) \in E$ gelten muss, ist dieses ganzzahlige Problem eine Modellierung für das Färbungsproblem. Sei $\mathscr{SCP}$ die Menge aller Lösungen dieses Problems. Eine Lösung des Problems ist dabei eine zulässige Belegung $(x_{11}, \ldots, x_{nm}, w_1, \ldots, w_m)$ der Variablen.

Diese Modellierung ist oft nicht geeignet, um das Färbungsproblem mit einem Algorithmus zu lösen, da es bei einigen Graphen sehr lange dauern würde, bis eine optimale Lösung gefunden wird. Das Problem dieser Modellierung ist, dass es zu viele zulässige, aber symmetrische Lösungen gibt. Symmetrische Lösungen haben das gleiche Endergebnis, eine k-Färbung. Allerdings benutzen sie dafür zum Beispiel andere Farben oder Permutationen der Farben anderer Lösungen oder eine unterschiedliche Aufteilung in unabhängige Mengen. Es gibt also viele verschiedene zulässige Färbungen des Graphen G mit Farben aus der zugehörigen Farbmenge S. Dies ist aber für das Ergebnis, also die Lösung des Problems, nicht wichtig. Meistens ist es nur wichtig herauszufinden, was die chromatische Zahl des Graphen ist oder ob es eine k-Färbung gibt und nicht wieviele k-Färbungen zu gegebener Farbmenge existieren. Will man also nur eine mögliche k-Färbung finden, so ist es sinnvoll, solche symmetrischen Lösungen auszuschließen.

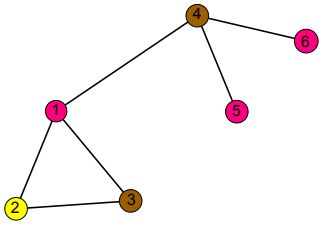
1.3.1 Color order model

Beispiel 2:
Sei $G = (V, E)$ der Graph aus Beispiel 1. Sei S={blau, grün, rot, gelb, orange, pink, lila, braun} die Farbmenge. Gesucht wird eine 3-Färbung mit Farben aus der Farbmenge S.

Eine mögliche Färbung aus $\mathscr{SCP}$ ist:



Dies ist eine andere Färbung aus $\mathscr{SCP}$, die genau die gleiche Anzahl an Farben wie die Färbung vorher benutzt, aber andere Farben aus der Farbmenge auswählt:



Da uns beim Färbungsproblem jedoch nur interessiert, wieviele Farben man benötigt, um den Graphen einzufärben und nicht, welche Farben man benutzt, reicht es, für eine k-Färbung nur die ersten k Farben zu betrachten.

Um diese Einschränkung zu beachten, muss man zu den Nebenbedingungen (1) und (2) noch weitere hinzufügen:

$$w_j \leq \sum_{i \in V} x_{ij} \qquad \forall\ i \leq j \leq n \qquad (3)$$

$$w_j \geq w_{j+1} \qquad \forall\ 1 \leq j \leq n \qquad (4)$$

Nebenbedingung (3) stellt sicher, dass w_j kleiner ist als die Summe der Knoten, die mit der Farbe j gefärbt werden. Wenn kein Knoten die Farbe j hat, muss $w_j = 0$ sein. Nebenbedingung (4) schränkt die Farbwahl ein, denn durch sie wird garantiert, dass eine Farbe nur benutzt werden darf, wenn die Farbe, die in der Reihenfolge vor ihr liegt, schon benutzt wurde. Dadurch wird sichergestellt, dass für eine k-Färbung nur die ersten k Farben aus der Farbmenge benutzt werden und symmetrische Lösungen, wie in Beispiel 2, ausgeschlossen werden.

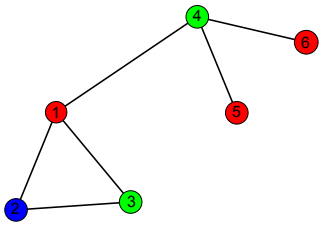
Sei $\mathscr{CP} = \text{conv}(\mathscr{SCP} \cap \{(x,w) : (x,w) \text{ erfüllt (3) und (4)}\})$.
Die Anzahl der zulässigen Lösungen verringert sich durch diese beiden neuen Nebenbedingungen sehr. Es sei eine Aufteilung in unabhängige Mengen gegeben, mit u vielen Mengen. In der Farbmenge sind $|S|$ viele Farben. Damit gab es vor dieser Einschränkung $\binom{|S|}{u} \cdot u!$ Möglichkeiten, den Graph zu färben. Jetzt gibt es nur noch $u!$ Möglichkeiten. Für den Graph und die Wahl der unabhängigen Mengen aus Beispiel 2 ergibt sich eine Verringerung von 56 auf 6 Möglichkeiten. Beachten man, dass normalerweise keine Aufteilung in unabhängige Mengen vorgegeben ist, verringert sich die Anzahl der Lösungen also für jede mögliche Aufteilung, was die Lösungsmenge erheblich verkleinert.

1.3.2 Independent set order model 1

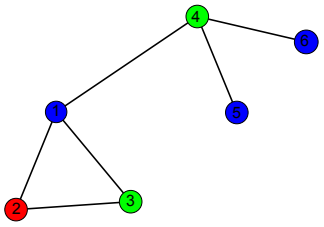
Trotzdem sind in der Lösungsmenge noch viele symmetrische Lösungen enthalten, wie das folgende Beispiel zeigt:

Beispiel 3:
Man betrachte wieder den Graphen aus Beispiel 1 und 2 und die Farbmenge S={blau, grün, rot, gelb, orange, pink, lila, braun}. Gesucht ist eine 3-Färbung des Graphen. Durch die neu hinzugefügten Nebenbedingungen (3) und (4) wurde die Farbwahl eingeschränkt, das heißt, es sind nur noch solche Färbungen zulässig, die nur die ersten 3 Farben, also blau, grün und rot benutzen.

Eine mögliche Färbung aus $\mathscr{CP}$ ist:



Eine weitere Lösung aus $\mathscr{CP}$ ist:



Die verschiedenen möglichen Lösungen entstehen durch Permutionen der ersten k Farben. Um einige davon auszuschließen, fügt man zu den Nebenbedingungen (1) und (2) folgende Nebenbedingungen hinzu:

$$w_j \leq \sum_{i \in V} x_{ij} \qquad \forall\ 1 \leq j \leq n \qquad (5)$$

$$\sum_{i=1}^{n} x_{ij} \geq \sum_{i=1}^{n} x_{ij+1} \qquad \forall\ 1 \leq j \leq n-1 \qquad (6)$$

Nebenbedingung (5) ist identisch zu Nebenbedingung (3). Nebenbedingung (6) bewirkt, dass die Farbe j immer mehr Knoten färbt, als die Farbe $j+1$. Hat man also eine Aufteilung des Graphen in unabhängige Knotenmengen und sortiert diese absteigend nach der Anzahl der enthaltenen Knoten, so ist sichergestellt, dass einer der Knotenmengen mit maximaler Anzahl an Knoten die Farbe 1 zugeordnet wird und einer Knotenmenge mit minimaler Anzahl die Farbe k. Gibt es nur eine Sortierung, also enthalten alle unabhängigen Mengen eine unterschiedliche Anzahl an Knoten, so ist auch die Farbverteilung eindeutig.

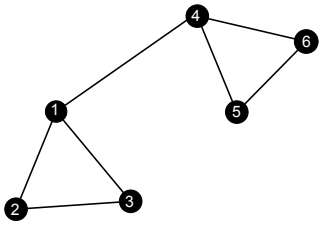
Sei $\mathscr{CP}1 = \text{conv}(\mathscr{SCP} \cap \{(x,w) : (x,w) \text{ erfüllt (4) und (6)}\})$.
$\mathscr{CP}1$ ist eine Teilmenge von $\mathscr{CP}$, denn jede Lösung aus $\mathscr{CP}1$ benutzt für eine k-Färbung nur die ersten k Farben, da die Farbe j mehr Knoten einfärben muss als die Farbe $j+1$. Dadurch kann man keine Farbe „überspringen“, da sonst alle darauf folgenden Farben nicht mehr benutzt werden dürfen.

Die Anzahl der symmetrischen Lösungen in $\mathscr{CP}1$ wird umso kleiner, je mehr unabhängige Knotenmengen mit unterschiedlicher Anzahl an Knoten es gibt.

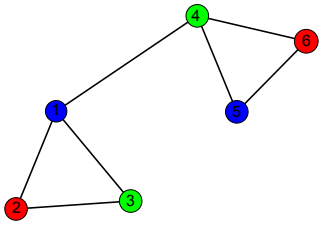
1.3.3 Independent set order model 2

Wird der Graph aber so in unabhängige Knotenmengen aufgeteilt, dass einige die gleiche Anzahl an Knoten haben, so entstehen weiterhin viele symmetrische Lösungen durch Permutationen. Denn immer wenn mehrere unabhängige Knotenmengen gleich viele Knoten enthalten, kann man sie in der Sortierung der Knotenmengen vertauschen, sodass es mehrere mögliche Färbungen gibt, die die Nebenbedingungen nicht verletzen.

Beispiel 4:
Sei S={rot, grün, blau, gelb} und G=(V,E) folgendermaßen:



Eine Unterteilung in unabhängige Knotenmengen mit anschließender Färbung könnte so aussehen:



Bei diesem Graphen sind nun alle unabhängigen Knotenmengen gleich groß. Die Sortierung der Knotenmengen ist also nicht eindeutig. Es gibt für diese Aufteilung 3! mögliche Sortierungen. Das heißt, jede Menge kann auch eine andere der ersten 3 Farben annehmen, ohne dass eine Nebenbedingung verletzt wird, sodass für diese Aufteilung 6 äquivalente Lösungen entstehen.

