Technische Universität Wien
Vienna University of Technology

Dissertation

Knickminimales Orthogonales Zeichnen Planarer Graphen im Kandinsky Modell

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

Prof. Dr. Petra Mutzel
186 Institut für Computergraphik und Algorithmen

eingereicht an der Technischen Universität Wien
Fakultät für Informatik

von
Dipl.-Ing. Canan Yildiz
9827052
12., Vivenotgasse 57/10

Wien, am 20. Dezember 2005

Danksagung

An dieser Stelle möchte ich mich bei all jenen bedanken, die zur Fertigstellung dieser Arbeit beigetragen haben.

Mein besonderer Dank gilt meiner Betreuerin Frau Univ.Prof.Dr.techn. Petra Mutzel für Ihre wertvolle Hilfe.

Bei Herrn Univ.Prof.Dr.rer.nat. Wilhelm Barth möchte ich mich ebenfalls ganz herzlich bedanken. Er hat mich während der Abwesenheit von Frau Prof. Petra Mutzel betreut und ich habe viel Wertvolles von ihm gelernt.

Weiters möchte ich mich bei DI Karsten Klein und DI Markus Chimani bedanken, die mit ihren wertvollen Ratschlägen zur Verbesserung dieser Arbeit beigetragen haben.

Zuletzt bedanke ich mich bei meinen Eltern und meinen Geschwistern, die mich während meiner Arbeit unterstützt und motiviert haben.

Kurzfassung

Graphen werden häufig zur Visualisierung von komplexen Zusammenhängen zwischen verschiedenen Objekten verwendet. Das Gebiet des Graphenzeichnens beschäftigt sich mit dem Problem, automatisch und effizient übersichtliche grafische Darstellungen für Graphen zu konstruieren, sodass die komplexe Struktur der zugrunde liegenden Informationen leicht erfassbar ist. Wir befassen uns mit dem Knickminimierungsproblem in Kandinsky Zeichnungen planarer Graphen, da die Übersichtlichkeit solcher Zeichnungen zum Großteil von der Anzahl der Knicke abhängt. Die Komplexität dieses Problems ist bis heute unbekannt. Wir führen einen neuen 2-Approximationsalgorithmus (Cyclic-Shift Algorithmus) ein der in der Praxis sehr gute Ergebnisse liefert.

Abstract

Graphs are widely used to visualize complex relations between objects. The field of graph drawing addresses the problem of generating clear drawings for graphs such that the underlying information is easy to conceive. In this work we deal with the problem of minimizing the number of bends in Kandinsky drawings of planar graphs, hence the clearity and readability of such drawings depends mostly on the number of bends. The complexity of this problem is yet unknown. We introduce a new 2-approximation algorithm (Cyclic-Shift algorithm) that yields very good results in praxis.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Graphen werden häufig zur Visualisierung von Zusammenhängen zwischen verschiedenen Objekten verwendet. Dabei werden die Objekte als Knoten modelliert, die grafisch als einfache geometrische Zeichen dargestellt werden. Besteht ein Zusammenhang zwischen zwei Objekten, werden die zugehörigen Knoten im Graph mit einer Kante verbunden, die in der Zeichnung wiederum als Linien oder Bögen zwischen den zugehörigen graphischen Objekten dargestellt werden.

Graph Zeichnungen kommen in den unterschiedlichsten Bereichen zum Einsatz:

- Mit Hilfe von PERT Diagrammen werden in der Netzplantechnik Abhängigkeiten im zeitlichen Ablauf von Projekten dargetellt.
- In der Softwareentwicklung werden mittels UML-Diagrammen die Struktur von Software Systemen visualisiert (Abb. 1.1).
- In der theoretischen Informatik werden Syntaxdiagramme zur Beschreibung der Syntax von formalen Sprachen benutzt.
- Entity-Relationship Diagramme werden zur Modellierung von Datenbanken eingesezt.
- Mit Hilfe von Feynman-Diagrammen (auch Feynman-Graphen) werden in der Teilchenphysik Wechselwirkungen von Elementarteilchen grafisch

dargestellt.

- Im VLSI-Design werden Graph Zeichnungen beim Entwurf hoch integrierter Schaltungen eingesetzt.

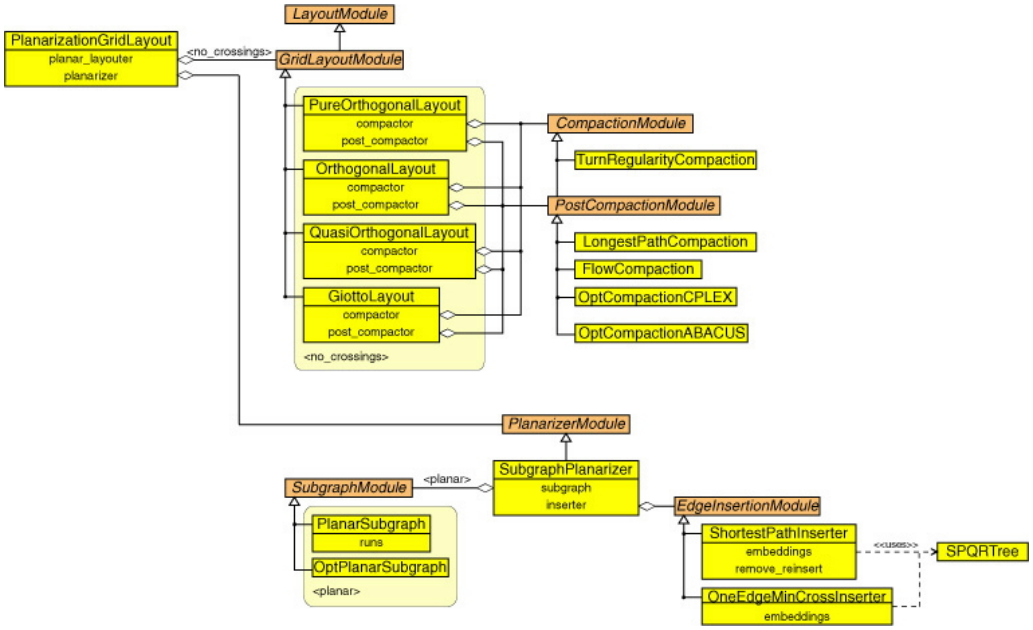

Abbildung 1.1: Dieses Klassendiagramm bietet einen Überblick über die Planarisation Methode von AGD.

Das Gebiet des Graphenzeichnens beschäftigt sich mit dem Problem, automatisch und effizient übersichtliche und *schöne* grafische Darstellungen für Graphen zu konstruieren, sodass die komplexe Struktur der zugrunde liegenden Informationen leicht erfassbar ist und dadurch das Verständnis des Sachverhaltes erleichtert wird. Es gibt jedoch keine beste Art einen Graphen zu zeichnen. Je nach Anwendungsgebiet werden unterschiedliche Anforderungen an die Zeichnung gestellt. Diese Anforderungen müssen spezifiziert und formal beschrieben werden, um im Rahmen automatisierter Zeichenverfahren effektiv eingesetzt werden zu können. Dabei kann zwischen wesentlichen und ästhetischen Forderungen unterschieden werden.

Bei den wesentlichen Anforderungen handelt es sich um gewisse Eigenschaften, die eine Zeichnung besitzen muss, um zulässig zu sein. Diese Eigen-

schaften können meist zu einem *Modell* zusammengefasst werden, die dann die Kriterien für die Zugehörigkeit einer Zeichnung zu diesem Modell festlegen. Beispiele sind:

- Kanten werden als gerade Linien dargestellt (geradlinige Zeichnungen).
- Kanten werden als zusammenhängende Folge von geraden Linien dargestellt (Polyline Zeichnungen)
- Kanten werden als zusammenhängende Folge von horizontalen und vertikalen Linien dargestellt (orhogonale Zeichnungen).
- Kanten überkreuzen sich nicht, ausser an ihren gemeinsamen Endknoten (kreuzungsfreie Zeichnungen).
- Knoten, Knicke und Kreuzungspunkte haben ganzzahlige Koordinaten (Gitter Zeichnungen).

Die ästhetischen Anforderungen legen Kriterien für eine *schöne* bzw. übersichtliche Zeichnung fest, die es *so weit wie möglich* zu erfüllen gilt. Beispiele sind:

- Anzahl von Knicken ist minimal.
- Die maximale Anzahl von Knicken auf einer Kante ist minimal.
- Anzahl von Kantenüberkreuzungen ist minimal.
- Gesamtfläche der Zeichnung ist minimal.
- Die Symmetrien des Graphen werden in der Zeichnung wiedergegeben.

Manche dieser Kriterien können dabei in Widerspruch zueinander stehen. Beispielsweise kann die Kreuzungsminimierung zu einem Anstieg in der Anzahl der Knicke führen (siehe Abb. 1.2(a),(b)). Wird Orthogonalität vorausgesetzt, erhöht sich die (minimale) Anzahl der Knicke weiter (Abb. 1.2(c)). Meistens entspricht die Einhaltung eines einzelnen Kriteriums schon einem

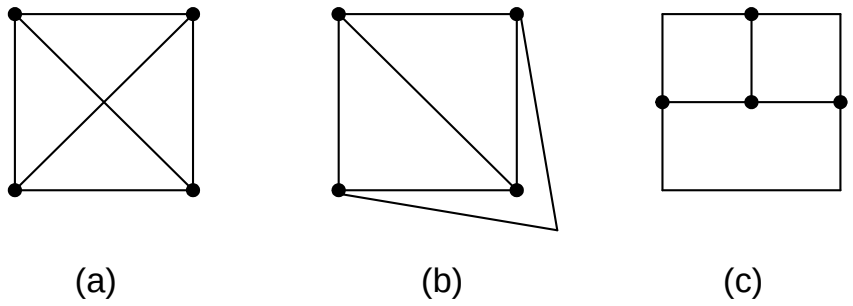

Abbildung 1.2: (a) Geradlinige Zeichnung, (b) kreuzungsfreie Zeichnung, (c) orthogonale, knickminimale Zeichnung

NP-schwierigen Problem ([DT81], [GJ83], [GT95]). Daher müssen im allgemeinen, je nach Anwendung und verwendetem Modell, Prioritäten gesetzt werden.

Eines der wichtigsten und am häufigsten eingesetzten Modelle ist das orthogonale Modell. Bei ER-Diagrammen, UML-Diagrammen, Schaltkreisentwürfen und vielen anderen Anwendungen wird dieses Modell verwendet (Abb. 1.1). Die Übersichtlichkeit einer orthogonalen Zeichnung wird hauptsächlich von der Anzahl der Kantenüberkreuzungen und Knicke beeinflusst, die es daher zu minimieren gilt.

Wir befassen uns in dieser Arbeit mit dem Knickminimierungsproblem bei orthogonalen Zeichnungen planarer Graphen. Eine Verallgemeinerung auf nicht planare Graphen ist möglich, indem diese vorher planarisiert werden.

Das folgende zweite Kapitel bietet einen Überblick über theoretische Grundlagen, die in den darauf folgenden Kapiteln benötigt werden. Im dritten Kapitel wird der von Tamassia in [Tam87] eingeführte Netzwerk-Fluss basierte Polynomialzeit-Algorithmus zur Berechnung knickminimaler orthogonaler Zeichnungen für planare Graphen mit Grad ≤ 4 vorgestellt. Bei diesem Algorithmus wird das Knickminimierungsproblem auf ein Min-Cost Flow (MCF) Problem zurückgeführt, wobei die planare Einbettung des Graphen beibehalten wird (Knickminimierung über alle Einbettungen ist NP-schwer [GT95]). Bei dem hier verwendeten orthogonalen Modell werden Knoten als Punkte dargestellt, sodass Kanten höchstens an vier Stellen an einen Knoten ansetzen

können. Daher können nur Graphen mit einem Grad ≤ 4 in diesem Modell gezeichnet werden. Das schränkt das Anwendungsgebiet solcher Zeichnungen erheblich ein.

In Kapitel 4 stellen wir das von Fößmeier und Kaufmann eingeführte Kandinsky Modell vor, in dem auch Graphen mit höherem Knotengrad orthogonal gezeichnet werden können. Durch eine Erweiterung des Ansatzes von Tamassia wird das Knickminimierungsproblem im Kandinsky Modell auf ein MCF Problem mit gewissen Nebenbedingungen zurückgeführt. Es stellt sich jedoch heraus, dass dieses Problem nicht ohne weiteres mit den klassischen, Netzwerk-Fluss basierten Algorithmen gelöst werden kann. Es handelt sich dabei um ein spezielles Min-Cost Flow Problem, dessen Koplexität bisher unbekannt ist. In [BDD00] führen Bertolazzi et.al. das Simple Kandinsky Modell ein. Das Knickminimierugnsproblem in diesem vereinfachten Modell kann auf ein gewöhnliches MCF Problem zurückgeführt und so in polynomialer Zeit gelöst werden.

In Kapitel 5 stellen wir zunächst die bisherigen Ansätze zur Lösung dieses Problems vor. In Abschn. 5.1 fassen wir den Fößmeier-Kaufmann Ansatz kurz zusammen und gehen näher auf die Probleme ein, die bei der Anwendung von klassischen MCF Algorithmen auf das KMCF Problem auftreten. In Abschn. 5.2 stellen wir den von Eiglsperger in [Eig03] eingeführten Transformationslagorithmus mit Gütegarantie 2 vor. In Abschn. 5.4 führen wir unseren neu entwickelten Cyclic-Shift Algorithmus ein, der ebenfalls den Approximationsfaktor 2 besitzt, in der Praxis jedoch viel bessere Ergebnisse liefert. Ein Vergleich der beiden 2-Approximationsalgorithmen in Abschn. 5.5 bietet eine Erklärung für den grossen Qualitätsunterschied zwischen den Lösungen dieser beiden Verfahern.

Kapitel 6 enthält die Ergebnisse und Auswertungen unserer Tests. Dabei haben wir unseren Cyclic-Shift Algorithmus, den Transformationsalgorithmus von Eiglsperger und den ebenfalls von Eiglsperger eingeführten modifizierten sukzessiven Transformationsalgorithmus getestet und die Näherungslösungen dieser drei Algorithmen mit den exakten Lösungen verglichen. Für die Tests haben wir neben den Rome Graphen [DGLTTV95] zusätzlich insgesamt 14000 zufällig generierte, planare Graphen aus unterschiedlichen Klas-

sen sowie 29 vollständige Graphen verwendet.

In Kapitel 7 stellen wir eine Variation des Kandinsky Modells, das *High Degree Kandinsky Modell*, vor und zeigen, dass das Knickminimierungsproblem in diesem Modell Fest-Parameter behandelbar ist. In Kapitel 8 bieten wir eine Zusammenfassung und schliessen die Arbeit mit einem Ausblick auf zukünftige Forschungsmöglichkeiten ab.

Kapitel 2

Theoretische Grundlagen

In diesem Kapitel führen wir zunächst einige Grundbegriffe und Notationen ein, die wir im weiteren benutzen werden. Für weitere Details zu den jeweiligen Themen sei auf [DETT99], [Har69], [AMO93] und [NW88] verwiesen.