Um auch diese Symmetrien für eine bestimmte Aufteilung in unabhängige Knotenmengen auszuschließen, geht man folgendermaßen vor:

- sortiere die unabhängigen Knotenmengen nach dem kleinsten enthaltenen Knoten
- erlaubt sind nur Färbungen, die der j-ten unabhängigen Knotenmenge die j-te Farbe zuweisen

Die entsprechenden Nebenbedingungen, die hinzugefügt werden müssen, lauten:

$$x_{ij} = 0 \qquad \forall\ j \geq i+1 \qquad (7)$$

$$x_{ij} \leq \sum_{k=j-1}^{i-1} \qquad \forall\ i \in V \setminus \{1\},\ \forall\ 2 \leq j \leq i-1 \qquad (8)$$

Der Knoten i kann höchstens in der i-ten unabhängigen Knotenmenge sein und zwar genau dann, wenn die Knoten 1 bis i eine Clique bilden. Nebenbedingung (7) besagt nun, dass der Knoten i nur Farben bis i annehmen darf, da die Farbnummer der Nummer der unabhängigen Menge entspricht, zu der i gehört. So kann der Knoten 1 zum Beispiel nur die Farbe 1 annehmen, da er auf jeden Fall in der unabhängigen Menge ist, die bei der Sortierung an erster Stelle steht. Nebenbedingung (8) stellt sicher, dass die Farbe j nur dann benutzt werden darf, wenn die Farbe $j-1$ schon benutzt wurde. Es reicht dabei, die Summe erst bei $k = j-1$ zu beginnen, da alle anderen Knoten nicht in der Knotenmenge $j-1$ sein können.

Sei $\mathscr{CP}2 = \text{conv}(\mathscr{SCP} \cap \{(x,w):(x,w) \text{ erfüllt (7) und (8)}\})$.
Für jede mögliche Aufteilung in unabhängige Knotenmengen ist die Lösung jetzt eindeutig, denn es gibt keine symmetrischen Lösungen mehr, die durch die Äquivalenz der Farben entstehen, da die Zuweisung eindeutig ist. Allerdings gibt es weiterhin noch mehrere Möglichkeiten, den Graphen einzufärben, denn es gibt meistens mehrere Aufteilungen in unabhängige Knotenmengen.

$\mathscr{CP}2$ ist eine Teilmenge von $\mathscr{CP}$, denn Nebenbedingung (8) bewirkt, dass für eine k-Färbung nur die ersten k Farben der Farbmenge benutzt werden dürfen.

Die Symmetrie, die durch verschiedene Aufteilungen in unabhängige Mengen entsteht, könnte man zum Beispiel beheben, indem man aus allen möglichen Aufteilungen die Aufteilung auswählt, die bei minimaler Anzahl an Mengen möglichst viele Mengen unterschiedlicher Größe besitzt. Kommen so mehrere Aufteilungen in Betracht, kann man zum Beispiel die Aufteilung wählen, bei der möglichst viele Knoten in der unabhängigen Menge mit Knoten 1 sind. Wenn es trotzdem mehrere Möglichkeiten gibt, wendet man das gleiche Kriterium auf die unabhängige Menge mit dem kleinsten Knoten an, der nicht in der vorher betrachteten Menge ist und so weiter. Gibt es weiterhin mehrere Möglichkeiten, zieht man die Aufteilung vor, bei der die kleineren Knoten in der ersten Menge sind, beziehungsweise in einer späteren, wenn bis dahin alle Knoten gleich waren. So könnte man auch diese Symmetrie noch beheben und zu einer wirklich eindeutigen Lösung gelangen.

1.4 Bewertung der 3 Modelle

Die 3 Ansätze eliminieren symmetrische Lösungen anhand verschiedener Kriterien. Normalerweise bevorzugt man ein Modell mit möglichst wenigen symmetrischen Lösungen, da es das Problem am besten wiederspiegelt. Dies würde bedeuten, dass man die dritte Modellierung den anderen Beiden vorziehen würde, da $\mathscr{CP}2$ am wenigesten symmetrische Lösungen enthält. Will man dieses Modell aber benutzen, um mit einem Algorithmus eine Lösung zu erzielen, sind auch andere Kriterien wichtig, zum Beispiel die Anzahl der Variablen, die Anzahl der Nebenbedingungen und der benötigte Speicherplatz, denn davon hängt zum Beispiel die Schnelligkeit eines LP-Solvers auf der zugehörigen LP-Relaxation ab. Man muss also die Vor- und Nachteile der 3 Modelle abwägen, um zu einem Modell zu gelangen, das den Ansprüchen des Benutzers entspricht. Es kann also sein, dass für einen Algorithmus die erste Modellierung geeigneter ist, obwohl die zugehörige Lösungsmenge größer ist, als die der anderen beiden Modellierungen. Man muss also bei der Entwicklung eines Algorithmus zur Lösung des Färbungsproblems nicht nur die einzelnen Komponenten des Algorithmus verbessern, sonder auch unterschiedliche Modellierungen betrachten, um so zu einer für den Benutzer optimalen Lösung zu gelangen.

2 LP-Relaxation und Schnittebenenverfahren

2.1 LP-Relaxation

Eine Relaxation ist eine Vereinfachung. Die Idee ist, einen Teil des Problems wegzulassen, sodass das Problem wesentlich einfacher wird. Die LP-Relaxation ist eine solche Vereinfachung. Bei der Modellierung des Färbungsproblems wurden die Variablen x_{ij} und w_j benutzt, wobei x_{ij} den Wert 1 annimmt, wenn der Knoten i mit der Farbe j gefärbt wird. Sonst gilt $x_{ij} = 0$. Wenn die Farbe j benutzt wird, hat w_j den Wert 1, sonst 0. Zwei Nebenbedingungen unseres Problems lauten also: $x_{ij} \in \{0,1\}$ und $w_j \in \{0,1\}$. Bei der LP-Relaxation werden die Ganzzahligkeitsbedingungen weggelassen. Das heißt, dass zum Beispiel aus einer Nebenbedingung $x \in \mathbb{Z}$ durch die LP-Relaxation $x \in \mathbb{R}$ wird. In unserem Fall werden aus den Nebenbedingungen $x_{ij} \in \{0,1\}$ und $w_j \in \{0,1\}$ die neuen Nebenbedingungen $0 \leq x_{ij} \leq 1$ und $0 \leq w_j \leq 1$. Durch diese Vereinfachung erhält man das folgende lineare Grundmodell:

$$\min \sum_{j=1}^{n} w_j$$

s.t.

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i \in V \tag{1}$$

$$x_{ij} + x_{kj} \leq w_j \qquad \forall\, (i,k) \in E,\ 1 \leq j \leq n \tag{2}$$

$$0 \leq x_{ij} \leq 1 \qquad \forall\, i \in V,\ 1 \leq j \leq n$$

$$0 \leq w_j \leq 1 \qquad \text{für } 1 \leq j \leq n$$

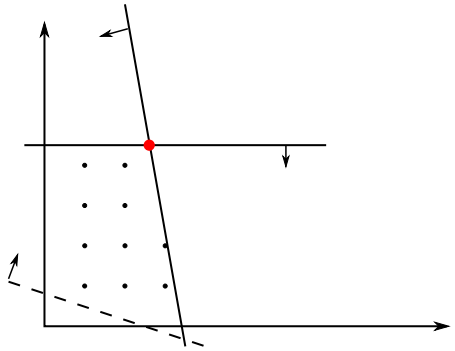
Bei den anderen Modellen werden nur lineare Nebenbedingungen hinzugefügt, das heißt, wir haben auf jeden Fall ein lineares Problem. Der Lösungsraum ist ein konvexes Polyeder, welches unter anderem alle zulässigen Lösungen des ganzzahligen Problems enthält. Dieses lineare Optimierungsproblem kann man nun zum Beispiel mit dem Simplex-Algorithmus lösen. So erhält man eine optimale Lösung für die LP-Relaxation. Ist diese Lösung ganzzahlig, hat man auch eine Lösung für das ursprüngliche Problem gefunden. Ist sie nicht ganzzahlig, muss ein Weg gefunden werden, anhand dieser Lösung der LP-Relaxation zu einer zulässigen Lösung unseres Modells zu gelangen. Dies kann man zum Beispiel mit dem Schnittebenenverfahren erreichen.

2.2 Das Schnittebenenverfahren

Definition:
Sei $P \subseteq \mathbb{R}^n$ das Polyeder der LP-Relaxation zu einem diskreten linearen Modell und sei Z die Menge der zulässigen ganzzahligen Lösungen. Außerdem sei $x^* \in \mathbb{R}^n$ die nicht ganzzahlige gefundene Optimallösung für P. Eine ***Schnittebene*** *ist eine Ebene $s^T x = b$, die den Punkt x^* von allen zulässigen ganzzahligen Lösungen trennt. Das heißt, es gilt bei einem Maximierungsproblem: $s^T x^* > \max\{s^T z | z \in conv(Z)\}$ (bei einem Minimierungsproblem analog mit „<“ und „min“).*
Das Problem, eine solche Ebene zu finden, nennt man ***Separierungsproblem****. [7]*

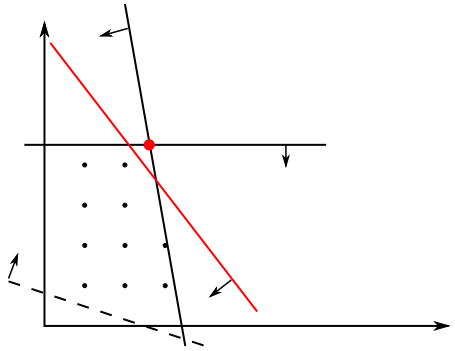
Beim Schnittebenenverfahren wird mit Hilfe der LP-Relaxation eine optimale Lösung x_{opt} für das ursprüngliche System gefunden. Dies funktioniert folgendermaßen:

Schritt 1:
Man berechnet z.B. mit dem Simplex-Algorithmus eine optimale Lösung x^* für das Polyeder P der LP-Relaxation, also das ursprüngliche Problem ohne die Ganzzahligkeitsbedingung.



Schritt 2:
Nun muss man überprüfen, ob die so erhaltene Lösung ganzzahlig ist oder nicht. Ist sie ganzzahlig, hat man eine Lösung für das ursprüngliche System gefunden und ist fertig. Ansonsten hat man bei einem Maximierungsproblem eine obere Schranke ($x_{opt} < x^*$), bei einem Minimierungsproblem eine untere Schranke ($x_{opt} > x^*$) für den Wert der Optimallösung des Ursprungssystems gefunden, denn der Lösungsraum ist eine Teilmenge des konvexen Polyeders der LP-Relaxation.

Schritt 3:
Ist die Optimallösung nicht ganzzahlig, muss man eine Schnittebene finden, die man als Ungleichung dem Programm der LP-Relaxation hinzufügen kann, sodass die gefundene optimale Lösung nicht mehr zu dem neuen zulässigen Bereich gehört. Man muss also eine Ebene finden, die die gefundene Lösung vom neuen Lösungsraum separiert und dabei keine zulässige ganzzahlige Lösung „abschneidet“. Sei $s^T x = b$ die gefundene Schnittebene.



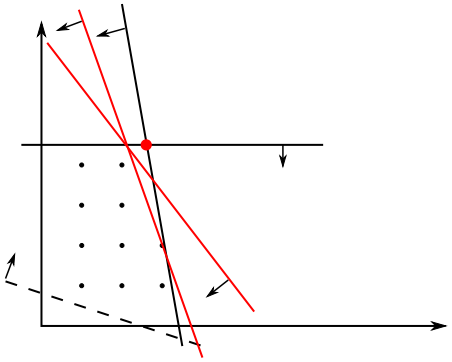
Schritt 4:
Man fügt die gefundene Schnittebene als Nebenbedingung zum linearen Programm der LP-Relaxation hinzu. Das neue Polyeder eines Maximierungsproblems wird definiert durch: $P' := P \cap s^T x \leq b$ (beim Minimierungssystem analog mit „$\geq$“). Die Lösungsmenge des linearen Problems wurde auf diese Art verkleinert, sodass sie meistens ein anderes Optimum besitzt. Man wiederholt die Schritte 1 bis 4 solange, bis die Lösung der LP-Relaxation ganzzahlig ist. Diese Lösung ist eine optimale Lösung des ursprünglichen ganzzahligen Problems, es gilt also: $x_{opt} = x^*$.

Das Schnittebenenverfahren entwickelt also mit Hilfe der zuletzt gefundenen Lösung ein neues verbessertes lineares Programm, indem es eine weitere Nebenbedingung hinzufügt. Die Lösungen dieser verbesserten Programme konvergieren gegen die gesuchte Lösung des ganzzahligen Problems [5]. Es ist allerdings im Voraus nicht bekannt, wieviele Schritte maximal benötigt werden. In jeder Iteration kann aber die obere bzw. untere Schranke für die Lösung des ganzzahligen Problems verbessert werden, da die neue Lösungsmenge immer eine Teilmenge (wieder ein konvexes Polyeder) des vorherigen ist. So kann der Zielfunktionswert beim Maximierungsproblem nur kleiner werden oder er bleibt gleich. Analog gilt beim Minimierungsproblem, dass der Zielfunktionswert der Teilmenge nur größer werden oder gleich bleiben kann.

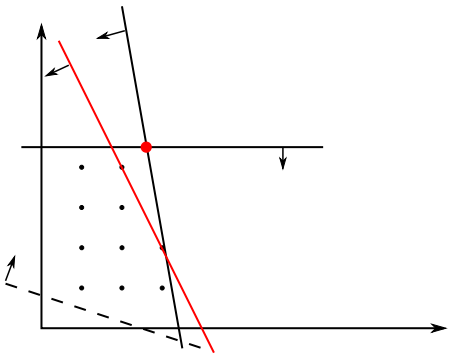
2.3 Wahl der Schnittebenen

Die Wahl der Schnittebenen beeinflusst die Geschwindigkeit, mit der man zu einer Optimallösung des ganzzahligen Problems gelangen kann. Daher ist für den Benutzer interessant, möglichst gute Schnittebenen zu finden, um den Lösungsraum schnell zu verkleinern.

Beispiel 5:
Es gibt viele Möglichkeiten, eine Schnittebene so zu wählen, dass sie das Optimum der LP-Relaxation von der neuen Lösungsmenge separiert.



Da man allerdings die Anzahl der Durchläufe des Schnittebenenverfahrens verringern möchte, ist eine solche Schnittebene besser als die oben gezeigten, da sie einen möglichst großen Anteil der Lösungsmenge abschneidet:



Man benötigt also ein Kriterium, um sinnvolle Schnittebenen zu finden.

Definition:
Eine nichttriviale Seitenfläche F eines konvexen Polyeders P heißt ***Facette*** *von P, falls F in keiner anderen echten Seitenfläche von P enthalten ist.*
Ist $P \subset \mathbb{R}^n$*, so gilt:* $\dim(F) = \dim(P) - 1$. *[6]*

Eine gute Schnittebene schneidet viel „unnötigen“ Lösungsraum ab. Außerdem ist es sinnvoll, nur solche Ebenen zu wählen, die nicht durch das Einfügen einer anderen Schnittebene redundant werden. Denn sonst hätte man diese neue Schnittebene sofort wählen können und sich so eine Iteration gespart.

Man darf den Lösungsraum der LP-Relaxation nur so weit verkleinern, dass auch wirklich noch alle zulässigen Punkte in ihr liegen. Das heißt, die konvexe Hülle conv(P) ist immer eine Teilmenge dieses Lösungsraums. Es dürfen also keine Schnittebenen hinzugefügt werden, die einen Teil der konvexen Hülle „abschneiden“.

Die Optimallösung liegt immer auf dem Rand der konvexen Hülle der zulässigen ganzzahligen Lösungen. Deshalb ist es sinnvoll, den Lösungsraum der LP-Relaxation in bestimmten Bereichen möglichst schnell auf diese konvexe Hülle zu verkleinern. Dafür wählt man, um möglichst viel vom Lösungsraum der LP-Relaxation abzuschneiden, ohne dass die Schnittebene später redundant wird, am besten die Facetten der konvexen Hülle der zulässigen ganzzahligen Lösungen. Mit dieser Wahl der Schnittebenen kann man schneller zu einer optimalen ganzzahligen Lösung gelangen.

3 Facetten von CP

Im folgenden wird mit der Modellierung des Color Order Models aus 1.3.1 gearbeitet. Die konvexe Hülle der ganzzahligen Lösungen dieser Modellierung ist $\mathscr{CP}$.

Da für eine Facette F eines Polyeders P gilt $\dim(F) = \dim(P) - 1$, muss man zuerst die Dimension der konvexen Hülle aller ganzzahligen zulässigen Lösungen berechnen, bevor man ihre Facetten bestimmen kann.

3.1 Dimension der konvexen Hülle der Lösungen

Definition:
*Die Vektoren $x_1, ...x_k$ heißen **affin unabhängig**, falls aus $\sum_{i=1}^{k} \lambda_i x_i = 0$ und $\sum_{i=1}^{k} \lambda_i = 0$ folgt: $\lambda_1 = ... = \lambda_k = 0$ [9].*

Geht man davon aus, dass der Graph keinen Knoten enthält, der adjazent zu allen anderen ist, also keine sogenannten universalen Knoten besitzt, gilt: $\chi(G) < |V|$. Enthält der Graph außerdem keine isolierten Knoten, die durch keine Kante mit einem anderen Knoten verbunden sind, gilt: $2 \leq \chi(G) < |V|$, da jeder Knoten mindestens einen Nachbar hat. In diesem Fall gilt:

Satz 1:
Die Dimension von $\mathscr{CP}$ ist $n^2 - \chi(G) - 1$

Beweisidee:
Teil 1: *zu zeigen:* $\dim \mathscr{CP} \geq n^2 - \chi(G) - 1$

Existiert nur eine Lösung, so ist die Lösungsmenge ein Punkt und hat die Dimension 0. Gibt es dagegen 2 affin unabhängige Lösungen, so ist der Raum, in dem die beiden liegen, eine Gerade und hat die Dimension 1. Um die Aussage zu beweisen, muss man also zeigen, dass es $n^2 - \chi(G)$ affin unabhängige Lösungen gibt.

Man betrachtet einen Graph mit n Knoten $v_1, .., v_n$ und einer Farbmenge S mit n Farben. Da es keine universalen Knoten gibt, kann man o.B.d.A annehmen, dass v_{n-1} und v_n nicht adjazent sind.

Eine erste Färbung kann man erzeugen, indem man einfach jedem Knoten eine Farbe zuordnet (v_1 hat die Farbe 1 usw.)

Farbe 1	Farbe 2	Farbe 3	...	Farbe i	...	Farbe $n-1$	Farbe n
v_1	v_2	v_3	...	v_i	...	v_{n-1}	v_n

Für $i = 1$ kann man nun aus dieser Lösung weitere Lösungen durch das Vertauschen einzelner Knoten erzeugen. Für $j = 1, .., n-1$, $i \neq j$ kann man folgendermaßen tauschen:

- Knoten v_n wird der Farbe i zugeordnet
- Knoten v_j der Farbe n
- Knoten v_i der Farbe j

Außerdem kann man den Knoten v_i der Farbe n zuordnen und den Knoten v_n der Farbe i.