2.1 Graphen

Definition 2.1 *Ein* ***Graph*** $G = (V, E)$ *besteht aus einer endlichen Menge* $V = V(G)$ *von Knoten und einer endlichen Menge* $E = E(G)$ *von Kanten. Jede Kante ist ein ungeordnetes Paar* (u, v) *von Knoten, das die Endknoten* u *und* v *der Kante miteinander verbindet.*

Ist (u, v) eine Kante in E, werden die Knoten u und v als *adjazent* oder *benachbart* bezeichnet. Die Menge aller Knoten, die mit einem festen Knoten $v \in V$ benachbart sind, wird als die *Nachbarschaftsmenge* $N(v)$ von v bezeichnet. Eine Kante $e \in E$ ist *inzident* mit einem Knoten v, wenn v ein Endknoten von e ist. In diesem Fall ist auch v inzident mit e. Zwei unterschiedliche Kanten sind *adjazent*, wenn sie mit einem gemeinsamen Knoten v inzident sind.

Der *Grad* $\deg(v)$ eines Knotens v ist die Anzahl der mit v inzidenten Kanten. Der *Minimalgrad* eines Graphen G ist als $\delta(G) = \min\{\deg(v) : v \in V\}$, der *Maximalgrad* als $\Delta(G) = \max\{\deg(v) : v \in V\}$ definiert. Einen Graphen mit Maximalgrad k bezeichnen wir als **k-Graph**.

Eine Kante (v, v) wird als **Schlinge** bezeichnet. Kommt eine Kante mehrfach in E vor, wird sie als **Mehrfachkante** bezeichnet. Ein Graph ohne Schlingen und Mehrfachkanten ist ein **einfacher Graph**.

Definition 2.2 *Ein* ***Kantenzug*** *der Länge n ist eine Folge $(v_0, v_1, ..., v_n)$ von Knoten, sodass $e_i = (v_{i-1}, v_i) \in E$ ist; im Fall von $v_0 = v_n$ spricht man von einem geschlossenen Kantenzug. Wenn die Kanten e_i paarweise verschieden sind, liegt ein* ***Weg*** *bzw. im Fall $v_0 = v_n$ ein geschlossener Weg vor. Falls auch die Knoten v_j paarweise verschieden sind, wird der Weg als* ***Pfad*** *bezeichnet. Ein Pfad ist ein* ***Kreis****, wenn $v_0 = v_n$ ist.*

In jedem dieser Fälle verwenden wir auch die Schreibweise $(e_1, ..., e_n)$.

Definition 2.3 *Ein Graph G ist* ***zusammenhängend****, wenn jedes Paar von Knoten in G durch mindestens einen Pfad miteinander verbunden ist. Andernfalls ist er* ***unzusammenhängend****.*

Definition 2.4 *Der* ***Zusammenhang*** *eines Graphen G ist die kleinste Anzahl von Knoten, durch deren Entfernung der Graph unzusammenhängend wird. G heißt* **k-*zusammenhängend***, *wenn der Zusammenhang von G gleich k ist.*

Definition 2.5 *Ein* ***Baum*** *ist ein zusammenhängender Graph, der keine Kreise enthält.*

Definition 2.6 *Ein Graph G mit n Knoten ist* ***vollständig*** *und wird mit K_n bezeichnet, wenn jedes Paar von Knoten in G adjazent ist.*

Definition 2.7 *Ein Graph G ist* ***serien-parallel****, wenn er durch Verdoppelung und/oder Unterteilung von Kanten rekursiv aus dem Graphen K_2 erzeugt werden kann.*

Dabei ist die *Verdoppelung* einer Kante $e = (u, v) \in E$ das Hinzufügen einer parallelen Kante $e' = (u, v)$ in E (Abb. 2.1, (a)→(b)). Die *Unterteilung* einer Kante (u, v) ist das Hinzufügen eines neuen Knotens w in V und das Ersetzen von (u, v) durch die zwei Kanten (u, w) und (w, v) (Abb. 2.1, (b)→(c)).

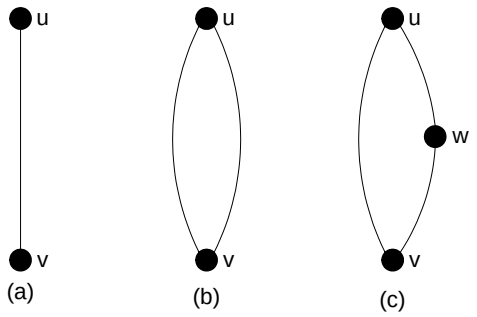

Abbildung 2.1: Verdoppelung und Unterteilung von Kanten.

Definition 2.8 *Ein **gerichteter Graph** $G = (V, E)$ besteht aus einer endlichen Menge $V = V(G)$ von Knoten und einer endlichen Menge $E = E(G)$ von Kanten, wobei jede Kante ein geordnetes Paar $(\overrightarrow{u, v})$ von Knoten ist, das den Anfangsknoten u mit dem Endknoten v verbindet.*

Eine Kante $e = (\overrightarrow{u, v})$ ist *Ausgangskante* von u und *Eingangskante* von v. Mit $\mathrm{Out}(v)$ bezeichnen wir die Menge aller Ausgangskanten, mit $\mathrm{In}(v)$ die Menge aller Eingangskanten eines Knotens v. Der *Ausgangsgrad* $\deg^+(v)$ eines Knotens v ist die Anzahl der Ausgangskanten von v. Gleichermaßen ist der Eingangsgrad $\deg^-(v)$ die Anzahl der Eingangskanten von v.

Die Begriffe *Kantenzug*, *Weg* und *Pfad* in einem gerichteten Graphen sind genau so definiert, wie in Def. 2.2; mit dem einzigen Unterschied, dass es sich hier um gerichtete Kanten $e_i = (\overrightarrow{v_{i-1}, v_i}) \in E$ handelt.

Wir gehen in dieser Arbeit davon aus, dass die zu zeichnenden Graphen ungerichtet sind. Dies ist keine Einschränkung, da bei gerichteten Graphen der zugrunde liegende ungerichtete Graph betrachtet werden kann. Gerichtete Graphen kommen im weiteren nur in Form von Netzwerken vor (siehe Abschn. 2.2).

Definition 2.9 *Eine **ebene Einbettung** P eines Graphen $G = (E, V)$ ist eine Funktion, die jedem Knoten $v \in V$ eindeutig einen Punkt $\Gamma(v)$ und jeder Kante $(u, v) \in E$ eine stetige, offene Jordan-Kurve $\Gamma(u, v)$ mit den Endpunkten $\Gamma(u)$ und $\Gamma(v)$ auf der zweidimensionalen Ebene zuordnet, sodass sich zwei Kurven nicht schneiden, außer an ihren gemeinsamen Endpunkten.*

*Ein Graph G ist **planar**, wenn er eine ebene Einbettung besitzt. Die Menge $\Gamma(V) = \{\Gamma(v) : v \in V\}$ zusammen mit der Menge $\Gamma(E) = \{\Gamma(u, v) : (u, v) \in E\}$ wird als eine **Zeichnung** $\Gamma(G)$ von G bezeichnet.*

Durch eine ebene Einbettung P ist für jeden Knoten v die zyklische Reihenfolge, in der die mit v inzidenten Kanten um diesen Knoten herum im Uhrzeigersinn angeordnet sind, eindeutig festgelegt. Zudem unterteilt P die zweidimensionale Ebene in topologisch zusammenhängende Regionen, die als *Flächen* bezeichnet werden. Die eine unbegrenzte Fläche wird als *Außenfläche* bezeichnet. Jede Fläche in einer ebenen Einbettung kann durch eine Liste von Kanten repräsentiert werden, die diese Fläche begrenzen.

Definition 2.10 *Sei $G = (V, E, F)$ ein planarer Graph mit einer fixen, ebenen Einbettung P, wobei F die Menge der durch P festgelegten Flächen ist. Die **planare Repräsentation** von G ist eine Menge $\mathcal{P}$ von zyklischen Kantenlisten $P(f)$, jeweils eine Liste für jede Fläche $f \in F$. Dabei enthält $P(f)$ die f begrenzenden Kanten, und zwar in der Reihenfolge, in der sie am Rand von f gegen den Uhrzeigersinn angeordnet sind.*

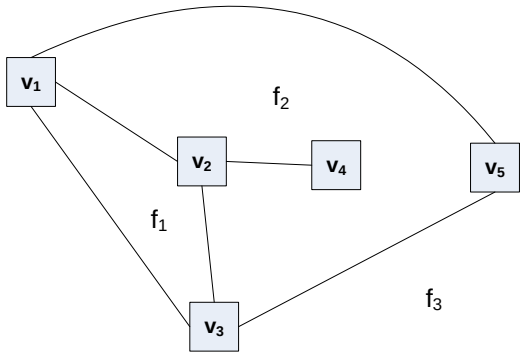

Abbildung 2.2: Ein Beispiel für eine planare Einbettung. f_3 ist die Aussenfläche.

Ein Beispiel für eine planare Repräsentation ist in Tab. 2.1 gegeben. Man beachte, dass jede Kante (u, v) genau zwei Mal in diesen Listen vorkommt, da sie an zwei Flächen angrenzt. Sie wird aber von diesen zwei Flächen aus

$$\begin{aligned} P(f_1) = &\quad ((v_2, v_1),\ (v_1, v_3),\ (v_3, v_2)) \\ P(f_2) = &\quad ((v_1, v_2),\ (v_2, v_4),\ (v_4, v_2),\ (v_2, v_3),\ (v_3, v_5),\ (v_5, v_1)) \\ P(f_3) = &\quad ((v_1, v_5),\ (v_5, v_3),\ (v_3, v_1)) \end{aligned}$$

Tabelle 2.1: Die Planare Repräsentation des Graphen in Abb. 2.2

in unterschiedliche Richtungen durchlaufen. Im Zusammenhang mit planaren Repräsentationen unterscheiden wir in diesem Sinne zwischen $(\overrightarrow{u,v})$ und $(\overrightarrow{v,u})$ und betrachten sie als zwei unterschiedliche Ausrichtungen derselben Kante.

Definition 2.11 *Sei $G = (V, E, F)$ ein planarer Graph mit einer fixen Einbettung P. Für jede ungerichtete Kante $(u, v) \in E$ mit den Endpunkten u und v, bezeichnen wir die zwei möglichen Ausrichtungen $(\overrightarrow{u,v})$ und $(\overrightarrow{v,u})$ dieser Kante als **Richtungskanten**. Wir definieren die Menge $\overrightarrow{E}$ der Richtungskanten mit*

$$\overrightarrow{E} := \{(\overrightarrow{v,u}), (\overrightarrow{u,v}) : (u, v) \in E\}.$$

Es ist $(\overrightarrow{u,v}) \in P(f)$ genau dann, wenn die Fläche f links von der Richtungskante $(\overrightarrow{u,v})$ liegt. In diesem Fall schreiben wir $\text{face}(\overrightarrow{u,v}) = f$. Den (zyklischen) Nachfolger einer Richtungskante $(\overrightarrow{u,v})$ in der Liste $P(f)$bezeichnen wir mit $\text{succ}_f(\overrightarrow{u,v})$, ihren (zyklischen) Vorgänger mit $\text{pred}_f(\overrightarrow{u,v})$.

Definition 2.12 *Sei $G = (V, E, F)$ ein planarer Graph mit einer fixen Einbettung P und sei $(u, v) \in E$. Wir bezeichnen die Richtungskante $(\overrightarrow{v,u})$ als die **Umkehrung** der Richtungskante $(\overrightarrow{u,v})$ und schreiben*

$$\overline{(\overrightarrow{u,v})} = (\overrightarrow{v,u})$$

2.2 Flüsse in Netzwerken

Definition 2.13 *Ein **Min-Cost Flow (MCF) Netzwerk** ist ein gerichteter, zusammenhängender Graph $\mathcal{N} = (N, A)$, in dem mit jeder Kante $a \in A$ eine untere Kapazitätsgrenze $\text{lcap}(a)$, eine obere Kapazitätsgrenze $\text{ucap}(a)$*

und Kosten $\mathrm{cost}(a)$ *assoziiert sind. Außerdem ist mit jedem Knoten* $n \in N$ *ein Bedarf* $b(n)$ *assoziiert, sodass*

$$\sum_{n \in N} b(n) = 0$$

ist. Ein Knoten mit positivem Bedarf wird als Quelle, einer mit negativem Bedarf als Senke bezeichnet. Einen Knoten, dessen Bedarf gleich Null ist, bezeichnen wir als ausgeglichen.

Wenn nicht anders angegeben ist wird angenommen, dass für alle Kanten a im Netzwerk $\mathrm{lcap}(a) = 0$ ist. Für Netzwerkkanten verwenden wir die Schreibweise $(i, j) \in A$ anstatt von $\left(\overrightarrow{i, j}\right) \in A$, wenn es klar ist, dass es sich um eine gerichtete Netzwerkkante handelt.

Definition 2.14 *Ein* ***Fluss*** *auf einem Netzwerk* $\mathcal{N}$ *ist eine Funktion* x*, die jeder Kante* $a \in A$ *eine positive Zahl* $x(a)$ *zuordnet. Ein Fluss* x *auf* $\mathcal{N}$ *ist gültig, wenn sie die folgenden zwei Bedingungen erfüllt:*

$$\mathrm{lcap}(a) \leq x(a) \leq \mathrm{ucap}(a) \qquad \forall a \in A \tag{2.1}$$

$$\sum_{a \in \mathrm{Out}(n)} x(a) - \sum_{a \in \mathrm{In}(n)} x(a) = b(n) \qquad \forall n \in N. \tag{2.2}$$

(2.1) wird als die Kapazitätsbedingung, (2.2) als die Flusserhaltungsbedingung bezeichnet.

Man kann sich den Fluss als eine Art Warentransport vorstellen: eine Ware soll von den Quell-Knoten, die einen Überschuss an Ware besitzen, über die Kanten im Netzwerk an die Senken (Knoten mit Warendefizit) verteilt werden. Die oberen Kapazitätsgrenzen geben an, wieviele Einheiten höchstens durch eine Kante transportiert werden dürfen. Offensichtlich sollte die Warenmenge, die einen Quellknoten n verlässt, genau um $b(n)$ größer sein als die Warenmenge, die den Knoten erreicht, da zusätzlich zu den Waren, die den Knoten nur passieren, auch die überschüssige Ware abtransportiert

werden muss. Entsprechendes gilt für die Senken und ausgeglichenen Knoten. Dies wird durch die Flusserhaltungsbedingung an den Knoten sichergestellt. Der Transport einer Wareneinheit über eine Kante a verursacht dabei $\text{cost}(a)$ Kosten. Die *Kosten* des gesamten Flusses x sind also durch

$$\text{cost}(x) = \sum_{a \in A} x(a) \cdot \text{cost}(a)$$

gegeben. Der *Wert* eines Flusses ist

$$\sum_{n \in N : b(n) > 0} \sum_{a \in \text{Out}(n)} x(a).$$

Definition 2.15 *Das Problem, auf einem gegebenen Netzwerk $\mathcal{N} = (N, A)$ einen gültigen Fluss x zu finden, sodass die Gesamtkosten $\text{cost}(x)$ minimal sind, wird als das* ***Min-Cost Flow (MCF) Problem*** *bezeichnet.*

Es gibt effiziente Netzwerk-Fluss Algorithmen, die dieses Problem in polynomialer Zeit lösen können [AMO93]. Diese können in zwei Hauptgruppen unterteilt werden: die sogenannten *Feasible Flow* Algorithmen und die *Optimal Infeasible Flow* Algorithmen. Die ersteren beginnen mit einem gültigen, jedoch nicht optimalen Startfluss, berechnen iterativ neue, kostengünstigere Flüsse und terminieren, wenn der aktuelle Fluss optimal ist. Die Optimalität des aktuellen Flusses wird dabei mit Hilfe gewisser Optimalitätskriterien ermittelt. Die Optimal Infeasible Flow Algorithmen beginnen mit einem ungültigen aber optimalen Startfluss und berechnen iterativ neue Flüsse, bis sie einen gültigen Fluss erreicht haben. Dabei bleibt die Optimalität des aktuellen Flusses in jedem Iterationsschritt erhalten.