Damit ergeben sich folgende $n-1$ Lösungen:

Farbe 1	Farbe 2	Farbe 3	...	Farbe i	...	Farbe $n-1$	Farbe n
v_n	v_1	v_3	...	v_i	...	v_{n-1}	v_2
v_n	v_2	v_1	...	v_i	...	v_{n-1}	v_3
...	...	...	...	...	...	...	...
v_n	v_2	v_3	...	v_1	...	v_{n-1}	v_i
...	...	...	...	...	...	...	...
v_n	v_2	v_3	...	v_i	...	v_1	v_{n-1}
v_n	v_2	v_3	...	v_i	...	v_{n-1}	v_1

Nach dieser Vorgehensweise kann man außerdem $n-1$ Färbungen für jedes $i \in \{2,..n-2\}$ erzeugen. Es ergeben sich also für $i = 1,...n-2$ genau $(n-2)\cdot(n-1)$ Färbungen.

Für $i = n-1$ und $j = 1,...,n-1$ lassen sich durch folgende Vertauschungsvorschrift neue Lösungen aus der ersten Lösung erzielen:

- ordne den Knoten v_n der Farbe $n-1$ zu
- Knoten v_j der Farbe n
- Knoten v_{n-1} der Farbe j

Damit ergeben sich folgende $n-1$ Färbungen:

Farbe 1	Farbe 2	Farbe 3	...	Farbe i	...	Farbe $n-1$	Farbe n
v_{n-1}	v_2	v_3	...	v_i	...	v_n	v_1
v_1	v_{n-1}	v_3	...	v_i	...	v_n	v_2
...	...	...	...	...	...	...	...
v_1	v_2	v_3	...	v_{n-1}	...	v_n	v_i
...	...	...	...	...	...	...	...
v_1	v_2	v_3	...	v_i	...	v_n	v_{n-1}

Weitere $n-1$ Lösungen lassen sich dadurch erzeugen, dass den Knoten v_{n-1} und v_n immer eine gemeinsame Farbe zugeordnet wird. Dies ist erlaubt, da die beiden Knoten laut Annahme nicht adjazent sind. Es ergeben sich folgende Lösungen:

Farbe 1	Farbe 2	Farbe 3	...	Farbe i	...	Farbe $n-1$	Farbe n
v_{n-1}, v_n	v_2	v_3	...	v_i	...	v_1	-
v_1	v_{n-1}, v_n	v_3	...	v_i	...	v_2	-
...	...	...	...	...	...	...	...
v_1	v_2	v_3	...	v_{n-1}, v_n	...	v_i	-
...	...	...	...	...	...	...	...
v_1	v_2	v_3	...	v_i	...	v_{n-1}, v_n	-

Wählt man nun noch je eine j-Färbung für $j = \chi(G),...,n-2$ ergibt sich für die gesamte Anzahl der erzeugten Lösungen: $1+(n-1)(n-2)+(n-1)+(n-1)+(n-2)-(\chi(G)-1) = n^2-\chi(G)$

Kann man nun noch zeigen, dass diese $n^2-\chi(G)$ Lösungen affin unabhängig sind, gilt: $\dim \mathscr{C}\mathscr{P} \geq n^2-\chi(G)-1$

Teil 2: *zu zeigen:* $\dim \mathscr{CP} \leq n^2 - \chi(G) - 1$

Satz 2:
Sei A eine m×n-Matrix und Ax = b ein lineares Gleichungssystem, das mindestens eine Lösung besitzt. Dann ist die Dimension des Lösungsraumes: n−rang(A) [8].

Man betrachtet das Gleichungssystem GS:

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i \in V$$
$$w_j = 1 \quad 1 \leq j \leq \chi(G)$$
$$\sum_{i=1}^{n} x_{in} = w_n$$

Das Gleichsungssystem GS hat $n + \chi(G) + 1$ linear unabhängige Gleichungen, das heißt, es gilt für dieses System: $\text{rang}(A) = n + \chi(G) + 1$. Es gibt $n^2 + n$ Variablen, das heißt, falls eine Lösung existiert, hat die Lösungsmenge von GS die Dimension $n^2 + n - (n + \chi(G) + 1 = n^2 - \chi(G) - 1$.

Kann man nun zeigen, dass $\mathscr{CP}$ eine Teilmenge der Lösungsmenge von GS ist, so gilt: $\dim \mathscr{CP} \leq n^2 - \chi(G) - 1$

Teil 3:
Aus Teil 1 und Teil 2 folgt nun: $\dim \mathscr{CP} = n^2 - \chi(G) - 1$

Den vollständigen Beweis kann man bei [4] nachlesen. □

3.2 Facetten definierende Ungleichungen

Es gibt viele Bereiche, anhand derer man Ungleichungen finden kann, die Facetten der konvexen Hülle der Lösungen, also in diesem Fall von $\mathscr{CP}$, definieren.

3.2.1 Ungleichungen anhand unabhängiger Mengen

Definition:
Mit $\alpha(G)$ bezeichnet man die größte Mächtigkeit einer unabhängigen Menge im Graphen G. Dies ist die ***Stabilitätszahl****. [2]*

Da jede Knotenmenge, der eine Farbe zugeordnet wird, eine unabhängige Menge ist, muss deren Größe kleiner als $\alpha(G)$ sein. Da dies natürlich auch für jeden Teilgraph $G' = (V', E')$ mit der zugehörigen Stabilitätszahl dieses Graphen gilt, ist die folgende Ungleichung für jede Farbe j_0 erfüllt:

$$\sum_{v \in V'} x_{v j_0} \leq \alpha(G') w_{j_0}$$

Weist man der größten unabhängigen Menge in G' die Farbe j_0 zu, so werden $\alpha(G')$ viele der n Knoten von G' mit dieser Farbe gefärbt. Man benötigt also noch höchstens $n - \alpha(G') + 1$ Farben, um die restlichen Knoten zu färben. Jede $(n - \alpha(G') + 1)$-Färbung, die diese Farbwahl vorweist, erfüllt die Ungleichung mit Gleichheit. Das heißt, es handelt sich um eine Seitenfläche. Es muss sich allerdings nicht um eine Facette handeln.

Man kann diese Ungleichung noch verstärken, denn es gilt laut [4] für jede Lösung:

$$\sum_{j=n-\alpha(G')+1}^{n} \sum_{v \in V} x_{vj} \leq w_{n-\alpha(G')+1}$$

Aus diesen beiden Ungleichung ergibt sich:

$$\sum_{v \in V'} x_{vj_0} + \sum_{j=n-\alpha(G')+1}^{n} \sum_{v \in V} x_{vj} \leq \alpha(G') w_{j_0} + w_{n-\alpha(G')+1}$$

Wann diese Ungleichung eine Facette beschreibt, wird im folgenden Satz aufgezeigt:

Satz 3:
Sei $V' \subset V$, $G' = G(V')$ mit $\alpha(G') < \alpha(G)$, $j_0 \leq n - \alpha(G')$ und der gültigen Ungleichung

$$\sum_{v \in V'} x_{vj_0} + \sum_{j=n-\alpha(G')+1}^{n} \sum_{v \in V} x_{vj} \leq \alpha(G') w_{j_0} + w_{n-\alpha(G')+1}$$

Diese Ungleichung definiert eine Facette, falls gilt:

- *Es gibt eine unabhängige Menge der Größe $\alpha(G') + 1$ in $G(V' \cup \{v\})$ für alle $v \in V \setminus V'$*
- *Es existiert eine maximale unabhänige Menge I von G', so dass $V \setminus I$ keine Clique ist*
- *Es gibt eine $\chi(G)$-Färbung, die auf der Seitenfläche, die durch die Ungleichung erzeugt wird, liegt [4]*

Beweis: siehe [4]

3.2.2 Cliquen-Ungleichung

Definition:
*Ein **vollständiger Graph** ist ein Graph, in dem jeder Knoten mit jedem anderen durch eine Kante verbunden ist. Eine **Clique** in einem Graphen ist ein vollständiger Teilgraph. Eine **maximale Clique** ist eine Clique, die in keiner anderen (als Teilgraph) echt enthalten ist. [1]*

Zu einer beispielhaften Ungleichung, die eine Facette des Graphen definiert, kann man gelangen, indem man sich die Cliquen des Graphen anschaut.

Satz 4:
Sei K eine maximale Clique. Die Cliquen-Ungleichung

$$\sum_{v \in K} x_{vj_0} - w_{j_0} \leq 0$$

ist eine Facetten definierende Ungleichung von $\mathscr{CP}$ [4].

Beweis:
Die Ungleichung aus Satz 4 ist für jede Clique des Graphen erfüllt, denn höchstens ein Knoten der Clique kann mit der Farbe j_0 gefärbt werden. Wählt man $V' = K$, so gilt $\alpha(G') = 1$, da der betrachtete Teilgraph eine Clique ist. Aus der Cliquen-Ungleichung ergibt sich also: $\sum_{v \in V'} x_{vj_0} \leq \alpha(G') w_{j_0}$.

Außerdem gilt, da die Farbe n bei einem Graph mit n Knoten höchstens einmal benutzt wird:

$$\begin{aligned}
&\sum_{j=n-\alpha(G')+1}^{n} \sum_{v \in V} x_{vj} \\
&= \sum_{j=n}^{n} \sum_{v \in V} x_{vj} \\
&= \sum_{v \in V} x_{vn} \\
&\leq w_n \\
&= w_{n-\alpha(G')+1}
\end{aligned}$$

Damit ist die Cliquen-Ungleichung aus Satz 4 ein Spezialfall der Ungleichung aus Satz 3. Um zu zeigen, dass die Cliquen-Ungleichung eine Facette ist, muss man also beweisen, dass die drei Bedingungen aus Satz 3 gelten.

Bedingung 1:
zu zeigen: Es gibt eine unabhängige Menge der Größe 2 in $G(K \cup \{v\})$ für alle $v \in V \setminus K$

Da K eine maximale Clique ist, gibt es zu jedem Knoten $v \in V \setminus K$ einen Knoten $v' \in K$, der nicht adjazent zu v ist. Denn sonst würde dieser Knoten v zur Clique gehören. Das geht nicht, da es eine maximale Clique ist. Betrachtet man nun den Graph $G(K \cup \{v\})$, gibt es eine unabhängige Menge der Größe 2, zu der genau die beiden Knoten v und v' gehören. Da man diese Menge für jeden Knoten $v \in V$ finden kann, ist die erste Bedingung erfüllt.

Bedingung 2:
zu zeigen: Es existiert eine maximale unabhänige Menge I von G', so dass $V \setminus I$ keine Clique ist

Da K eine maximale Clique ist, hat jede unabhängige Menge von K die Größe 1 und man kann zwei Knoten $v_1, v_2 \in K$ wählen, sodass v_1 einen nicht adjazenten Knoten in $V \setminus K$ besitzt. Wählt man also als maximale unabhängige Menge den Knoten v_2, so ist $V \setminus \{v_2\}$ keine Clique. Damit ist Bedingung 2 erfüllt.

Bedingung 3:
zu zeigen: Es gibt eine $\chi(G)$-Färbung, die auf der Seitenfläche, die durch die Ungleichung erzeugt wird, liegt

Um dies zu zeigen, muss man beweisen, dass es eine $\chi(G)$-Färbung gibt, die die Cliquen-Ungleichung mit Gleichheit erfüllt. Jede Färbung, die einem Knoten in der Clique K die Farbe j_0 zuweist erfüllt die Ungleichung mit Gleichheit, denn wird die Farbe j_0 benutzt, gilt $w_{j_0} = 1$. Da K eine Clique ist, kann höchstens ein Knoten die Farbe j_0 haben, das heißt, es gilt: $\sum_{v \in K} x_{vj_0} - w_{j_0} = 1 - 1 = 0$. Da es eine $\chi(G)$-Färbung gibt, die die Farbe j_0 einem Knoten in der Clique K zuweist, ist auch die dritte Bedingung erfüllt. □

3.2.3 Weitere Facetten

Außer der Betrachtung von Cliquen gibt es noch weitere Bereiche, durch die man zu Facetten definierenden Ungleichungen gelangen kann. Einige Ungleichungen aus der Modellierung des Färbungsproblems sind schon Facetten. Man muss also zeigen, dass sie eine Seitenfläche der Dimension $n^2 - \chi(G) - 1$ beschreiben. Weitere Facetten kann man finden, wenn man sich zum Beispiel Kreise ohne Sehnen und ihre Komplenmente oder Wege des Graphen anschaut und die gefundenden Gleichungen auf die Situation in Satz 3 zurückführt. Es gibt noch viele weitere Bereiche, bei deren Betrachtung man Facetten von $\mathscr{CP}$ finden kann. Man kann zum Beispiel außerdem Mehrfarben-Cliquen-Ungleichungen, Mehrfarben-Wege-Ungleichungen und Block-Farben-Ungleichungen aufstellen, die Facetten von $\mathscr{CP}$ definieren. Diese gefundenen Facetten kann man nun als Schnittebenen benutzen, um zu einer optimalen Färbung zu gelangen.

4 Das Schnittebenenverfahren von I. Méndez-Díaz und P. Zabala

4.1 LP-Relaxation

Der erste Schritt zu einem guten Schnittebenenverfahren ist es, eine gute Relaxation für das ganzzahlige Problem zu finden.

Die normale LP-Relaxation aus Kapitel 2.1 lässt nur die Ganzzahligkeitsbedingungen weg. So hat man aber immer noch viele Nebenbedingungen der Art:

$$x_{uj} + x_{vj} \leq w_j \qquad \forall\, (i,k) \in E,\ 1 \leq j \leq n.$$

Die Nebenbedingungen legen fest, dass zwei benachbarte Knoten nie die gleiche Farbe haben. Bei kleinen Graphen ist das kein Problem, man kann sie mit einem Schnittebenenverfahren, das eine normale LP-Relaxation als Grundlage hat, färben. Bei großen und dichten Graphen gibt es allerdings zu viele solcher Nebenbedingungen, was das Lösen erschwert, da der LP-Solver zu lange braucht und so der gesamte Algorithmus zu langsam ist. Man muss also eine verbesserte Relaxation als Grundlage für das Schnittebenenverfahren finden, wenn man den Algorithmus auf Graphen aller Art anwenden will. Es gibt allerdings keinen vorgeschriebenen Weg, der zu einer besseren Relaxation führt. Man muss also verschiedene Ansätze durchprobieren, bis man zu einem guten Ergebnis kommt. Ob ein Ergebnis, also die gefundene Relaxation, gut ist, kann man an bestimmten Kriterien erkennen, die für den Anwender wichtig sind. Ein mögliches Kriterium ist zum Beispiel die Zeit, die der LP-Solver benötigt, um die Relaxation zu lösen. Ein anderes Kriterium ist die Entwicklung der unteren Schranke für die chromatische Zahl $\chi(G)$ des Graphen, wenn man eine bestimmte Relaxation als Grundlage wählt, denn man möchte, wenn es geht, eine optimale Lösung mit möglichst wenigen Durchgängen finden.

Méndez-Díaz und Zabala haben zum Beispiel zuerst versucht, die oben genannten Nebenbedingen einfach wegzulassen oder durch Ungleichungen zu ersetzen, die man durch die Betrachtung von Cliquen erhält. Beide Möglichkeiten wurden verworfen, da sie zu keiner wirklichen Verbesserung der Relaxation und des darauf aufbauenden Schnittebenenverfahrens geführt haben. Dies wurde durch Ausprobieren des Algorithmus auf viele Graphen verschiedener Größe und Dichte herausgearbeitet. Auf diese Weise sind sie auch zu einer für sie guten Lösung gelangt, indem sie neben der LP-Relaxation eine weitere Relaxation mit Hilfe von Nachbarschaftsungleichungen vorgenommen haben.

Sei $v \in V$ und sei $N(v)$ die Nachbarschaft von v. $N(v)$ umfasst also alle Knoten, die adjazent zu v sind. Es gilt: $\delta(v) = |N(v)|$, $\delta(v)$ gibt also an, wieviele Nachbarn der Knoten v hat.

Das Modell des Färbungsproblems enthält für alle Knoten v und alle Farben j die Nebenbedingungen:

$$x_{vj} + x_{kj} \leq w_j \qquad \forall\, k \in N(v)$$

Kombiniert man diese Nebenbedingungen für einen Knoten v, so erhält man die Ungleichung:

$$\sum_{k \in N(v)} x_{kj} + \delta(v) x_{vj} \leq \delta(v) w_j$$

Jede Lösung der Modellierung erfüllt diese Ungleichung, da immer einer dieser drei Fälle eintritt:

- Knoten v hat die Farbe $j \Rightarrow$ kein Knoten aus der Nachbarschaft kann die Farbe j haben $\Rightarrow 0 + \delta(v) = \delta(v)$
- Knoten v hat nicht die Farbe j, aber mindestens ein Knoten der Nachbarschaft wird mit Farbe j gefärbt $\Rightarrow \sum_{k \in N(v)} x_{kj} + 0 \leq \delta(v)$
- Keiner der betrachteten Knoten wird mit der Farbe j gefärbt $\Rightarrow 0 = 0$ oder $0 \leq \delta(v)$, je nachdem ob die Farbe j zum Färben anderer Knoten genutzt wird.

Diese gefundene Ungleichung kann noch verstärkt werden, indem man statt $\delta(v)$ die Zahl r wählt, wobei r die Größe der größten unabhängigen Menge des Graphen ist. So erhält man die Ungleichung:

$$\sum_{k \in N(v)} x_{kj} + r \cdot x_{vj} \leq r \cdot w_j$$

Jede Lösung der Modellierung erfüllt auch diese Ungleichung, da $\sum_{k \in N(v)} x_{kj}$ höchstens den Wert r annehmen kann.

Da eine Färbung aber gerade eine Aufteilung in unabhängige Mengen ist, wobei jeder unabhängigen Menge eine Farbe zugeordnet wird, kann man die Zahl r vor der Lösung des Problems nicht bestimmen. Stattdessen kann man N(v) mit einem Greedy-Algorithmus in Cliquen zerlegen. Der Greedy-Algorithmus fügt jeden betrachteten Knoten $v_i \in N(v)$ zur ersten Clique hinzu, in der nur Knoten enthalten sind, die adjazent zu v_i sind. Gibt es keine solche Clique, in die der Knoten v_i passt, wird eine neue Clique hinzugefügt, die bis jetzt nur den Knoten v_i beinhaltet. Diese Aufteilung gibt meistens nicht die wirkliche Anzahl der Cliquen wieder, sondern nur eine obere Schranke. Es kann sein, dass man einen Knoten zu einer Clique hinzufügt, der dann das Hinzufügen anderer Knoten behindert und man so mehr Cliquen als nötig als Lösung erhält. Anstatt der Zahl r wählt man nun die Anzahl der gefundenen Cliquen. Dies ist sinnvoll, denn die Farbe j kann in jeder Clique höchstens einmal vertreten sein.