Sei nun $\mathcal{N} = (N, A)$ ein MCF Netzwerk, x' der aktuelle Fluss in einem Zwischenschritt eines MCF Algorithmus und $(i, j) \in A$ eine Kante mit dem Flusswert $x'(i, j)$. Dann können noch $\text{ucap}(i, j) - x'(i, j)$ Einheiten von i nach j über die Kante (i, j) gesendet werden. Außerdem können bis zu $x'(i, j)$ Einheiten von j nach i über die Kante (i, j) gesendet werden, was dem Aufheben des Flusses auf dieser Kante gleichkommt. Während das Senden einer Einheit von i nach j über (i, j) die Kosten um $\text{cost}(i, j)$ erhöht,

reduziert das Senden einer Einheit von j nach i über (i,j) die Kosten um $\text{cost}(i,j)$.

Unter Verwendung dieser Überlegungen wird das Restnetzwerk bezüglich eines Flusses x' wie folgt aufgebaut: Jede Kante (i,j) im Netzwerk $\mathcal{N}$ wird durch die Kanten (i,j) und (j,i) ersetzt. Dabei hat die Kante (i,j) die Kosten $\text{cost}(i,j)$ und die **Restkapazität** $\text{rcap}(i,j) = \text{ucap}(i,j) - x'(i,j)$, die Kante (j,i) die Kosten $-\text{cost}(i,j)$ und die Restkapazität $x'(i,j)$. **Das Restnetzwerk** $\mathcal{N}(x')$ besteht aus den Kanten des Netzwerkes, die bezüglich des Flusses x' eine positive Restkapazität besitzen.

Jeder Pfad P im Restnetzwerk $\mathcal{N}(x')$ wird als ein **augmentierender Pfad** bezüglich des Flusses x' bezeichnet. Die Restkapazität des Pfades P ist durch

$$\text{rcap}(P) = \min_{(i,j)\in P} [\text{rcap}(i,j)] \tag{2.3}$$

gegeben, die Kosten (auch Länge) des Pfades P durch

$$\text{cost}(P) = \sum_{(i,j)\in P} \text{cost}(i,j) \tag{2.4}$$

Ein Pfad P von einem Knoten s zu einem Knoten t mit minimalen Kosten wird als ein **kürzester Pfad** (shortest path) bezeichnet.

Definition 2.16 *Das Problem, von einem gegebenen Knoten s je einen kürzesten Pfad zu allen anderen Knoten im Netzwerk zu bestimmen, wird als das* ***Shortest Path Problem*** *(Kürzeste Wege Problem) bezeichnet. Einen kürzesten Pfad von einem Knoten s zu einem Knoten i bezeichnen wir mit $SP(s,i)$. Die* ***kürzeste Distanz*** *$d(i)$ eines Knotens i ist definiert als die Kosten eines kürzesten Pfades von s nach i.*

Dijkstra's Algorithmus

Auf einem Netzwerk, in dem alle Kanten nicht-negative Kosten besitzen, kann das Shortest Path Problem mit *Dijkstra's Algorithmus* gelöst werden. Dieser Algorithmus berechnet für jeden Knoten $i \in N$ eine Distanzvariable $d(i)$ und einen Vorgängerindex $pred(i)$. Die Grundidee des Algorithmus ist

es, von dem Knoten s ausgehend alle Knoten des Netzwerkes in der Reihefolge ihrer Distanz zu s durchzugehen und sie nacheinander permanent zu markieren. Bei einem permanent markierten Knoten gibt die Distanzvariable die kürzeste Distanz des Knotens an. Bei einem nicht permanent markierten Knoten entspricht sie einer oberen Grenze für seine kürzeste Distanz.

Die kürzesten Pfade von s zu allen anderen Knoten im Netzwerk bilden einen gerichteten Baum mit der Wurzel s (Shortest Path Tree), in dem jeder Knoten i von s aus durch einen gerichteten Pfad $\mathrm{SP}(s,i)$ erreicht werden kann. Dieser Baum wird im Vektor *pred* festgehalten: für jeden Knoten i kann der kürzeste Pfad $\mathrm{SP}(s,i)$ mit Hilfe des Vektors *pred* zurückverfolgt werden.

Algorithmus 2.1 ***Dijkstra's Algorithmus***

Eingabe: *Ein Netzwerk $\mathcal{N} = (N, A)$ und ein gegebener Knoten s aus N.*

Ausgabe: *Ein Vektor d, der die kürzesten Distanzen von s zu allen anderen Knoten im Netzwerk angibt und ein Vektor pred, der für jeden Knoten i seinen Vorgängerknoten auf dem kürzesten Pfad von s nach i angibt.*

D1 Initialisierung: $d(s) = 0$ *und* $d(i) = \infty \ \forall i \in N \setminus \{s\}$.

D2 (Wahl eines Knotens) Es wird ein nicht permanent markierter Knoten i mit der kleinsten Distanz ausgewählt und permanent markiert.
$\longrightarrow$ *Sind alle Knoten permanent markiert, terminiert der Algorithmus und liefert d und pred zurück.*

D3 (Aktualisieren der Distanzen) Es werden alle Kanten $(i,j) \in \mathrm{Out}(i)$ *durchgegangen: Ist* $d(i)+\mathrm{cost}(i,j) < d(j)$, *wird* $d(j) = d(i)+\mathrm{cost}(i,j)$ *und* $pred(j) = i$ *gesetzt.*

In Abbildung 2.3 ist der Algorithmus auf einem Beispiel illustriert. Im ersten Schritt des Algorithmus wird s gewählt, da es die kleinste Distanz besitzt, und den Knoten 1 und 2 werden jeweils die Distanzen 2 und 3 zugewiesen (Abb. 2.3, links). In den folgenden Schritten werden die Knoten 1,2 und 3 gewählt und die entsprechenden Aktualisierungen durchgeführt. Am Ende ist der gestrichelt markierte Pfad $(s,1,2,3)$ mit Kosten 4 der kürzeste Pfad von s nach 3.

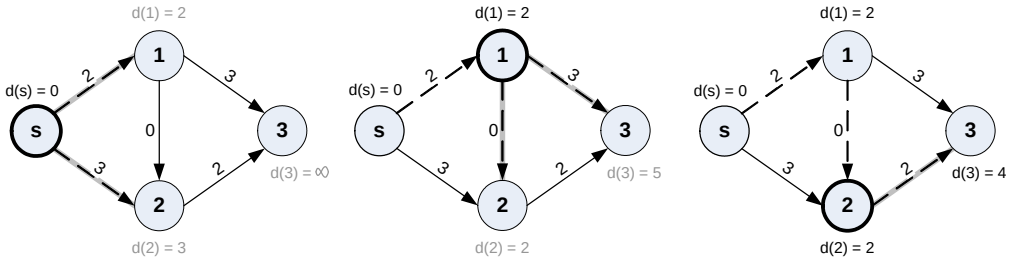

Abbildung 2.3: Berechnung der kürzesten Pfade von s zu allen anderen Knoten im Netzwerk mittels Dijkstra's Algorithmus.

Satz 1 *[AMO93] (**Kürzeste Wege Optimalitätskriterien**)*
Eine Menge von Distanzvariablen $d(i)$ definiert kürzeste Distanzen von einem gegebenen Knoten s zu allen anderen Knoten im Netzwerk $\mathcal{N} = (N, A)$, genau dann, wenn sie Distanzen von s ausgehender Pfade repräsentiert und die folgenden Kürzeste Wege Optimalitätskriterien erfüllt:

$$d(j) \leq d(i) + \text{ cost}(i,j) \quad \forall (i,j) \in A \tag{2.5}$$

Es kann leicht nachvollzogen werden, dass für jede Kante (i, j), die auf einem kürzesten Pfad liegt, $d(j) = d(i) + \text{cost}(i,j)$ ist.

Definition 2.17 *Es sei mit jedem Knoten $i \in N$ eine Zahl $\pi(i)$ assoziiert, die als das **Knotenpotential** von i bezeichnet wird. Die **reduzierten Kosten** einer Kante (i, j) in Bezug auf die Knotenpotentiale π sind wie folgt definiert:*

$$\text{cost}^{\pi}(i,j) = \text{cost}(i,j) - \pi(i) + \pi(j). \tag{2.6}$$

Durch die Einführung der reduzierten Kosten wird es unter anderem möglich, das folgende Kriterium für die Optimalität eines Flusses zu formulieren:

Satz 2 *[AMO93] (**Reduzierte Kosten Optimalitätskriterien**)*
Ein gültiger Fluss x^* auf einem MCF Netzwerk $\mathcal{N}$ ist eine optimale Lösung des Min-Cost Flow Problems, genau dann, wenn er in Bezug auf gewisse Knotenpotentiale π die folgenden Reduzierte Kosten Optimalitätskriterien erfüllt:

$$\text{cost}^{\pi}(i,j) \geq 0 \quad \textit{für jede Kante } (i,j) \textit{ im Restnetzwerk } \mathcal{N}(x^*). \tag{2.7}$$

Bei der Einhaltung dieser Kriterien spielen die folgenden zwei Lemmata eine wichtige Rolle.

Lemma 2.1 *[AMO93] Sei x ein Fluss auf $\mathcal{N}$, der die Reduzierte Kosten Optimalitätskriterien in Bezug auf gewisse Knotenpotentiale π erfüllt. Sei d ein Vektor, der die kürzesten Distanzen von einem Knoten s zu allen anderen Knoten im Restnetzwerk $\mathcal{N}(x)$ bezüglich der reduzierten Kosten cost^{π} repräsentiert. Dann gilt:*

(a) x erfüllt die Reduzierte Kosten Optimalitätskriterien auch in Bezug auf die Knotenpotentiale $\pi' = \pi - d$.

(b) Die reduzierten Kosten $\text{cost}^{\pi'}(i,j)$ sind gleich Null für alle Kanten (i,j), die auf einem kürzesten Pfad von s zu einem anderen Knoten im Netzwerk liegen.

Lemma 2.2 *Sei x ein Fluss, der die Reduzierte Kosten Optimalitätskriterien erfüllt und sei x' ein Fluss, der von x durch Augmentierung des Flusses entlang eines kürzesten Pfades von s zu einem anderen Knoten i erhalten wird. Dann erfüllt auch x' die Reduzierte Kosten Optimalitätskriterien.*

Der Successive Shortest Path Algorithmus

Der Successive Shortest Path Algorithmus (SSP) beginnt mit einem 0-Fluss, der die Flusserhaltungsbedingung an Quellen und Senken verletzt und versucht durch suksezzive Augmentierungen des Flusses entlang kürzester Pfade einen gültigen Fluss zu erzielen. Die Optimalität des Flusses wird in jedem Schritt des SSP durch die Einhaltung der Reduzierte Kosten Optimalitätskriterien (Def. 2) gewährleistet. Anstatt der tatsächlichen Kosten $\text{cost}(i,j)$ arbeitet SSP mit den sogenannten reduzierten Kosten $\text{cost}^{\pi}(i,j)$ einer Kante (i,j) (Def. 2.17).

Algorithmus 2.2 ***Der Successive Shortest Path Algorithmus (SSP)***
Eingabe: *Ein MCF Netzwerk $\mathcal{N} = (N, A)$.*
Ausgabe: *Ein gültiger Fluss x auf $\mathcal{N}$ mit minimalen Kosten $\text{cost}(x)$.*

SSP1 *Initialisierung:* $x := 0$ *und* $\pi := 0$.
$\longrightarrow$ *Der Startfluss* x^0 *erfüllt somit die Kapazitätsbedingung und die Reduzierte Kosten Optimalitätskriterien bezüglich der Knotenpotentiale* π, *da* $\text{cost}^{\pi}(i,j) = \text{cost}(i,j) \geq 0$ *für alle Kanten* (i,j) *in* $\mathcal{N} = \mathcal{N}(x^0)$ *ist.*

SSP2 *Es wird ein Knoten* s *mit* $b(s) > 0$ *und ein Knoten* t *mit* $b(t) < 0$ *im Restnetzwerk ausgewählt.*
$\longrightarrow$ *Ist* $b(n) = 0\ \forall n \in N$, *terminiert der Algorithmus mit einer optimalen Lösung des MCF Problems.*

SSP3 *Es werden die kürzesten Distanzen* $d(j)$ *und kürzesten Pfade* $SP(s,j)$ *von* s *zu allen anderen Knoten* j *im Restnetzwerk bezüglich der reduzierten Kosten* cost^{π} *berechnet.*
$\longrightarrow$ *Da alle reduzierten Kosten nicht negativ sind, kann dazu Dijkstra's Algorithmus benutzt werden.*

SSP4 *Die Knotenpotentiale werden aktualisiert:* $\pi := \pi - d$.

SSP5 *Der Fluss wird entlang eines kürzesten Pfades* $P = SP(s,t)$ *von* s *nach* t *im Restnetzwerk um* $\delta = \min[b(s), -b(t), \text{rcap}(P)]$ *Einheiten augmentiert.*
$\longrightarrow$ *Jede Kante* $(i,j) \in P$ *hat die reduzierten Kosten Null (Lemma 2.1(b)). Somit hat jede (evtl. neue) Kante* (j,i) *im Restnetzwerk ebenfalls die reduzierten kosten Null.*

SSP6 *Die reduzierten Kosten* cost^{π} *und das Restnetzwerk werden aktualisiert. Der Bedarf der Knoten* s *und* t *wird aktualisiert:* $b(s) := b(s) - \delta$ *und* $b(t) := b(t) + \delta$. *Weiter zu Schritt SSP2.*
$\longrightarrow$ *Aufgrund von Lemma 2.1(a) und Lemma 2.2 erfüllt der neue Fluss* x *die Reduzierte Kosten Optimalitätskriterien in Bezug auf die aktualisierten Knotenpotentiale* π.