Bei der normalen LP-Relaxation gab es mn Ungleichungen der Art $x_{uj} + x_{vj} \leq w_j$. Ersetzt man diese durch die gefundenen Nachbarschaftsungleichungen, wird das Polytop weiter relaxiert. Diese neue LP-Relaxation hat nicht so viele Nebenbedingungen wie die ursprüngliche, was das Lösen von großen und dichten Graphen in annehmbarer Zeit ermöglicht. Laut den Erfahrungen von Méndes-Díaz und Zabala kann mit dieser Relaxation ein gutes Gleichgewicht zwischen der benötigten Zeit, dem benötigten Speicherplatz und der Verbesserung der unteren Schranke geschaffen werden. Dies waren die für sie wichtigen Kriterien, um eine Verbesserung der ursprünglichen LP-Relaxation zu finden. Es kann also sein, dass es Anwender gibt, für die diese Relaxation nicht optimal ist, da für sie hauptsächlich eines dieser 3 Kriterien wichtig ist. Sie würden eine Relaxation bevorzugen, die auf diesem Gebiet Verbesserungen bringt und sind dazu auch bereit Verschlechterungen auf anderen Gebieten hinzunehmen. Möchte man aber eine Relaxation, bei der diese 3 Kriterien ungefähr gleich wichtig sind, was normalerweise der Fall ist, so ist die gefundene Relaxation geeignet.

4.2 Obere und untere Schranken

Obere und untere Schranken sind wichtig, da man damit die Anzahl der Variablen einschränken kann und so das Problem nicht zu groß wird. Hat man eine untere Schranke $\check{\chi}$ gefunden, so bedeutet dies, dass gilt $\check{\chi} \leq \chi(G)$. Man weiß also, dass die Farben 1 bis $\check{\chi}$ auf jeden Fall benutzt werden, was die Lösungsmenge verkleinert.

Eine obere Schranke ist wichtig, um die Anzahl der Farben einzuschränken, die zum Färben des Graphen genutzt werden können. Hat man zum Beispiel einen Graph mit 1000 Knoten, liegt die obere Schranke $\hat{\chi}$ für $\chi(G)$ zunächst bei 1000. Würde man diese nicht verbessern, müsste man den Graphenfärbungsalgorithmus mit einer Farbmenge durchführen, für die gilt: $|S| = 1000$. Es müssten also 1000 Farben in der Farbmenge sein. So erhält man 1001000 Variablen und viele Nebenbedingungen. Findet man nun aber heraus, dass $\chi(G) \leq 10 = \hat{\chi}$ gilt, kann man nun eine Farbmenge mit $|S| = 10$

betrachten. Dadurch sinkt die Zahl der Variablen von 1001000 auf 10010 und es fallen auch einige Nebenbedingungen weg.

Eine untere Schranke $\check{\chi}$ kann man mit einem Greedy-Algorithmus bestimmen. Dieser könnte zum Beispiel so aussehen: Man beginnt mit einer Clique C, zu der nur der Knoten 1 gehört. Dann fügt man zur Menge C der Reihe nach die Knoten hinzu, die zu allen bisher enthaltenen Knoten adjazent sind. So findet man eine Clique des Graphen und damit eine untere Schranke. Diese Clique ist in den meisten Fällen kleiner als die wirklich größte Clique des Graphen, da man wahrscheinlich einen Knoten zu C hinzufügt, der normalerweise nicht in der größten Clique enthalten ist, aber zum Einfügezeitpunkt mit allen bis dahin in der Menge C enthaltenen Knoten adjazent ist. Will man eine genauere untere Schranke erreichen, muss dieser Greedy-Algorithmus also verbessert werden. Eine Möglichkeit ist zum Beispiel, die Knoten nicht in der Reihenfolge der Numerierung durchzuprobieren, sondern zuerst die Knoten zu betrachten, die möglichst viele Nachbarn haben.

Hat man eine Clique der Größe $\check{\chi}$ gefunden, kann man das Problem vereinfachen, indem man jedem Knoten dieser gefundenen Clique eine der ersten $\check{\chi}$ Farben zuweist und damit die Werte einiger Variablen fixiert. Denn man weiß nun, dass $w_1, ..., w_{\check{\chi}}$ den Wert 1 annehmen müssen und auch der Wert der Variabeln x_{cj} mit $c \in C$ und $1 \leq j \leq \check{\chi}$ ist fest bestimmt. Dadurch verkleinert sich das Problem und kann schneller gelöst werden.

Eine obere Schranke kann man mit dem DSATUR Algorithmus finden, der dichtere Bereiche des Graphen zuerst färbt und so zu einer möglichst guten oberen Schranke führt. Am Anfang werden alle Knoten nach dem Grad geordnet, also nach der Anzahl der adjazenten Knoten, beginnend bei dem Knoten mit den meisten Nachbarn. Jedem Knoten wird die Zahl sat der Farben beigeordnet, die bisher benutzt wurden, um benachbarte Knoten einzufärben. Diese Zahl gibt also an, wieviele Farben aus der Farbmenge der bisher benutzten Farben man nicht mehr zum Färben des betrachteten Knoten benutzen kann. Am Anfang wird jedem Knoten die Zahl 0 zugeordnet, da noch kein Knoten gefärbt wurde. Zuerst wird ein Knoten mit maximalem Grad mit der Farbe 1 gefärbt. Danach wird für alle übrigen Knoten die beigeordnete Zahl sat neu berechnet. Im weiteren Verlauf wird immer der Knoten ausgewählt, dem die größte Zahl sat zugeordnet wurde. Gibt es mehrere Knoten, die so zur Auswahl stehen, wird einer der Knoten mit maximalem Grad ausgewählt. Dieser Knoten wird nun mit einer möglichst kleinen Farbe gefärbt. Dies wird solange wiederholt, bis alle Knoten gefärbt sind. Die Anzahl der Farben, die für diese Färbung benutzt wurden, ist eine obere Schranke für die chromatische Zahl $\chi(G)$.

Hat man eine solche obere Schranke $\hat{\chi}$ gefunden, kann man alle Variablen w_j und x_{vj} mit $j > \hat{\chi}$ entfernen. Durch die geringere Anzahl an Variablen und Nebenbedingungen erhält man ein kleineres Modell und kann so schneller zu einer Lösung gelangen.

4.3 Finden von Schnittebenen

Das wichtigste beim Schnittebenenverfahren ist die Suche nach Schnittebenen, die die gefundene nicht ganzzahlige Lösung der Relaxation von der konvexen Menge der Lösungen trennt. In Teil 3 wurden Facetten definierende Ungleichungen angesprochen, die als Schnittebenen dienen können. Diese sind allerdings nur allgemein definiert und noch nicht auf den zu färbenden Graphen angepasst. Zum Beispiel werden bei der Cliquen-Ungleichung alle Knoten einer Clique betrachtet. Wenn der Algorithmus gestartet wird, ist aber noch nicht bekannt, welche Knoten des Graphen eine Clique bilden. Man muss also die Facetten definierenden Ungleichungen erst aufstellen beziehungsweise finden. Es ist wichtig, die Schnittebenen auf eine möglichst effiziente Art zu finden, damit der gesamte Algorithmus schneller und mit weniger Speicheraufwand zu einer optimalen Lösung gelangen kann. Méndez-Díaz und Zabala haben daher für die verschiedenen Arten von Facetten definierenden Ungleichungen unterschiedliche Wege gefunden, diese aufzustellen.

Um die Cliquen-Ungleichung aufzustellen, könnte man zum Beispiel, bevor man den Algorithmus startet, eine Liste mit maximalen Cliquen des Graphen aufstellen. Anhand dieser Liste könnte man dann die zugehörigen Ungleichungen aufstellen. Dies ist laut [4] allerdings nicht geeignet, da es zwar eine schnelle Möglichkeit ist, die Ungleichungen aufzustellen, man so aber nicht genügend Schnittebenen erhält. Außerdem kann man so nicht den Wert der Lösung der LP-Relaxation (x^*, w^*) ausnutzen.

Um Cliquen-Ungleichungen zu finden, verwenden sie stattdessen für jede Farbe j_0 die folgende Greedy-Heuristik:

- Man erstellt eine Liste mit den Werten $\{x^*_{ij_0} : i \in V, x^*_{ij_0} < 1\}$ der aktuellen Lösung (x^*, w^*) der LP-Relaxation, beginnend mit dem größten Wert.
- Ist $x^*_{ij_0}$ nicht ganzzahlig, initialisiert man eine Clique, die nur den Knoten i enthält.
- Dieser Clique fügt man nun, in der Reihenfolge der Liste, weitere Knoten hinzu, wenn diese zu allen Knoten der bisher gefundenen Clique adjazent sind.
- Es werden so viele Knoten hinzugefügt, bis eine bestimmte Zahl erreicht wurde, die man beim Starten des Algorithmus bestimmen muss oder man am Ende der Liste angelangt ist.

Mit der so gefundenen Clique kann man nun die Cliquen-Ungleichung aufstellen. Diese kann aber nicht immer auch als Schnittebene benutzt werden, da bisher nur eine Facette von $\mathscr{CP}$, also der konvexen Hülle der Lösungen, gefunden wurde. Es kann sein, dass die aktuelle Lösung (x^*, w^*) der LP-Relaxation die Ungleichung erfüllt. Das heißt, das Hinzufügen dieser Ungleichung würde das Problem nur vergrößern, da nun eine zusätzliche Nebenbedingung existiert. Sie würde aber nicht dazu beitragen, dass durch das erneute Lösen der LP-Relaxation eine verbesserte Annäherung an eine optimale ganzzahlige Lösung erhalten wird. Man muss also zuerst prüfen, ob die durch die Greedy-Heuristik gefundene Cliquen-Ungleichung den Punkt (x^*, w^*) von $\mathscr{CP}$ separiert. Ist dies der Fall, kann die gefundene Ungleichung als Schnittebene verwendet werden.

Mit diesen gefundenen Cliquen kann man auch die Mehrfarben-Cliquen-Ungleichungen aufstellen und prüfen, ob sie als Schnittebenen verwendet werden können. Für die übrigen Arten Facetten definierender Ungleichungen muss man andere Wege finden, Gleichungen aufzustellen und Schnittebenen zu erzeugen. Bei den Block-Farben-Ungleichungen geht man alle möglichen Ungleichungen per Brute-Force durch. Mehrfarben-Pfad-Ungleichungen werden aufgestellt, indem man Pfade anhand von bestimmten Gewichten generiert, die man den Kanten zuweist. Für die Ungleichungen, zu denen man durch das Betrachten sehnenloser Kreise gelangen kann, wird ein bestimmter Algorithmus verwendet.

Es gibt also viele verschiedene Arten, wie man Facetten definierende Ungleichungen aufstellen und überprüfen kann. Will man einen bestehenden Algorithmus verbessern, so ist eine Möglichkeit zum Beispiel das Finden von neuen und besseren Methoden zum Aufstellen von Schnittebenen. Die hier genannten Methoden waren aus der Sicht von Méndez-Díaz und Zabala am geeignetesten.

4.4 Verlauf des Schnittebenenverfahrens

Im Gegensatz zur Erklärung des Schnittebenenverfahrens in Teil 2.2, wird bei dem Algorithmus von Méndez-Díaz und Zabala nicht nur eine Schnittebene pro Iteration hinzugefügt, sondern eine Menge von Ungleichungen. Dies ist ein Vorteil, da der Algorithmus für große Graphen sehr viel Zeit benötigen würde, wenn in jeder Iteration nur eine Ungleichung hinzugefügt werden würde. Eine Schwierigkeit, die dabei auftritt, ist die zunehmende Größe der zu lösenden LP-Relaxation. Man kann nicht einfach nur bei jeder Iteration alle gefundenen Schnittebenen hinzufügen, da das Problem sonst mit der Zeit zu groß wird und der LP-Solver zu lange braucht, um die Relaxation zu lösen. Daher müssen die Ungleichungen, die bei einer früheren Iteration zum Problem hinzugefügt wurden, jetzt aber nicht mehr wichtig sind, gelöscht werden. Das heißt, sie werden nicht mehr als Nebenbedingungen des Problems verwendet.

Da es aber sein kann, dass diese Ungleichungen zu einem späteren Zeitpunkt noch einmal gebraucht werden, ist es nicht sinnvoll, sie vollständig zu „löschen“, da man sie sonst neu finden muss. Man sammelt diese Ungleichungen also in einer Datenbasis, damit man schnell wieder auf sie zugreifen kann. Neben diesen Ungleichungen werden dort auch solche gespeichert, die nie als Nebenbedingungen zum Problem hinzugefügt wurden, da sie zu dem Zeitpunkt, als sie gefunden wurden, die Lösung (x^*, w^*) nicht von $\mathscr{CP}$ separiert haben. Dies kann aber in einer späteren Iteration der Fall sein. Da nur Heuristiken verwendet werden, um die Ungleichungen zu finden, kann es sein, dass man eine Ungleichung, die man in der ersten Iteration aufgestellt hat, aber nicht benutzen konnte, so schnell nicht wieder finden und aufstellen kann. Deshalb werden auch diese Ungleichungen in der Datenbasis gespeichert.

Die Ungleichungen aus der Datenbasis können schnell überprüft werden, weshalb man am Anfang jeder Iteration, also nachdem die neue Relaxation gelöst wurde, zuerst diese Basis nach verletzten Ungleichungen durchsucht. Werden in der Datenbasis schon mehr als 200 Ungleichungen gefunden, die (x^*, w^*) von $\mathscr{CP}$ separieren, optimiert man das Problem, indem man diese Ungleichungen als Schnittebenen hinzufügt und eventuell unwichtige Nebenbedingungen entfernt.

5 Ergebnisse zum Schnittebenenverfahren von I. Méndez-Díaz und P. Zabala

Wie gut ein Schnittebenenverfahren funktioniert, hängt davon ab, wie gut die Wahl der Schnittebenen ist und wie schnell und mit welchem Aufwand man diese Ungleichungen finden kann. Facetten sind meist gute Schnittebenen, was aber nicht immer so sein muss. Um auszuwerten, welche Facetten besonders geeignet sind, um sie als Schnittebenen der Relaxation hinzuzufügen, kann man zum Beispiel folgende Bewertungskriterien betrachten:

- Wie entwickelt sich die untere Schranke?
- Welche Zeit wird benötigt?
- Wieviel Zeit wird zum Finden von Schnittebenen verwendet?
- Wieviel Prozent der gefundenen Ungleichungen können als Schnittebenen verwendet werden?

Um dies für verschiedene Kombinationen von Schnittebenen zu testen, muss man zuerst Graphen erzeugen, die man einfärben lässt. Méndez-Díaz und Zabala haben sich zufällige Graphen mit 125 Knoten und verschiedener Dichte erzeugen lassen, wobei bei geringer Dichte weniger als 30% der möglichen Kanten verwendet wurden, bei mittlerer Dichte zwischen 40% und 50% und bei hoher Dichte mehr als 70%. Für jede Dichte haben sie 8 Graphen erstellt und auf diesen jeweils 50 Iterationen des Schnittebenenverfahrens durchgeführt. Dabei hat sich herausgestellt, dass nach diesen Iterationen bei allen getesteten Kombinationen die gleiche untere Schranke gefunden wurde, außer wenn die Cliquen-Ungleichungen nicht als mögliche Schnittebenen verwendet wurden. War dies der Fall, konnte die untere Schranke nicht so sehr verbessert werden. Darum scheinen Cliquen-Ungleichungen wichtige Schnittebenen zu sein, die man auf jeden Fall verwenden sollte. Es wurden also folgende Kombinationen der möglichen Ungleichungen getestet:

C_1	Cliquen
C_2	Cliquen, Block-Farben, Mehrfarben-Wege
C_3	Cliquen, Block-Farben, Mehrfarben-Wege, Mehrfarben-Cliquen
C_4	Cliquen, Mehrfarben-Cliquen, sehnenlose Kreise
C_5	Cliquen, Block-Farben, Mehrfarben-Wege, Mehrfarben-Cliquen, s. Kreise

5.1 Wie entwickelt sich die untere Schranke und in welcher Zeit?

Die Entwicklung der unteren Schranke ist wichtig, da dies wiedergibt, wie und wie schnell sich die Lösung der LP-Relaxation einer optimalen ganzzahligen Lösung annähert. Während der 50 Iterationen pro Graph wird beobachtet, bei welcher Iteration die beste untere Schranke LB angenommen wird und wieviel Zeit dafür benötigt wurde. Außerdem wird für die 8 Graphen, die der gleichen Dichtekategorie angehören, berechnet, wieviel Prozent durchschnittlich am Anfang zwischen der oberen Schranke $\hat{\chi}$ und der unteren Schranke $\check{\chi}$ liegen und wieviel Prozent im Durchschnitt nach den 50 Iterationen zwischen $\hat{\chi}$ und LB liegen. Die Ergebnisse sind in den folgenden Tabellen dargestellt:

	geringe Dichte		mittlere Dicht		hohe Dichte	
	Zeit	Iteration	Zeit	Iteration	Zeit	Iteration
C_1	24	11	215	29	369	35
C_2	28	11	214	28	386	35
C_3	26	11	249	25	484	37
C_4	39	11	250	26	426	32
C_5	37	11	265	26	524	36

	geringe Dichte	mittlere Dichte	hohe Dichte
Unterschied $\hat{\chi}, \check{\chi}$	47%	52%	43%
Unterschien $\hat{\chi}, LB$	35%	38%	27%

Man kann also im Bezug auf das Finden der endgültigen unteren Schranke LB sagen, dass sie bei geringer Dichte schon am Anfang gefunden wird, bei mittlerer Dichte fast genau in der Mitte der 50 Iterationen und es bei hoher Dichte auch am Schluss noch Verbesserungen gibt. Diese untere Schranke wird für die jeweilige Dichte in etwa in der gleichen Runde gefunden, egal welche Ungleichungen man als Schnittebenen verwendet. Man kann also anhand von diesem Kriterium keine Ungleichung als besonders nützlich oder unbrauchbar bezeichnen. Mit zunehmender Dichte des Graphen fällt auch der durchschnittliche prozentuale Anstieg der unteren Schranke höher aus, das heißt, die untere Schranke wird deutlicher verbessert, wenn es sich um einen dichten Graphen handelt. Betrachtet man nun die Zeit, die zum Finden dieser unteren Schranke benötigt wurde, sieht man, dass der Algorithmus vor allem bei den dichteren Graphen länger braucht, wenn man die durch sehnenlose Kreise bestimmten Ungleichungen und Mehrfarben-Cliquen-Ungleichungen benutzt. Diese beiden Arten von Ungleichungen scheinen laut diesem Kriterium weniger geeignet zu sein.

5.2 Wieviel Zeit wird zum Finden von Schnittebenen verwendet?

Ein weiteres Bewertungskriterium ist die Zeit, die zum Separieren benötigt wird. Fällt diese sehr hoch aus, sollte man überlegen, andere Verfahren zum Aufstellen der Schnittebenen zu finden. Die folgende Tabelle zeigt für die verschiedenen benutzten Ungleichungen, welche Zeit durchschnittlich für die jeweils 8 Graphen einer Dichtekategorie verwendet wurde, und wieviel Zeit davon im Durchschnitt für das Finden von Schnittebenen, also für das Separieren, angefallen ist.

	geringe Dichte			mittlere Dicht			hohe Dichte		
	Gesamtzeit	Separieren	%	Gesamtzeit	Separieren	%	Gesamtzeit	Separieren	%
C_1	47,2	6,6	14	254,8	9,2	4	472,6	11	2
C_2	49,6	6,8	14	255	10,8	4	481	12,4	3
C_3	51,8	8	15	298,8	11,2	4	564,2	17,2	3
C_4	66,2	15,8	24	312	23,6	8	661,8	35,8	5
C_5	69,2	18	26	328,4	24,8	8	629,4	38	6

Man kann sehen, dass insgesamt nur ein geringer Teil der Zeit zum Finden von Schnittebenen verwendet wird. Es wird aber prozentual mehr Zeit benötigt, wenn man Mehrfarben-Cliquen-Ungleichungen und Ungleichungen, die anhand von sehnenlosen Kreisen gefunden wurden, als Schnittebenen verwendet. Auch nach diesem Kriterium scheinen diese beiden Ungleichungsarten nicht optimal zu sein.