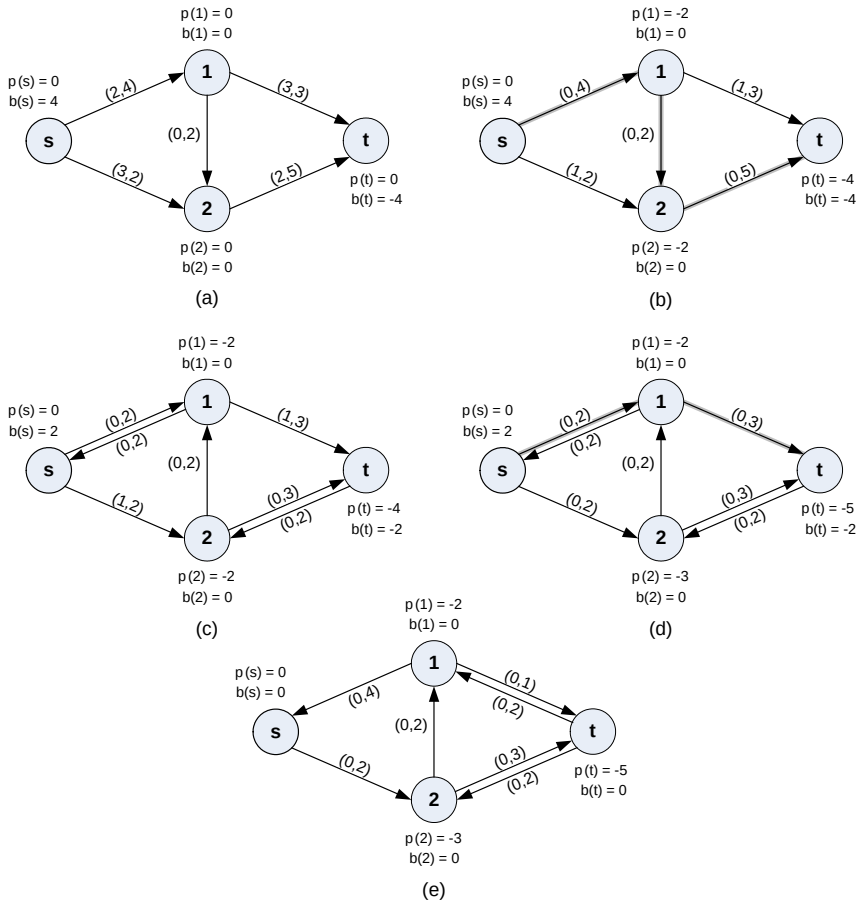

Abbildung 2.4: (a) Das Startnetzwerk mit $x = 0$ und $\pi = 0$; (b) Netzwerk nach Aktualisierung der Knotenpotentiale π und der reduzierten Kosten; (c) Netzwerk nach der Augmentierung von 2 Einheiten entlang des Pfades (s,1,2,t); (d) Netzwerk nach erneuter Aktualisierung; (e) Netzwerk nach der Augmentierung von 2 Einheiten entlang des Pfades (s,1,t).

2.3 Lineare Programmierung

Definition 2.18 *Ein **Lineares Programm (LP)** ist ein Optimierungsproblem, in der es gilt, einen Vektor $(x_1, ..., x_n) \in \mathbb{R}^n$ zu finden, die eine gegebene lineare Zielfunktion*

$$z(x) = c^T x \tag{2.8}$$

minimiert (bzw. maximiert) und die linearen Nebenbedingungen

$$Ax \leq b \tag{2.9}$$

$$x \geq 0 \tag{2.10}$$

erfüllt.

Dabei ist $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ und $c \in \mathbb{R}^n$. Lineare Programme können in polynomialer Zeit gelöst werden, z.B. mit Hilfe der Ellipsoid Methode [PS82].

Man beachte, dass eine lineare Ungleichung der Form $a^T x \geq b'$ auch als $-a^T x \leq b'$ geschrieben werden kann. Eine lineare Gleichung $a^T x = b'$ kann durch die zwei Ungleichungen $a^T x \geq b'$ und $-a^T x \geq -b'$ ersetzt werden. Daher können wir uns ohne Beschränkung der Allgemeinheit auf lienare Nebenbedingungen wie in Gl. 2.9 beschränken.

Definition 2.19 *Ein **Ganzzahliges Lineares Programm (ILP)** ist ein LP, in dem zusätzlich die Ganzzahligkeit der Variablen gefordert wird:*

$$\min \; c^T x$$

sodass

$$\begin{aligned} Ax &\leq b \\ x &\geq 0 \\ & x \text{ ganzzahlig} \end{aligned}$$

Aufgrund der zusätzlichen Ganzzahligkeitsbedingung wird das ILP zu einem NP-schweren Problem [GJ79].

Definition 2.20 *Eine ganzzahlige quadratische Matrix wird als* ***unimodular*** *bezeichnet, wenn ihre Determinante den Wert* 1 *oder* -1 *hat. Eine ganzzahlige Matrix A wird als* ***total unimodular*** *bezeichnet, wenn jede quadratische, reguläre Teilmatrix von A unimodular ist.*

Lässt man bei einem ILP die Ganzzahligkeitsbedingung weg, erhält man die **LP-Relaxierung** des ILPs. Ist die Koeffizientenmatrix A eines ILPs total unimodular und ist b ganzzahlig, dann ist jede optimale Basislösung der LP-Relaxierung automatisch ganzzahlig [PS82]. In dem Fall kann eine ganzzahlige optimale Lösung für das ILP in polynomialer Zeit berechnet werden.

Formuliert man das Min-Cost Flow Problem als ganzzahliges Lineares Programm, dann ist die Koeffizientenmatrix A dieses ILPs total unimodular. Sind zudem alle unteren und oberen Kapazitätsgrenzen ganzzahlig, hat das MCF Problem immer eine ganzzahlige optimale Lösung, die in polynomialer Zeit berechnet werden kann.

Kapitel 3

Orthogonale Zeichnungen

Eine *orthogonale Zeichnung* eines planaren Graphen G ist eine Zeichnung, in der jeder Knoten als ein Punkt und jede Kante als eine alternierende Folge von horizontalen und vertikalen Liniensegmenten dargestellt wird, wobei zwei Kanten sich nur an ihren gemeinsamen Endpunkten schneiden. Die Punkte, an denen eine Kante ihre Richtung ändert, werden als *Knicke* bezeichnet. Je weniger Knicke eine orthogonale Zeichnung enthält, desto übersichtlicher ist sie. In Abb. 3.1(b) und (c) sind zwei orthogonale Zeichnungen desselben planaren Graphen mit jeweils 7 und 3 Knicken zu sehen. Man beachte, dass ein Graph G nur dann orthogonal gezeichnet werden kann, wenn $\Delta(G) \leq 4$ ist, da in einer orthogonalen Zeichnung eine Kante höchstens an vier Punkten an einen Knoten ansetzen kann.

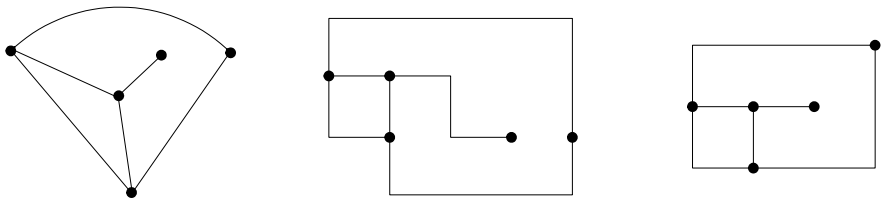

Abbildung 3.1: Zwei orthogonale Zeichungen eines planaren Graphen (a): eines hat 7 Knicke (b), das andere 3 Knicke (c).

In diesem Kapitel wird zunächst die Orthogonale Repräsentation defi-

niert, die eine formale Beschreibung orthogonaler Zeichnungen ermöglicht. Danach wird ein Netzwerk-Fluss basierter Algorithmus [Tam87] zur Berechnung von knickminimalen orthogonalen Zeichnungen planarer 4-Graphen vorgestellt, wobei die gegebene Einbettung des Graphen beibehalten wird. Das Problem, eine knickminimale Zeichnung über alle Einbettungen zu finden, ist NP-vollständig [GT95].

3.1 Die Orthogonale Repräsentation

Die orthogonale Repräsentation erhält man durch eine Erweiterung der planaren Repräsentation. Dabei wird zu jeder Richtungskante angegeben, welchen Winkel sie mit ihrem Nachfolger an ihrem Zielknoten formt und welche Knicke auf der linken Fläche entlang dieser Kante auftreten.

Aus Gründen der Übersichtlichkeit werden wir im folgenden die Richtungskanten $(\overrightarrow{u,v})$ mit (u,v) angeben. Man beachte, dass es weiterhin geordnete Paare von Knoten sind. Da bei Richtungskanten explizit angegeben wird, dass es sich um solche handelt, sollten Verwechslungen ausgeschlossen sein.

Definition 3.1 *Eine **orthogonale Repräsentation** (auch orthogonale Darstellung) eines 4-planaren Graphen $G = (V, E, F)$ ist eine Menge $\mathcal{H}$ von Flächenbeschreibenden, zyklischen Listen $H(f)$, jeweils eine für jede Fläche $f \in F$. Jedes Element $r_{(u,v)}$ einer solchen Liste ist ein Tripel $\big((u,v), s_{(u,v)}, \alpha_{(u,v)}\big)$, das wie folgt definiert ist:*

- *(u,v) ist eine Richtungskante aus $\bar{E}$.*
- *$s_{(u,v)}$ ist eine Bitfolge, die beschreibt, welche Knicke entlang der Richtungskante (u,v) auftreten: das k-te Bit entspricht dem k-ten Knick auf der linken Seite von (u,v). Dabei bezeichnet eine 0 einen $90°$-Knick und eine 1 einen $270°$-Knick. Eine leere Folge beschreibt eine gerade Kante.*
- *Die Zahl $\alpha_{(u,v)}$ ist eine ganze Zahl aus der Menge $\{90, 180, 270, 360\}$ und beschreibt den Winkel, den die Richtungskanten (u,v) und $(v,w) =$*

$\mathrm{succ}_f(u,v)$ *in der Fläche* f *formen (dabei ist* $r_{(v,w)}$ *der Nachfolger von* $r_{(u,v)}$ *in der zyklischen Liste* $H(f)$*).*

In Tabelle 3.1 ist ein Beispiel für eine orthogonale Repräsentation gegeben.

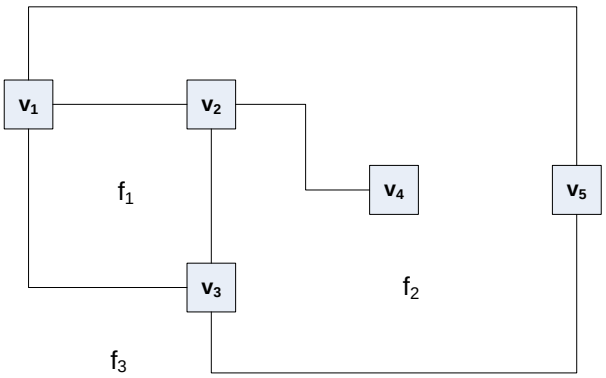

Abbildung 3.2: Eine orthogonale Zeichnung des planaren Graphen in Abb. 2.2

$$
\begin{array}{ll}
H(f_1) = & (((v_2,v_1),\varepsilon,90),\ ((v_1,v_3),0,90),\ ((v_3,v_2),\varepsilon,90)) \\
H(f_2) = & (((v_1,v_2),\varepsilon,180),\ ((v_2,v_4),10,360),\ ((v_4,v_2),10,90), \\
 & ((v_2,v_3),\varepsilon,180),\ ((v_3,v_5),00,180),\ ((v_5,v_1),00,90)) \\
H(f_3) = & (((v_1,v_5),11,180),\ ((v_5,v_3),11,90),\ ((v_3,v_1),1,180))
\end{array}
$$

Tabelle 3.1: Die Orthogonale Darstellung der Zeichnung in Abb. 3.2

Natürlich repräsentiert nicht jede derartige Menge von zyklischen Listen eine orthogonale Zeichnung. Es kann mit Hilfe elementarer Geometrie gezeigt werden, dass die folgende Aussage gültig ist:

Kriterium 3.1 *Eine Menge* $\mathcal{H}$ *von zyklischen Listen ist eine gültige orthogonale Darstellung einer orthogonalen Zeichnung eines planaren* 4*-Graphen* $G = (V, E, F)$*, genau dann, wenn sie die folgenden Kriterien erfüllt:*

(Kr1) G *besitzt eine planare Repräsentation, die durch die* (u,v)*-Felder der Listen in* $\mathcal{H}$ *gegeben ist.*

(Kr2) *Für jedes Paar von Elementen $r_{(u,v)}$ und $r_{(v,u)}$ in $\mathcal{H}$, erhält man die Bit-Folge $s_{(v,u)}$ durch die bitweise Negierung der Umkehrung von $s_{(u,v)}$.*

(Kr3) *Wir definieren für jedes Element $r_{(u,v)}$ in $\mathcal{H}$ die Rotation $\rho(u,v)$ wie folgt:*

$$\rho(u,v) = \left|s_{(u,v)}\right|_0 - \left|s_{(u,v)}\right|_1 + (2 - \frac{\alpha_{(u,v)}}{90}) \tag{3.1}$$

wobei $\left|s_{(u,v)}\right|_0$ die Anzahl der Nullen und $\left|s_{(u,v)}\right|_1$ die Anzahl der Einsen im String $s_{(u,v)}$ ist. Dann gilt:

$$\sum_{r_{(u,v)} \in H(f)} \rho(u,v) = \begin{cases} +4 & f \text{ Innenfläche} \\ -4 & f \text{ Außenfläche} \end{cases} \tag{3.2}$$

für alle Flächen $f \in F$.

(Kr4) *Für jeden Knoten $v \in V$ ist die Summe der Winkel um v herum, die durch die α-Felder in $\mathcal{H}$ gegeben ist, gleich $360°$.*

Die 2. Bedingung (Kr2) gewährleistet, dass die Knicke auf einer Kante (u,v) in den zwei Bitfolgen $s_{(u,v)}$ und $s_{(v,u)}$ konsistent beschrieben werden.

Die 3. Bedingung (Kr3) stellt sicher, dass jede durch $\mathcal{H}$ beschriebene Fläche ein geradliniges Polygon ist. Das folgt aus dem folgenden Lemma:

Lemma 3.1 *Die Summe der Winkel innerhalb eines Polygons beträgt $(2p-4)\cdot 90°$, die Summe der Winkel außerhalb eines polygons $(2p+4)\,90°$, wobei p die Anzahl der Winkel ist.*

Dabei ist zu beachten, dass es bei orthogonalen Zeichnungen zwei Arten von Winkeln gibt: *Knotenwinkel* und *Knickwinkel*. Daher ist

$$p = |f| + \sum_{r_{(u,v)} \in H(f)} \left|s_{(u,v)}\right| \tag{3.3}$$

wobei $|f| = |H(f)|$ der Anzahl der Knotenwinkel in f entspricht und $\left|s_{(u,v)}\right|$ die Länge der Bitfolge $s_{(u,v)}$ und somit die Anzahl der Knicke angibt.

Man beachte, dass eine orthogonale Repräsentation nur die Form einer Zeichnung beschreibt: zwei unterschiedliche Zeichnungen, die aufgrund ihrer äquivalenten Form die gleiche orthogonale Darstellung besitzen, müssen *nicht unbedingt* die gleichen Längen von Kantensegmenten oder die gleiche Reihenfolge der Kantenknicke aufweisen. Sie haben aber die gleichen Winkel in jedem Knoten und die gleiche Anzahl von Knicken. Dabei ist die Anzahl der Knicke in einer orthogonalen Darstellung $\mathcal{H}$ von $G = (V, E, F)$ durch

$$\text{bends}(\mathcal{H}) = \frac{1}{2} \sum_{f \in F} \sum_{r_{(u,v)} \in H(f)} \left| s_{(u,v)} \right| \tag{3.4}$$

gegeben. Da jede Kante in E genau zwei Mal in $\mathcal{H}$ vorkommt, werden in dieser Summe die Knicke auf jeder Kante doppelt gezählt; daher wird die Hälfte der Gesamtsumme genommen.

3.2 Tamassia's Knickminimierungsalgorithmus

In dem von Tamassia in [Tam87] vorgestellten Algorithmus werden erstmals Min-Cost Flow Algorithmen (Abschn. 2.2) zur Berechnung knickminimaler orthogonaler Darstellungen verwendet. Ausgehend von der gegebenen planaren Darstellung P eines 4-Graphen G wird ein Netzwerk $\mathcal{N}^T(P)$ aufgestellt, sodass aus einem kostenminimalen Fluss auf $\mathcal{N}^T(P)$, eine regionerhaltende, knickminimale orthogonale Darstellung für G abgeleitet werden kann.

3.2.1 Das Tamassia Netzwerk

Sei im folgenden $G = (V, E, F)$ ein planarer 4-Graph mit gegebener Einbettung P. Bei der Aufstellung des Netzwerkes $\mathcal{N}^T(P) = \left(N^T, A^T\right)$ wird zunächst für jeden Knoten in V und für jede Fläche in F jeweils ein Knoten in das Netzwerk eingeführt:

$$N^T = N_V \cup N_F$$

wobei

- $N_V = \{n_v \mid v \in V\} \quad b(n_v) = 4 - \deg(v) \quad \forall n_v \in N_V$
- $N_F = \{n_f \mid f \in F\} \quad b(n_f) = \begin{cases} -(|f| - 4) & f \text{ Innenfläche} \\ -(|f| + 4) & f \text{ Außenfläche} \end{cases}$

Die Kanten des Netzwerkes sind wie folgt zusammengesetzt:

$$A^T = A_{VF} \cup A_{FF}$$

wobei

- $A_{VF} = \{a_{u,v}^{VF} = (n_v, n_f) \quad \mid f = \text{face}(u,v),\ (u,v) \in \bar{E}\}$
  $\text{lcap}(a) = 0,\ \text{ucap}(a) = 3,\ \text{cost}(a) = 0 \quad \forall a \in A_{VF}$

- $A_{FF} = \{a_{u,v}^{FF} = (n_f, n_g) \quad \mid f = \text{face}(u,v),$
  $g = \text{face}(v,u),\ (u,v), (v,u) \in \bar{E}\}$
  $\text{lcap}(a) = 0,\ \text{lcap}(a) = \infty,\ \text{cost}(a) = 1 \quad \forall a \in A_{FF}$

Um die Formulierungen zu vereinfachen werden wir im folgenden die Knoten in N_V als *Knoten* und die Knoten in N_F als *Flächen* bezeichnen. Weiters bezeichnen wir Kanten aus A_{FF} (bzw. A_{VF}) als $f - f$ Kanten (bzw. $v - f$ Kanten). In den folgenden Abbildungen sind *Knoten* durch Quadrate, *Flächen* durch Kreise dargestellt.

Die Transformation eines Flusses x auf $\mathcal{N}^T(P)$ in eine orthogonale Darstellung $\mathcal{H}$ erfolgt auf der Basis folgender Überlegungen:

1. Jede Flusseinheit im Netzwerk repräsentiert einen Winkel von 90°:

   - Der Fluss auf einer Kante $a_{u,v}^{VF} = (n_v, n_f) \in A_{VF}$ repräsentiert den Winkel, den die Richtungskante (u,v) und ihr Nachfolger auf der Fläche f formen. Dieser ist gegeben durch $\left(x\left(a_{u,v}^{VF}\right) + 1\right) \cdot 90°$ (Abb. **??**).
   - Der Fluss auf einer Kante $a_{u,v}^{FF} = (n_f, n_g) \in A_{FF}$ repräsentiert die Anzahl der konvexen Knicke in der Fläche f, die entlang der

Richtungskante (u, v) auftreten. Jede Einheit, die von n_f nach n_g fließt, formt einen 90°-Knick auf der Fläche f (Abb. 3.4). Man beachte, dass dabei zugleich ein konkaver Knick auf der gegenüberliegenden Fläche g geformt wird.

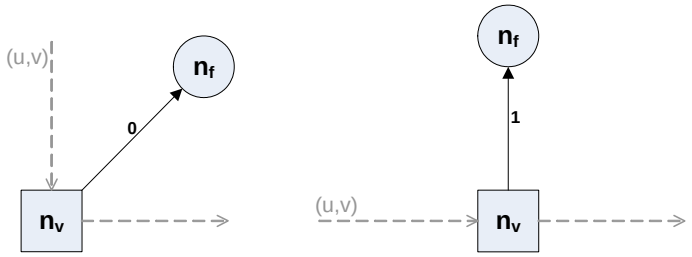

Abbildung 3.3: Der Fluss von einem *Knoten* in eine *Fläche* beschreibt den Winkel, der an dem entsprechenden Knoten in dieser Fläche geformt wird.

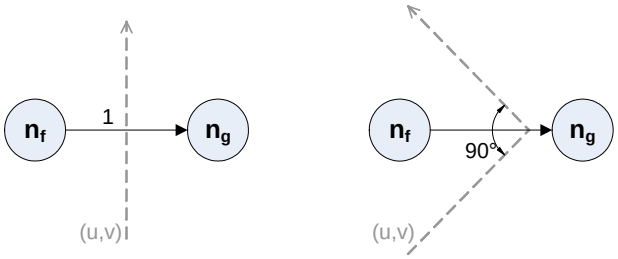

Abbildung 3.4: Der Fluss von einer *Fläche* in eine benachbarte *Fläche* formt einen konvexen Knick auf der Ausgangsfläche.

2. Die Flusserhaltungsbedingung an den *Knoten* $n_v \in N_V$ entspricht der Bedingung Kr4 (Kriterium 3.1) und stellt somit sicher, dass die Summe aller Winkel um $v \in V$ gleich 360° beträgt:

$$\begin{aligned} & \sum_{(u,v)\in \mathrm{In}(v)} \left(x\left(a_{u,v}^{VF}\right) + 1\right) \cdot 90 \\ = & \left(\deg(v) + \sum_{(u,v)\in \mathrm{In}(v)} x\left(a_{u,v}^{VF}\right)\right) \cdot 90 \\ = & \left(\deg(v) + 4 - \deg(v)\right) \cdot 90 \\ = & \ 360 \end{aligned}$$

3. Durch die Flusserhaltung an den Knoten $n_f \in N_F$ ist gewährleistet, dass jede Fläche als ein geradliniges Polygon dargestellt wird (Kr3, Kriterium 3.1): Die Summe der Winkel innerhalb einer Innenfläche f ergibt sich aus dem Fluss durch

$$\sum_{r(u,v)\in H(f)} \left(\left(x\left(a_{u,v}^{VF}\right)+1\right)\cdot 90 + x\left(a_{u,v}^{FF}\right)\cdot 90 + x\left(a_{v,u}^{FF}\right)\cdot 270\right).$$

Diese Summe muss gleich $(2p-4)\cdot 90$ ergeben. Lässt man zur Vereinfachung den $90°-$ Faktor weg, erhält man unter Verwendung der Gleichung 3.3 (S. 25)

$$\begin{aligned}
& |f| + \sum_{r(u,v)\in H(f)} \left(x\left(a_{u,v}^{VF}\right) + x\left(a_{u,v}^{FF}\right) + 3\cdot x\left(a_{v,u}^{FF}\right)\right) \\
= \; & |f| + \sum_{r(u,v)\in H(f)} \left(x\left(a_{u,v}^{VF}\right) + x\left(a_{v,u}^{FF}\right) - x\left(a_{u,v}^{FF}\right)\right) \\
& \qquad +2\cdot \sum_{r(u,v)\in H(f)} \left(x\left(a_{v,u}^{FF}\right) + x\left(a_{u,v}^{FF}\right)\right) \\
= \; & |f| + (|f|-4) + 2\cdot \sum_{r(u,v)\in H(f)} \left(\left|s_{(u,v)}\right|\right) \\
= \; & 2p-4
\end{aligned}$$

4. Die Kosten des Flusses x sind gleich der Anzahl der Knicke in $\mathcal{H}$:

$$\sum_{a\in A_{FF}} x(a) = \sum_{(u,v)\in \bar{E}} x\left(a_{u,v}^{FF}\right) = \sum_{(u,v)\in \bar{E}} \left|s_{(u,v)}\right|_0 = \text{bends}(\mathcal{H})$$

Man beachte, dass die Summe der konvexen Knicke über alle Richtungskanten die Anzahl aller Knicke ergibt. Die konkaven Knicke auf einer Richtungskante $(u,v)\in\bar{E}$ entsprechen konvexen Knicken auf der Richtungskante $(v,u)\in\bar{E}$ und sind als solche in der Summe enthalten.

Es kann nun gezeigt werden, dass für eine auf diese Weise definierte Beziehung zwischen Fluss und Darstellung, folgendes gilt:

Lemma 3.2 *[Tam87] Für jede orthogonale Darstellung $\mathcal{H}$ von G gibt es*

einen ganzzahligen Fluss x auf $\mathcal{N}^T(P)$, dessen Kosten gleich der Anzahl der Knicke in $\mathcal{H}$ ist.

Lemma 3.3 *[Tam87] Für jeden ganzzahligen Fluss x auf $\mathcal{N}^K(P)$, gibt es eine entsprechende orthogonale Darstellung $\mathcal{H}$, dessen Anzahl der Knicke gleich den Kosten des Flusses x ist. Weiters kann diese Darstellung aus dem Fluss x wie folgt abgeleitet werden:*

- Für jedes $(u,v) \in \bar{E}$:

$$\alpha_{(u,v)} = \left(x(a^{VF}_{u,v}) + 1\right) \cdot 90 \tag{3.5}$$

- Für jedes Paar $(u,v),\ (v,u) \in \bar{E}$:

$$s_{(u,v)} = 1^{x\left(a^{FF}_{v,u}\right)} 0^{x\left(a^{FF}_{u,v}\right)}$$

$$s_{(v,u)} = 1^{x\left(a^{FF}_{u,v}\right)} 0^{x\left(a^{FF}_{v,u}\right)}$$

Die folgende, grundlegende Aussage folgt direkt aus diesen zwei Lemmata und der Tatsache, dass auf einem Netzwerk mit ganzzahligen Kapazitäten ein kostenminimaler ganzzahliger Fluss existiert [AMO93]:

Satz 3 *[Tam87] Die minimale Anzahl von Knicken in einer orthogonalen Darstellung von G, ist gleich den minimalen Kosten eines gültigen Flusses auf $\mathcal{N}^T(P)$. Weiters kann jede knickminimale orthogonale Darstellung $\mathcal{H}$ von G, aus einem optimalen Fluss auf $\mathcal{N}^T(P)$ abgeleitet werden.*

Somit ist die Berechnung einer knickminimalen orthogonalen Zeichnung auf die Berechnung eines Min-Cost Flows zurückgeführt worden, was unter anderem eine polynomiale Laufzeit ermöglicht.

Der Netzwerk-Fluss basierte Ansatz bietet außerdem viele Erweiterungsmöglichkeiten. Zusätzliche Anforderungen an die Zeichnung können mit Hilfe entsprechender Nebenbedingungen an den Fluss oder entsprechender Modifikationen im Netzwerk realisiert werden.

Kapitel 4

Kandinsky Zeichnungen

In [FK96] stellen Fößmeier und Kaufmann das Kandinsky Modell vor, in dem auch Graphen mit höherem Knotengrad orthogonal gezeichnet werden können. In diesem Modell werden Knoten als Quadrate gleicher Größe dargestellt, die auf die Eckpunkte eines Knoten-Gitters platziert werden. Die Kanten setzen sich aus vertikalen und horizontalen Liniensegmenten zusammen, die entlang der Linien eines feineren Kanten-Gitters verlaufen (Abb. 4.1). Dadurch können mehrere Kanten von derselben Seite eines Knotens ausgehen. Das Kanten-Gitter ist fein genug, sodass die maximale Anzahl von Kanten, die inzident zu derselben Seite eines Knotens sind, darauf Platz finden.

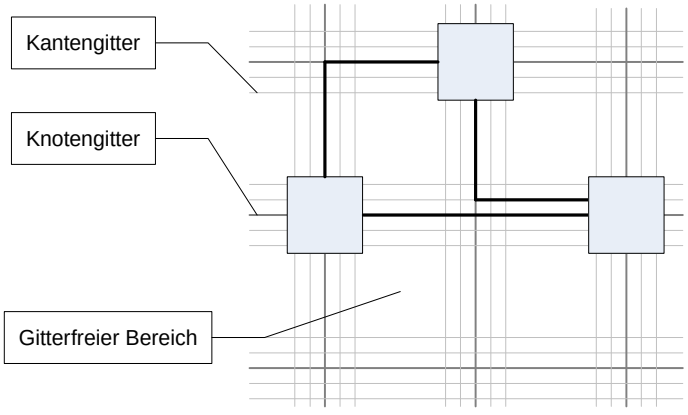

Abbildung 4.1: Knoten- und Kantengitter im Kandinsky Modell

Die Winkel zwischen zwei aufeinanderfolgenden Kanten, die an derselben Seite eines Knotens ansetzen, betrachten wir als 0°-Winkel (siehe Abb. 4.2) .

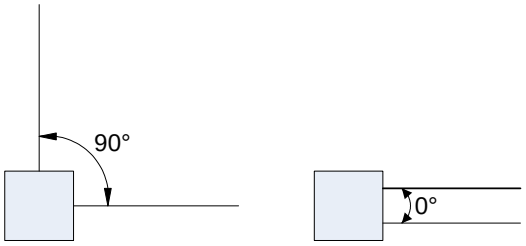

Abbildung 4.2: 0°-Winkel zwischen zwei parallel verlaufenden Kantensegmenten

Zeichnungen im Kandinsky Modell haben noch die zusätzliche Einschränkung, dass sie keine leeren Flächen enthalten dürfen (Abb. 4.3). Eine Fläche wird dabei als nicht leer bezeichnet, wenn sie einen Teil der gitterfreien Bereiche beinhaltet (Abb. 4.1).

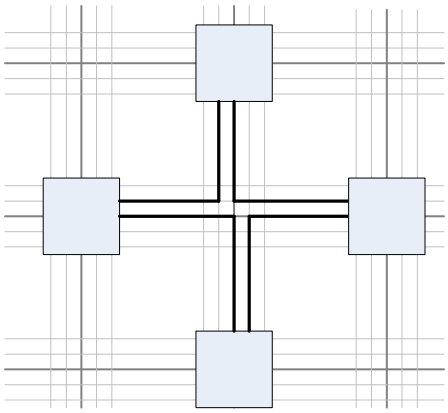

Abbildung 4.3: Eine leere Fläche.