5.3 Wieviel Prozent der gefundenen Ungleichungen können als Schnittebenen verwendet werden?

Wie schon angesprochen, kann nicht jede Ungleichung, die im Laufe der Lösungsfindung aufgestellt wurde, als Schnittebene verwendet werden. Es ist also interessant herauszufinden, wie hoch der Anteil der wirklich gebrauchten Ungleichungen gegenüber den insgesamt aufgestellten Ungleichungen aus den verschiedenen Bereichen ist. Ist der Anteil sehr gering, so kann es sinnlos sein, viel Zeitaufwand zu betreiben und Speicherplatz zu verwenden, um eine solche Gleichung aufzustellen.

Betrachtet man die einzelnen Arten von Ungleichungen, so kann man feststellen, dass das Suchen der Mehrfarben-Wege-Ungleichungen sehr effektiv ist, da die Wahrscheinlichkeit, dass sie auch als Schnittebene verwendet werden

können, hoch ist. Für die verschiedenen Dichten des Graphen liegt die Verwendbarkeit oft über 70%. Für die Cliquen-Ungleichungen liegt diese Verwendbarkeit dagegen, unabhängig von der Dichte des Graphen, zwischen 16,8 und 33,6%. Dies ist nicht sehr hoch, aber da die Cliquen-Ungleichungen auf jeden Fall enthalten sein sollten, da die Entwicklung der unteren Schranke sonst schlechter ist, ist sie annehmbar. Bei den Ungleichungen, die man anhand sehnenloser Kreise aufstellen kann, verschlechtert sich die Verwendbarkeit, je dichter der Graph ist. Bei einem Graph mit geringer Dichte (also einer Kantenwahrscheinlichkeit von 0,3) liegt die Verwendbarkeit bei etwa 85%, was eine sehr hohe Quote ist. Diese sinkt, bis sie bei einem Graph mit sehr hoher Dichte (Wahrscheinlichkeit 0,9) bei 11 bis 29% liegt. Die Block-Farben-Ungleichungen werden per Brute-Force aufgestellt, was die schlechten Ergebnisse erklärt. Der Algorithmus zur Suche dieser Ungleichungen wird nicht oft aufgerufen, aber wenn er aufgerufen wird, so liegt die Verwendbarkeit unter 5%. Am Schlechtesten fällt die Verwendbarkeit bei den Mehrfarben-Cliquen-Ungleichungen aus. Es werden zwar vereinzelt einigermaßen gute Ergebnisse, wie zum Beispiel 31,4% auf einem Graph mit sehr hoher Dichte erzielt, meistens liegt die Verwendbarkeit aber bei unter 1%. Wie in den vorherigen Betrachtungen, erhalten auch hier die Mehrfarben-Cliquen-Ungleichungen eine schlechte Bewertung.

Die genauen Ergebnisse und die Voraussetzungen, unter denen die Ergebnisse gefunden wurden, können in [4] nachgelesen werden.

6 Gesamtergebnis

Anhand der drei Beobachtungen aus Teil 5 kann man folgende Resultate ziehen:

- Cliquen-Ungleichungen sollten auf jeden Fall als Schnittebenen verwendet werden, da sie die Entwicklung der unteren Schranke positiv beeinflussen und somit wichtig für das Ergebnis des Schnittebenenverfahrens sind.
- Zusammen mit den Block-Farben-Ungleichungen und mit den Ungleichungen aus der Betrachtung sehnenloser Kreise erzielen sie in allen 3 Untersuchungsgebieten einigermaßen gute Ergebnisse. Mit dieser Kombination erzielt man eine gute Verbesserung der unteren Schranke in einer annehmbaren Zeit. Die Zeit, die zum Separieren verwendet wird, ist nicht zu hoch und auch die Verwendbarkeit der Ungleichungen als Schnittebenen ist vertretbar. Daher ist diese Kombination eine gute Grundlage zum Finden von Schnittebenen.
- Die anderen beiden Ungleichungsarten schneiden dagegen in fast allen Beobachtungen am schlechtesten ab. Die Zeit, die zum Finden der unteren Schranke benötigt wird, ist bei den Kombinationen, die diese Ungleichungen verwenden, höher. Es wird mehr Zeit zum Separieren benötigt, aber die gefundenen Ungleichungen können, im Falle der Mehrfarben-Cliquen-Ungleichungen, nur in sehr geringem Maße auch als Schnittebenen verwendet werden. Daher kann man zu dem Schluss kommen, dass diese Ungleichungen nicht geeignet sind, als Schnittebenen verwendet zu werden. Man könnte zum Beispiel den Algorithmus zuerst nur mit den anderen 3 Ungleichungsarten starten. Kann man mit diesen keine Verbesserung der unteren Schranke mehr erzielen oder findet man keine weiteren Ungleichungen, die als Schnittebenen dienen können, kann man immer noch auf diese beiden Arten zurückgreifen und versuchen, mit ihnen eine Verbesserung zu erzielen.

Insgesamt muss man sagen, dass dies nur eine Variante für ein Schnittebenenverfahren zum Färben von Graphen ist. Will man dieses verbessern, kann man dies an mehreren Stellen versuchen. Man kann versuchen, zu Beginn eine geeignetere Relaxation zu finden, so dass das Problem schneller gelöst werden kann. Oder man kann ein besseres Verfahren entwickeln, um eine genauere Abschätzung der unteren und oberen Schranken zu erreichen, so dass das Problem verkleinert wird und man so schneller zu einer Lösung gelangen kann. Dieses Verfahren sollte aber weiterhin recht einfach sein, da man immer abwägen muss, wieviel Aufwand sich lohnt, um eine bestimmte Verbesserung zu erzielen. Dies sind die Grundlagen des Algorithmus.

Das Herzstück des Algorithmus ist allerdings das Finden von als Schnittebenen geeigneten Ungleichungen. Gute Schnittebenen sind wichtig, da man mit ihnen weniger Iterationen benötigt, um zu einer optimalen Lösung zu gelangen. Wie sich herausgestellt hat, gehören die Cliquen-Ungleichungen zu dieser Gruppe, da sich die Entwicklung der unteren Schranke verbessert, wenn man diese Ungleichungsart als Schnittebenen verwendet. In Teil 5.3 wurde aber festgestellt, dass der Anteil der gefundenen Ungleichungen, die auch als Schnittebene verwendet werden können, nicht überragend ist. Ein Algorithmus oder eine Heuristik, die diesen Anteil in annehmbarer Zeit verbessert, würde auch die Ergebnisse des gesamten Algorithmus verbessern. Dies kann man natürlich nicht nur beim Aufstellen der Cliquen-Ungleichungen versuchen, sondern auch bei den übrigen. Es wäre zum Beispiel sinnvoll, für die Block-Farben-Ungleichungen eine andere Möglichkeit wie die Brute-Force Methode zu finden. Man darf dabei aber nicht vergessen, dass eine Brute-Force Methode, die viele unbrauchbare Ungleichungen aufstellt, sinnvoller sein kann als ein genauer Algorithmus, wenn dieser Algorithmus zu viel Zeit oder Speicherplatz benötigt, um eine Ungleichung aufzustellen. Man muss also immer den Nutzen neuer Methoden mit den dafür in Kauf genommenen Verschlechterungen vergleichen und abwägen, ob die Methoden sinnvoll sind.

Neben verbesserten Methoden zum Aufstellen von Schnittebenen aus dem Bereich der bekannten Facetten kann man auch versuchen, neue Ungleichungsarten zu finden, die man dann als Schnittebenen benutzen kann. Hat man solche Ungleichungen gefunden muss man Wege erarbeiten, diese effizient aufzustellen. So kann es sein, dass man mit Hilfe dieser neuen Ungleichungen schneller eine optimale Lösung finden kann. Es gibt auch Fälle, die mit den bisher betrachteten Ungleichungen nicht lösbar sind, da keine Schnittebenen mehr gefunden werden können, mit denen die untere Schranke verbessert werden kann, bis sie ganzzahlig ist. Mit den neu gefundenen Ungleichungen kann man dann im besten Falle zu einer optimalen ganzzahligen Lösung für jedes Färbungsproblem gelangen.

7 Benutzte Literatur

[1] H. Waldschmidt, K. Guntermann, Grundlagen der Informatik 2 Skript
[2] R. Diestel, Graphentheorie, 3. Auflage, Springer-Verlag, Berlin Heidelberg, 2006
[3] R. Hemmecke, Vorlesungsskript Algorithmische diskrete Mathematik, TU Darmstadt, 2009
[4] I. Méndez-Díaz, P. Zabala, A cutting plane algorithm for graph coloring
[5] Th. Ellinger, G. Beuermann, R. Leisten, Operations Research, 6. Auflage, Springer-Verlag, 2003
[6] A. Martin, Vorlesungsskript Optimierung, TU Darmstadt, 2008
[7] A. Martin, Vorlesungsskript Diskrete Optimierung, TU Darmstadt, 2006
[8] B. Pareigis, Lineare Algebra für Informatiker, Springer-Verlag, Berlin Heidelberg, 2000
[9] F. Jarre, J.Stoer, Optimierung, Springer-Verlag, Berlin Heidelberg, 2004