Eine Kandinsky Zeichnung ist demnach durch folgende Eigenschaften gekennzeichnet:

(E1) Die Knoten sind auf den Gitterpunkten des Knotengitters platziert.

(E2) Jedes Kantensegment verläuft entlang der Gitterlinien des Kantengitters.

(E3) Knoten und Kanten schneiden sich nicht, außer Kanten mit ihren Endknoten.

(E4) Die Zeichnung enthält keine leeren Flächen.

Nun gilt es, diese Eigenschaften so zu formulieren, dass sie algorithmisch erfasst und von einem automatisierten Zeichenverfahren realisiert werden können.

4.1 Die Kandinsky Repräsentation

Die Kandinsky Repräsentation erweitert die Orthogonale Repräsentation (Def. 3.1) um den 0°-Winkel.

Definition 4.1 *Eine **Kandinsky Repräsentation** (auch Kandinsky Darstellung) K eines planaren Graphen $G = (V, E, F)$ ist eine Menge von flächenbeschreibenden, zyklischen Listen $K(f)$ $(f \in F)$. Jede Liste $K(f)$ besteht aus Elementen $r_{(u,v)} = ((u, v), s_{(u,v)}, \alpha_{(u,v)})$, wobei*

- (u, v) *eine Richtungskante in* $\bar{E}$ *ist,*
- $\alpha_{(u,v)} \in \{0, 90, 180, 270, 360\}$ *den Winkel zwischen der Kante* (u, v) *und seinem Nachfolger auf der Fläche* f *angibt und*
- $s_{(u,v)}$ *ein Bit-String ist, der die Knicke entlang der Kante* (u, v) *beschreibt: eine* 0 *steht für einen* 90°*-Knick (konvex), eine* 1 *für einen* 270°*-Knick (konkav). Ein leerer String* ε *beschreibt eine gerade Kante.*

Hier stellt sich die Frage, welchen Bedingungen eine solche Menge K von zyklischen Listen genügen muss, damit sie einer korrekten Kandinsky Zeichnung entspricht.

Orthogonalität

Die Kriterien *(Kr1)-(Kr4)* (S. 24) gelten auch für Kandinsky Repräsentationen: Sie sind notwendig und hinreichend dafür, dass K eine korrekte, orthogonale Zeichnung beschreibt. Insbesondere stellt das Kriterium *(Kr3)* auch

hier sicher, dass Flächen als geradlinige Polygone dargestellt werden, da die Definition des 0°-Winkels konsistent mit dem Lemma 3.1 ist. Das ist leicht einzusehen, wenn wir 0°-Winkel als zwei aufeinanderfolgende 90°-Winkel betrachten (siehe Abb. 4.4). Ersetzen wir nun die zwei 90°-Winkel durch einen

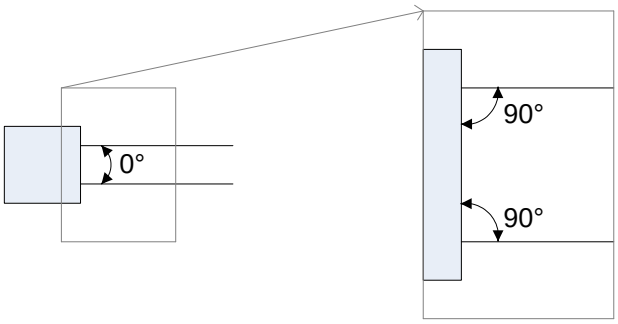

Abbildung 4.4: Ein 0°-Winkel kann als zwei aufeinanderfolgende 90°-Winkel innerhalb der Fläche betrachtet werden.

0°-Winkel, verringert sich die Summe der Winkel innerhalb der Fläche um 180°. Dabei wird die Anzahl der Winkel um 1 verringert, wodurch die Summe $(2p-4)\cdot 90°$ sich ebenfalls um 180° verringert (p ist die Anzahl der Winkel innerhalb der Fläche). Die Gleichung

$$(2p-4)\cdot 90° = \sum (\text{Winkel innerhalb Polygon}) \tag{4.1}$$

wird also durch das Einführen von 0°-Winkeln nicht zerstört und gilt daher auch für Kandinsky-Flächen (dasselbe gilt analog für die Außenfläche).

Nicht leere Flächen

Sei K eine Darstellung des Graphen $G = (V, E, F)$, die die Eigenschaften *E1-E3* (S. 32) erfüllt.

Lemma 4.1 *Jede Fläche f mit $|K(f)| \geq 4$ kann als nicht leere Fläche gezeichnet werden.*

Beweis. [FK96] Hat einer der Knotenwinkel in f einen positiven Wert (90, 180, 270 oder 360), dann kann die Fläche auf jeden Fall nicht leer gezeichnet

werden. Betrachten wir also nur solche Flächen, in denen alle Knotenwinkel $0°$ betragen. Ist $|f| = 4$, können die vier Knoten so platziert werden, dass sie paarweise verschiedene x- und y-Koordinaten haben. Das führt zu einer Zeichnung mit nicht leerer Fläche (siehe Abb. 4.6, (c)). Sei $|f| > 4$ und sei b_{270} die Anzahl der konkaven Knicke in f. Nehmen wir vorerst an, dass die Anzahl b_{90} der konvexen Knicke in f gleich Null ist. Dann gilt aufgrund von Lemma 3.1:

$$\begin{aligned} 2 \cdot (|f| + b_{270} - 2) \cdot 90 &= b_{270} \cdot 270 \\ 2 \cdot (|f| + b_{270} - 2) &= 3 \cdot b_{270} \\ 2\,|f| - 4 &= b_{270}. \end{aligned}$$

Für Flächen f mit $|f| > 4$ ist also $b_{270} > |f|$. Mindestens eines der zu f adjazenten Kanten weist daher zwei aufeinanderfolgende 270°-Knicke auf. Das Kantensegment zwischen diesen beiden Knicken (siehe Abb. 4.5) kann beliebig verschoben und die Fläche dadurch nicht leer gezeichnet werden. Jeder zusätzliche 90°-Knick in der Fläche existiert mit einen zugehörigen, zusätzlichen 270°-Knick und ändert das Ergebnis daher nicht. ■

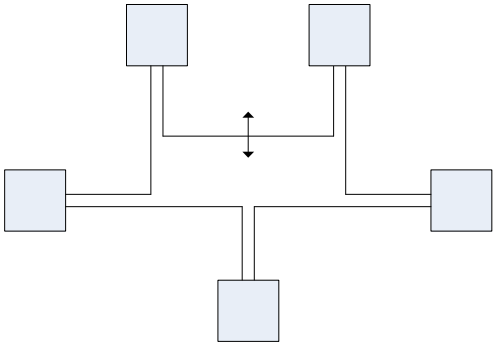

Abbildung 4.5: Jede Fläche f mit $|f| > 4$ kann nicht leer gezeichnet werden.

Es ist folglich ausreichend, Flächen der Größe drei zu betrachten. Das sogenannte T-Dreieck (Abb. 4.6, (a)) ist die einzige Darstellung einer 3-

Fläche, die nicht als nicht leere Fläche gezeichnet werden kann:

$$K(f) = (((u, v), \varepsilon, 0),\ ((v, w), 1, 0),\ ((w, u), 1, 0)).$$

Die Eigenschaft *E4* ist somit durch das Ausschließen von T-Dreiecken erfüllt.

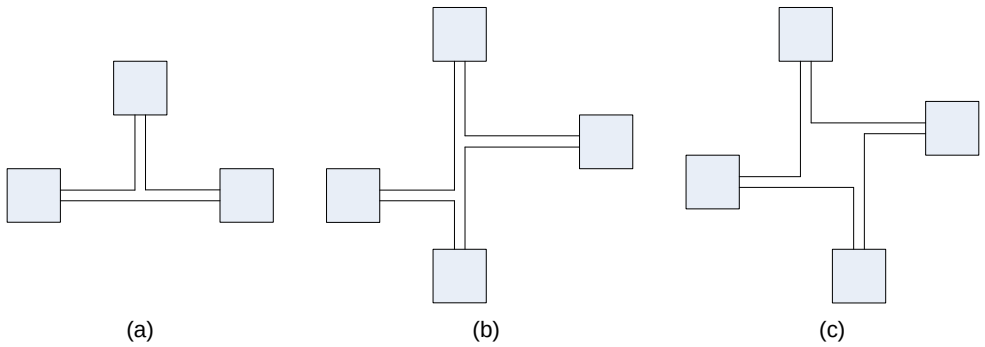

Abbildung 4.6: Ein T-Dreieck (a) kann nicht als nicht leere Fläche gezeichnet werden. Eine Fläche der Grösse 4 hingegen schon.

Bend-or-End

Es muss sichergestellt werden, dass von zwei Kanten, die von derselben Seite eines Knotens ausgehen, mindestens eine einen Knick nach außen, also einen konkaven Knick macht. Es können nicht beide Kanten gleichzeitig gerade verlaufen, weil das zu Überlappungen (Verletzung der Eigenschaft *E3*) und somit zu einer inkorrekten Zeichnung führen würde (Abb. 4.7).

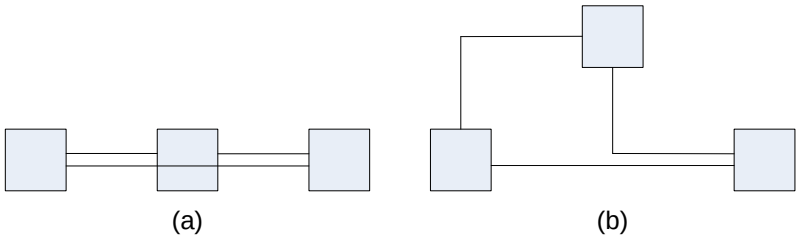

Abbildung 4.7: Ungültige (a) und gültige (b) Zeichnung im Kandinsky Modell.

Sind $r_{(u,v)}$ und $r_{(v,w)}$ zwei aufeinanderfolgende Elemente in $K(f)$ und ist $\alpha_{(u,v)} = 0$, muss daher entweder der letzte Knick auf der Richtungskante (u, v) oder der erste Knick auf der Richtungskante (v, w) ein konkaver Knick sein. Diese Eigenschaft wird auch als die *Bend-or-End* Eigenschaft bezeichnet.

Eine hinreichende Bedingung

Die bisher genannten Bedingungen reichen jedoch nicht aus, um eine korrekte Kandinsky Repräsentation zu garantieren. Ein Beispiel dafür ist die Darstellung einer 3-Fläche als L-Dreieck (Abb. 4.8,(a)):

$$K(f) = (((u, v), \varepsilon, 0),\ ((v, w), 1, 0),\ ((w, u), \varepsilon, 90)).$$

Sie erfüllt die Kriterien *(Kr1)-(Kr4)*, besitzt die Bend-or-End Eigenschaft und ist kein T-Dreieck. Es kann dennoch nicht ohne Überlappungen gezeichnet werden.

Es sei hier noch erwähnt, dass das Ausschließen von T- und L-Dreiecken, zusammen mit der Bend-or-End Eigenschaft ebenfalls keine korrekte Kandinsky Darstellung garantiert, da immer noch unkorrekte Darstellungen von größeren Flächen wie in Abb. 4.8,(b) möglich wären.[1]

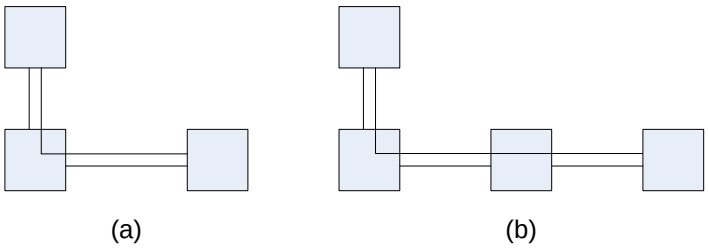

Abbildung 4.8: Beide Darstellungen können im Kandinsky Modell nicht korrekt gezeichnet werden.

Erst das folgende Lemma ermöglicht die Formulierung einer hinreichenden Bedingung:

[1] Das wird hier speziell deswegen erwähnt, weil Def. 4.8 in [Eig03] besagt, dass diese Eigenschaften ausreichend sind.

Lemma 4.2 *Jeder 0°-Winkel in einer Kandinsky Zeichnung kann eindeutig einem 270°-Knick zugeordnet werden.*

Beweis. [FK96] ■

Satz 4 *Eine Menge K von zyklischen Listen $K(f)$ ist eine gültige Kandinsky Repräsentation für den planaren Graphen $G = (V, E, F)$, wenn es zusätzlich zu den Kriterien (Kr1)-(Kr4) (S. 24), das folgende Kriterium erfüllt:*

(Kr5) Jedem 0°-Winkel in $K(f) \in K$ ist eindeutig ein 270°-Knick in f zuordenbar, der außerdem (von dem 0°-Winkel aus gesehen) als erster Knick auf einer der zwei Kanten auftritt, die den 0°-Winkel erzeugen.

Intuitiv läßt sich das folgendermaßen begründen: Sei K eine Menge von zyklischen Listen $K(f)$, die alle Kriterien (Kr1)-(Kr5) erfüllt.
Ein T-Dreieck genügt nicht dem Kriterium (Kr5) und ist daher bestimmt nicht in K enthalten.
Bleibt zu zeigen, dass K ohne Überlappungen gezeichnet werden kann. Überlappungen entstehen nur in Zusammenhang mit 0°-Winkeln: Verläuft eine

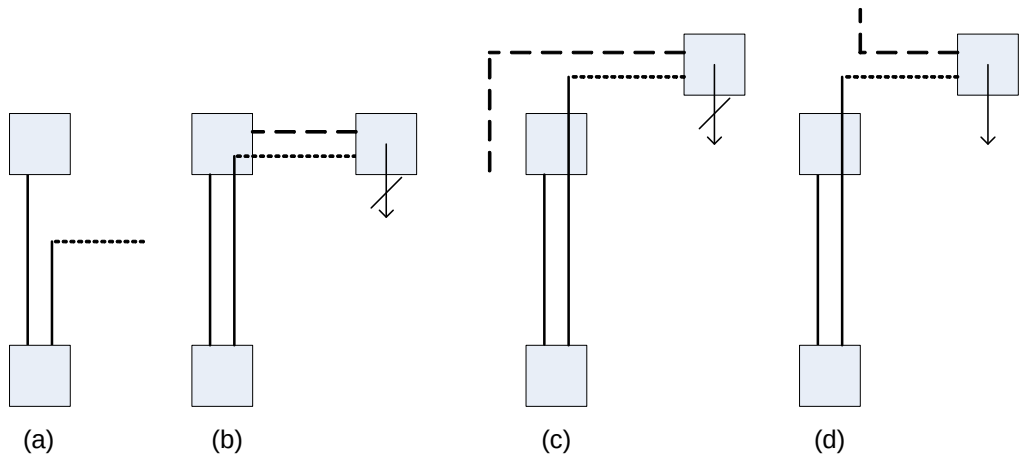

Abbildung 4.9: Von zwei Kanten, die einen 0°-Winkel formen (a), muss die kürzere rechtzeitig nach außen knicken, um Überkreuzungen zu vermeiden. Ist jedem 0°-Winkel eindeutig ein zugehöriger konkaver Knick zuordenbar, ist das immer möglich (d).

von zwei Kanten, die an dieselbe Seite eines Knotens ansetzen, gerade und endet in einem Knoten, dann muss die andere Kante *rechtzeitig* nach außen knicken (Abb. 4.9,(a)) , d.h. bevor diese den Endknoten der geraden Kante berührt, sonst kommt es zu einer Überkreuzung.

Konnte die Kante nicht rechtzeitig abbiegen und berührt den Knoten, dann gilt:

- Das Kantensegment nach dem konkaven Knick (in Abb. 4.9,(b) punktiert dargestellt) läuft geradeaus und endet in einem Knoten. Käme hier ein weiterer Knick, könnte das punktiert dargestellte Kantensegment zwischen zwei Knickpunkten frei verschoben und eine Überkreuzung verhindert werden.

- Das punktiert dargestellte Kantensegment formt mit seinem Nachfolger (in Abb. 4.9,(b) gestrichelt dargestellt) einen 0°-Winkel. Wäre der Winkel positiv, könnte das punktierte Kantensegment mitsamt dem Knoten verschoben und eine Überkreuzung verhindert werden.

- Die gestrichelt dargestellte Kante verläuft entweder geradeaus und endet in einem Knoten oder die gestrichelte Kante macht einen konvexen Knick, sodass der Ausgangsknoten nicht verschoben werden kann (Abb. 4.9,(b),(c)).

In beiden Fällen ist das Problem, dass der Knotenknick zugleich von einem anderen 0°-Winkel *benötigt* wird. Ist jedoch jedem 0°-Winkel ein eigener konkaver Knick zugeordnet, können sie frei platziert werden, sodass Kreuzungen durch geeignete Kompaktierung vermieden werden können (Abb. 4.9,(d)).

Sind $r_{(u,v)}$ und $r_{(v,w)}$ zwei aufeinanderfolgende Elemente in $K(f)$ und ist $\alpha_{(u,v)} = 0$, ist daher entweder der letzte Knick auf der Richtungskante (u, v) oder der erste Knick auf der Richtungskante (v, w) ein konkaver Knick *und* kann eindeutig dem Winkel $\alpha_{(u,v)}$ zugeordnet werden. Solche Knicke, die aufgrund von 0°-Winkeln auftreten, werden auch als **Knotenknicke** bezeichnet.

Um eine Unterscheidung von Knotenknicken und gewöhnlichen Kantenknicken innerhalb einer Kandinsky Darstellung zu ermöglichen, betrachten wir den Bit-String $s_{(u,v)}$ im weiteren (wenn erforderlich) als die Vereinigung von drei Teilstrings: $s_{(u,v)} = s^A_{(u,v)} \cup s^M_{(u,v)} \cup s^E_{(u,v)}$. Der String $s^M_{(u,v)}$ beschreibt die Kantenknicke auf der Kante (u, v). $s^A_{(u,v)}$ (bzw. $s^E_{(u,v)}$) ist ein Bit-String der Länge 1 und steht für einen Knotenknick am Anfang (bzw. am Ende)

der Richtungskante (u, v).

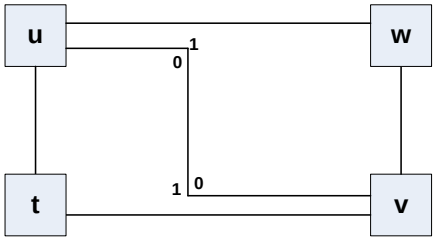

Abbildung 4.10: Beide Knicke auf der Kante (u, v) sind Knotenknicke.

Betrachten wir als Beispiel die Kante (u, v) in Abb. 4.10. Beide Knicke auf dieser Kante sind Knotenknicke. Die Bit-Strings für die Richtungskanten (u, v) und (v, u) sehen in dem Fall folgendermassen aus:

$$\begin{aligned} s_{(u,v)} &= 10,\ s_{(v,u)} = 10 \\ s^A_{(u,v)} &= 1,\ s^M_{(u,v)} = \varepsilon,\ s^E_{(u,v)} = 0 \\ s^A_{(v,u)} &= 1,\ s^M_{(u,v)} = \varepsilon,\ s^E_{(u,v)} = 0 \end{aligned}$$

Die Einschränkung auf nicht-leere Flächen ist nicht unbedingt notwendig oder wünschenswert. Möchte man leere Flächen erlauben, muss man die Bedingung *(Kr5)* fallen lassen, da diese T-Dreiecke verbietet. Es stellt sich allerdings als nicht so einfach heraus, eine Bedingung zu formulieren, die inkorrekte Darstellungen mit Überlappungen verhindert und zugleich T-Dreiecke erlaubt. Außerdem wird durch das Weglassen von *(Kr5) ausschließlich* die Darstellung von 3-Flächen als T-Dreiecke gewonnen, da korrekte Darstellungen von k-Flächen mit $k > 3$ dieses Kriterium in jedem Fall erfüllen, unabhängig davon, ob sie leer gezeichnet werden können oder nicht (siehe Beweis zu Satz 4.2).

Das Kandinsky Modell hat dazu noch den großen Vorteil, dass die Berechnung einer knickminimalen Kandinsky Repräsentation auf ein Netzwerk-Fluss Problem mit Nebenbedingungen zurückgeführt werden kann. Das wird dadurch ermöglicht, dass die Bedingung *(Kr5)* relativ einfach als Netzwerk-Fluss Bedingung formuliert werden kann.

4.2 Knickminimierung bei Kandinsky Zeichnungen

In diesem Abschnitt stellen wir das von Fößmeier in [Foe97] eingeführte Kandinsky Netzwerk $\mathcal{N}^K$ vor, das als eine Erweiterung des Tamassia Netzwerkes betrachtet werden kann, sodass ein kostenminimaler Fluss auf diesem Netzwerk einer knickminimalen Kandinsky Repräsentation entspricht. In [EFK00] wird dieser Min-Cost Flow basierte Ansatz als ILP formuliert, der auf einen alternativen Netzwerk aufbaut. Dieses stellen wir hier unter dem Namen *Alternativ Kandinsky Netzwerk* $\mathcal{N}^{AK}$ vor.

4.2.1 Das Kandinsky Netzwerk

Sei im folgenden $G = (V, E, F)$ ein planarer Graph mit gegebener Einbettung P. Bei der Aufstellung des Kandinsky Netzwerks $\mathcal{N}^K(P) = \left(N^K, A^K\right)$ wird von dem Tamassia Netzwerk (Abschn. 3.2.1) ausgegangen. Die Knoten und Kanten von $\mathcal{N}^K(P)$ setzen sich zunächst wie folgt zusammen:

$$\begin{aligned} N^K &= N_V \cup N_F \\ A^K &= A_{VF} \cup A_{FF} \end{aligned}$$

wobei die Mengen N_V, N_F, A_{VF} und A_{FF} genau wie in Abschn. 3.2.1 definiert sind.

Eine Kandinsky Darstellung hat gegenüber einer orthogonalen Darstellung zwei wesentliche Unterschiede, die im Netzwerk zusätzlich berücksichtigt werden müssen: Es enthält 0°-Winkel und es muss dem Kriterium (*Kr5*) genügen.

0°-Winkel

Der Winkel, den eine Richtungskante (u, v) mit ihrem Nachfolger auf einer Fläche f formt, ist gegeben durch

$$(x(n_v, n_f) + 1) \cdot 90°.$$

Demnach entspricht ein 0°-Winkel einem Fluss von -1 Einheiten auf der Kante $a_{u,v}^{VF} = (n_v, n_f)$. Dieser wird als ein Fluss von $+1$ Einheiten in die entgegengesetzte Richtung interpretiert. Um 0°-Winkel modellieren zu können, müssen also neue Kanten von *Flächen* nach *Knoten* eingeführt werden: (n_f, n_v). Die obere Kapazitätsgrenze solcher $f - v$ Kanten wird auf 1 gesetzt, da ein größerer Fluss einem negativen Winkel entsprechen würde.

Der von der Richtungskante (u, v) und ihrem Nachfolger auf der Fläche f geformte Winkel ist dann gegeben durch

$$(x(n_v, n_f) - x(n_f, n_v) + 1) \cdot 90°$$

Für einen Knoten v mit $\deg(v) > 4$ hat der zugehörige *Knoten* n_v im Netzwerk einen negativen Bedarf: $b(n_v) = 4 - \deg(v) < 0$. Die $\deg(v) - 4$ Einheiten, die über die $f - v$ Kanten in einen solchen Knoten einfließen, entsprechen den 0°-Winkeln, die um den Knoten v angeordnet sind. In Abb. 4.11 sind zwei mögliche Winkel-Verteilungen um einen Knoten mit Grad 5 dargestellt. Die *Knoten* sind jeweils mit ihrem Bedarf beschriftet. Hellgraue Netzwerkkanten haben Fluss 0, $f - f$ Kanten sind nicht eingezeichnet. Gestrichelte Kanten gehören zum Graph.

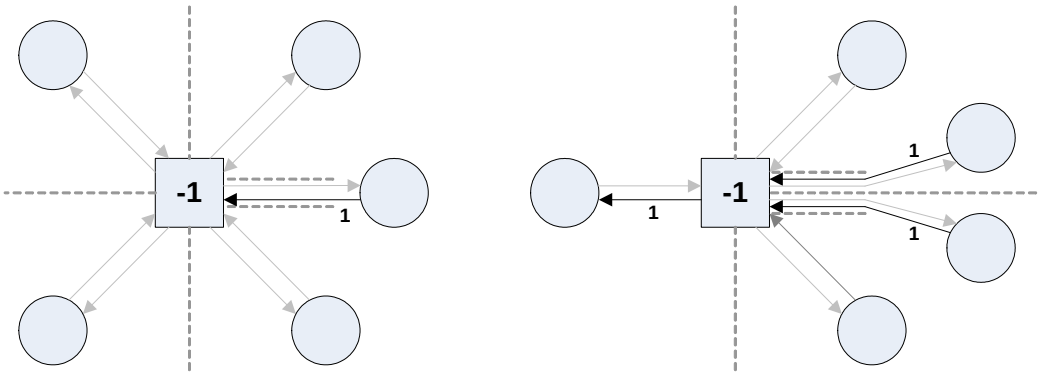

Abbildung 4.11: 0°-Winkel werden im *einfachen* Kandinsky Netzwerk mittels $f - v$ Kanten modelliert.

Das so aufgestellte Netzwerk (Tamassia Netzwerk + $f - v$ Kanten) wird als *einfaches Kandinsky Netzwerk* bezeichnet. Darauf existiert immer ein gül-

tiger Fluss mit 0 Kosten, der zur Erstellung von 2-Sichtbarkeitsdarstellungen verwendet werden kann [Foe97].

Kriterium (*Kr5*)

Damit der Fluss einer korrekten Kandinsky Darstellungen entspricht, muss für jeden 0°-Winkel die Existenz des zugehörigen 270°-Knickes sichergestellt sein.

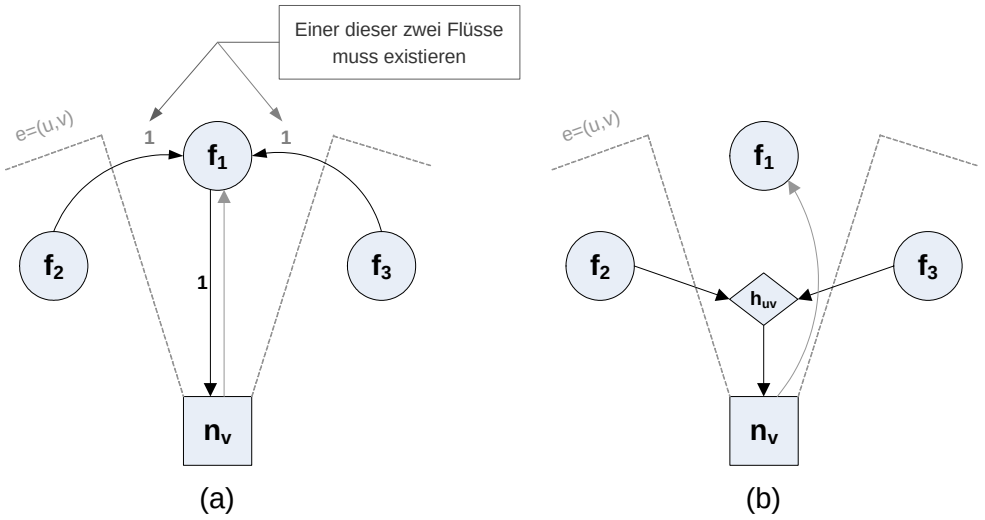

Abbildung 4.12: 0°-Winkel erzwingen Knotenknicke (a), die durch die Hilfskonstruktion modelliert werden (b).

Fließt im Netzwerk eine Einheit von der *Fläche* f_1 in den *Knoten* n_v (Abb. 4.12(a)), entspricht das im Graph einem 0°-Winkel an dem Knoten v, den die Richtungskante (u, v) mit ihrem Nachfolger in der Fläche f_1 formt. In dem Fall muss eine der Kanten, die diesen Winkel formen, nach außen knicken. Das bedeutet, es muss eine Einheit von einem der Nachbar*flächen* f_2 oder f_3 in die *Fläche* f_1 fließen. Dieser Knick muss eindeutig dem 0°-Winkel zugeordnet werden können, der es sozusagen erzwungen hat und muss außerdem von v aus als erster Knick auf der entsprechenden Kante auftreten.

Um das sicherzustellen, wird ein neuer Hilfsknoten $h_{u,v}$ eingeführt. Die Kante (f_1, n_v) wird durch die drei Kanten $\big(f_2, h_{(u,v)}\big)$, $(f_3, h_{u,v})$ und $(h_{u,v}, n_v)$

ersetzt (Abb. 4.12(b)). Ein Fluss auf dem Pfad $(f_2, h_{u,v}, n_v)$ wird als ein Fluss auf dem Pfad (f_2, f_1, n_v) interpretiert. Dadurch wird erreicht, dass ein 0°-Winkel nur mit einem zugehörigen 270°-Knick (Knotenknick) auftritt.

Diese Hilfskonstruktion löst durch die Einführung von Knotenknicken ein Problem, bringt aber ein weiteres mit sich: es können auf einer Richtungskante zwei Knotenknicke in entgegengesetzte Richtungen auftreten.

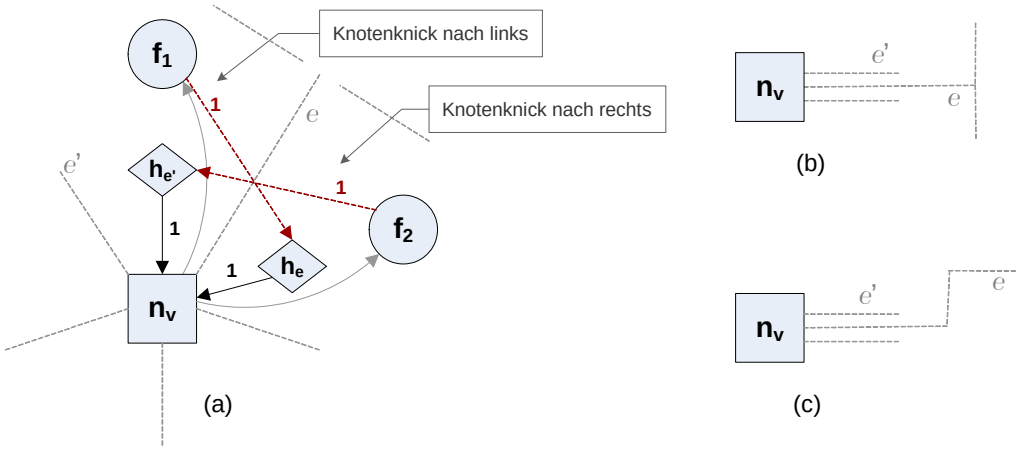

Abbildung 4.13: Um zwei Knotenknicke an demselben Ende einer Kante e in entgegengesetzte Richtungen zu verhindern (a), müssen Bündelkapazitäten gesetzt werden. Ungültige Zeichnungen wie in (b) und (c) werden dadurch verhindert.

Die Flüsse auf den Kanten (f_1, h_e) und $(f_2, h_{e'})$ in Abb. 4.13(a) entsprechen beide einem Knotenknick an dem gleichen Ende der Kante e. Da die zwei Knicke in entgegengesetzte Richtungen nicht beide gleichzeitig als erste Knicke gezeichnet werden können (Abb. 4.13(b)). Bleibt als alternative, einen der Knicke als normalen Kantenknick zu interpretieren und ihn nach dem anderen einzufügen. Das führt aber ebenfalls zu einer ungültigen Zeichnung, da einer der 0°-Winkel keinen zugehörigen Knotenknick besitzt und somit nicht sichergestellt ist, dass z.B. die Kante e' die Kante e nicht überkreuzt (Abb. 4.13(c)).

Um diesen Fall auszuschließen müssen wir verhindern, dass durch beide Kanten eines solchen Paares gleichzeitig eine Einheit fließt. Diese Beschränkung lässt sich leider nicht innerhalb des Netzwerk-Fluss Modells realisieren.

Wir fassen daher zwei solche Kanten, deren beider Fluss einen Knotenknick an dem gleichen Ende einer Kante in G repräsentiert zu einem *Bündel* zusammen und setzen die Kapazität des Bündels auf 1.

Das so aufgebaute Netzwerk können wir nun wie folgt zusammenfassen:

Definition 4.2 ***Das Kandinsky Netzwerk*** *Sei $G = (V, E, F)$ ein planarer Graph mit fixer Einbettung P. Die Menge N^K der Knoten des Kandinsky Netzwerkes $\mathcal{N}^K(P) = (N^K, A^K)$ ist wie folgt zusammengesetzt:*

$$N^K = N_V \cup N_F \cup N_H$$

wobei

- $N_V = \{n_v \mid v \in V\}$ $\quad b(n_v) = 4 - \deg(v) \quad \forall n_v \in N_V$
- $N_F = \{n_f \mid f \in F\}$ $\quad b(n_f) = \begin{cases} -(|f| - 4) & f \text{ Innenfläche} \\ -(|f| + 4) & f \text{ Außenfläche} \end{cases}$
- $N_H = \{h_{u,v} \mid (u,v) \in \bar{E}\}$ $\quad b(h_{u,v}) = 0 \quad \forall h_{u,v} \in N_H$

Die Menge A^K der Kanten ist gegeben durch

$$A^K = A_{VF} \cup A_{HV} \cup A_{FF} \cup A_{FH}$$

wobei

- $A_{HV} = \{a^{HV}_{u,v} = (h_{u,v}, n_v) \mid (u,v) \in \bar{E}\}$
  $\text{lcap}(a) = 0,\ \text{ucap}(a) = 1,\ \text{cost}(a) = 0 \quad \forall a \in A_{HV}$

- $A_{VF} = \{a^{VF}_{u,v} = (n_v, n_f) \mid f = \text{face}(u,v) \quad (u,v) \in \bar{E}\}$
  $\text{lcap}(a) = 0,\ \text{ucap}(a) = 3,\ \text{cost}(a) = 0 \quad \forall a \in A_{VF}$

- $A_{FF} = \{a^{FF}_{u,v} = (n_f, n_{f'}) \mid f = \text{face}(u,v),\ f' = \text{face}(v,u) \quad (u,v),(v,u) \in \bar{E}\}$
  $\text{lcap}(a) = 0,\ \text{ucap}(a) = \infty,\ \text{cost}(a) = 1 \quad \forall a \in A_{FF}$

- $A_{FH} = \bigcup_{(u,v)\in\bar{E}} B_{u,v}$

  wobei

  $B_{u,v} = \{a^R_{u,v} = (n_f, h_{w,v}) \mid f = \text{face}(u,v),\ (w,v) = \text{pred}_f(v,u)$
  $(u,v), (w,v) \in \bar{E}\} \cup$
  $\{a^L_{u,v} = (n_{f'}, h_{u,v}) \mid f' = \text{face}(v,u) \quad (v,u) \in \bar{E}\}$

  $\text{lcap}(a) = 0,\ \text{ucap}(a) = 1,\ \text{cost}(a) = 1 \quad \forall a \in A_{FH}$

  $\text{ucap}(B_{u,v}) = 1 \quad \forall (u,v) \in \bar{E}$ *(Bündelkapazität)*

Lemma 4.3 *Sei $G = (V, E, F)$ ein planarer Graph mit fixer Einbettung P. Für jede Kandinsky Repräsentation K von G gibt es einen ganzzahligen Fluss x im Kandinsky Netzwerk $\mathcal{N}^K(P)$, dessen Kosten gleich der Anzahl der Knicke in K ist.*

Beweis. Wir konstruieren nun aus der Darstellung K einen Fluss x mit den geforderten Eigenschaften. Sei im folgenden $(u,v) \in \bar{E}$ und $(v,w) = \text{succ}_f(u,v)$.

▶ Konstruktion des Flusses unter Einhaltung der Kapazitätsschranken:

- $\forall\ a^{VF}_{u,v} \in A_{VF}$

  Jeder 0°-Winkel hat genau einen zugehörigen Knotenknick. $\left|s^A_{(v,w)}\right|_1$ und $\left|s^E_{(u,v)}\right|_1$ sind Knotenknicke, die dem Winkel $\alpha_{(u,v)}$ zugeordnet sind. Demnach ist mindestens einer ein leerer String. Daher gilt: $\left|s^A_{(v,w)}\right|_1 + \left|s^E_{(u,v)}\right|_1 \leq 1$

- $\forall\ a^{HV}_{u,v} \in A_{HV}$

$$\begin{aligned} x\left(a^{HV}_{u,v}\right) &= \left|s^A_{(v,w)}\right|_1 + \left|s^E_{(u,v)}\right|_1 \\ &\leq 1 = \text{ucap}\left(a^{HV}_{u,v}\right) \quad \checkmark \end{aligned}$$

- $\forall\ a^{FF}_{u,v} \in A_{FF}$

$$\begin{aligned} x\left(a^{FF}_{u,v}\right) &= \left|s^M_{(u,v)}\right|_0 \\ &\leq \infty = \text{ucap}\left(a^{FF}_{u,v}\right) \quad \checkmark \end{aligned}$$

- $\forall\ a^R_{u,v},\ a^L_{u,v} \in A_{FH}$

$$\begin{aligned} x\left(a^R_{u,v}\right) &= \left|s^E_{(u,v)}\right|_0 \leq 1 \\ x\left(a^L_{u,v}\right) &= \left|s^E_{(u,v)}\right|_1 \leq 1 \text{ und} \\ x\left(a^R_{u,v}\right) + x\left(a^L_{u,v}\right) &= \left|s^E_{(u,v)}\right|_0 + \left|s^E_{(u,v)}\right|_1 \\ &\leq 1 = \mathrm{ucap}\left(B_{u,v}\right) \quad \checkmark \end{aligned}$$

$s^E_{(u,v)}$ ist ein Bit-String der Länge 1 und kann entweder 1 oder 0 sein, nicht beides gleichzeitig. Daher gilt: $\left|s^E_{(u,v)}\right|_0 + \left|s^E_{(u,v)}\right|_1 \leq 1$.

▶ x ist ein gültiger Fluss, wenn es zusätzlich die Flusserhaltungsbedingung an allen Knoten erfüllt.

- $\forall n_v \in N_V$:

$$\begin{aligned} & \sum_{(u,v)\in \mathrm{In}(v)} \left(x\left(a^{VF}_{u,v}\right) - x\left(a^{HV}_{u,v}\right)\right) \qquad (4.2) \\ = & \sum_{(u,v)\in \mathrm{In}(v)} \left(\frac{\alpha_{(u,v)}}{90} + \left|s^A_{(v,w)}\right|_1 + \left|s^E_{(u,v)}\right|_1 - 1 - \left|s^A_{(v,w)}\right|_1 + \left|s^E_{(u,v)}\right|_1\right) \\ = & \sum_{(u,v)\in \mathrm{In}(v)} \frac{\alpha_{(u,v)}}{90} - \sum_{(u,v)\in \mathrm{In}(v)} 1 \\ = & \ 4 - \deg(v) = b\left(n_v\right) \ \checkmark \end{aligned}$$

wobei $\mathrm{In}(v) \subset \bar{E}$ die Menge aller Richtungskanten ist, die im Knoten v münden.

- $\forall n_f \in N_F$:

$$\begin{aligned} & \sum_{r_{(u,v)}\in K(f)} \left(x\left(a^{FF}_{u,v}\right) + x\left(a^R_{u,v}\right) + x\left(a^L_{v,u}\right) - x\left(a^{FF}_{v,u}\right) - x\left(a^{VF}_{u,v}\right)\right) \qquad (4.3) \\ = & \sum_{r_{(u,v)}\in K(f)} \Big(\left|s^M_{(u,v)}\right|_0 + \left|s^E_{(u,v)}\right|_0 + \left|s^A_{(u,v)}\right|_0 - \left|s^M_{(u,v)}\right|_1 \\ & \qquad -\frac{\alpha_{(u,v)}}{90} - \left|s^A_{(v,w)}\right|_1 - \left|s^E_{(u,v)}\right|_1 + 1\Big) \\ = & \sum_{r_{(u,v)}\in K(f)} \Big(\left|s_{(u,v)}\right|_0 - \left|s_{(u,v)}\right|_1 + \left|s^A_{(u,v)}\right|_1 - \left|s^A_{(v,w)}\right|_1 - \frac{\alpha_{(u,v)}}{90} + 1\Big) \qquad (4.4) \end{aligned}$$

$$
\begin{aligned}
&= \overbrace{\sum_{r_{(u,v)} \in K(f)} \left(\left| s_{(u,v)} \right|_0 - \left| s_{(u,v)} \right|_1 - \frac{\alpha_{(u,v)}}{90} \right)}^{=-2|f|+4 \text{ (Kr3)}} \\
&\quad + \overbrace{\sum_{r_{(u,v)} \in K(f)} \left(\left| s^A_{(u,v)} \right|_1 - \left| s^A_{(v,w)} \right|_1 \right)}^{=0} + \sum_{r_{(u,v)} \in K(f)} 1 \\
&= \begin{cases} -2\,|f| + 4 + |f| & f \text{ Innenfläche} \\ -2\,|f| - 4 + |f| & f \text{ Außenfläche} \end{cases} \\
&= \begin{cases} -(|f| - 4) & f \text{ Innenfläche} \\ -(|f| + 4) & f \text{ Außenfläche} \end{cases} = b\,(n_f) \quad \checkmark
\end{aligned}
$$

- $\forall h_{u,v} \in N_H$:

$$
\begin{aligned}
& x\left(a^{HV}_{u,v}\right) - x\left(a^{L}_{u,v}\right) - x\left(a^{R}_{w,v}\right) \qquad (4.5) \\
&= \left| s^A_{(v,w)} \right|_1 + \left| s^E_{(u,v)} \right|_1 - \left| s^E_{(u,v)} \right|_1 - \left| s^E_{(w,v)} \right|_0 \\
&= 0 = b\,(h_{u,v}) \quad \checkmark
\end{aligned}
$$

Man beachte, dass $\left| s^A_{(v,w)} \right|_1 = \left| s^E_{(w,v)} \right|_0$ ist (ein konvexer Knick am Anfang von (v,w) ist zugleich ein konvexer Knick am Ende von (w,v)).

▶ Die Kosten des Flusses x sind gleich der Anzahl der Knicke in K:

$$
\begin{aligned}
\text{cost}\,(x) &= \sum_{(u,v) \in \bar{E}} \left(x\left(a^{FF}_{u,v}\right) + x\left(a^{R}_{u,v}\right) + x\left(a^{L}_{u,v}\right) \right) \\
&= \sum_{(u,v) \in \bar{E}} \left| s^M_{(u,v)} \right|_0 + \left| s^E_{(u,v)} \right|_0 + \left| s^E_{(u,v)} \right|_1 \\
&= \sum_{(u,v) \in \bar{E}} \left| s^M_{(u,v)} \right|_0 + \sum_{(u,v) \in \bar{E}} \left| s^E_{(u,v)} \right|_0 + \sum_{(u,v) \in \bar{E}} \left| s^A_{(v,u)} \right|_0 \\
&= \sum_{(u,v) \in \bar{E}} \left| s_{(u,v)} \right|_0 = \text{bends}\,(K) \quad \checkmark
\end{aligned}
$$

■

Lemma 4.4 *Sei $G = (V, E, F)$ ein planarer Graph mit fixer Einbettung P und $\mathcal{N}^K(P)$ das zugehörige Kandinsky Netzwerk. Für jeden ganzzahligen*

Fluss x in $\mathcal{N}^K(P)$ gibt es eine Kandinsky Repräsentation K, deren Anzahl von Knicken gleich den Kosten des Flusses ist. Weiters kann K aus dem Fluss berechnet werden.

Beweis. Wir konstruieren nun die Darstellung K, indem wir für jede Richtungskante $(u,v) \in \bar{E}$ die s und α Felder aus den Flusswerten im Netzwerk berechnen:

- $\forall\ (u,v) \in \bar{E}$:

$$\alpha_{(u,v)} = \left(x(a^{VF}_{u,v}) - x(a^{HV}_{u,v}) + 1\right) \cdot 90$$

- $\forall\ (u,v) \in \bar{E}$:

$$s_{(u,v)} = \overbrace{0^{x(a^L_{v,u})} 1^{x(a^R_{v,u})}}^{s^A_{(u,v)}} \overbrace{1^{x(a^{FF}_{v,u})} 0^{x(a^{FF}_{u,v})}}^{s^M_{(u,v)}} \overbrace{0^{x(a^R_{u,v})} 1^{x(a^L_{u,v})}}^{s^E_{(u,v)}} \tag{4.6}$$

▶ Das so konstruierte K genügt den Kriterien (Kr1)-(Kr5):

- (Kr1) ist automatisch erfüllt, da wir K auf P aufgebaut haben. ✓
- (Kr2) folgt einfach aus der Aufstellung der Bit-Strings (Gl.4.6). ✓
- (Kr3):

$$\begin{aligned}
&\sum_{r_{(u,v)} \in K(f)} \rho\,(u,v) \\
=\ &\sum_{r_{(u,v)} \in K(f)} \left(\left|s_{(u,v)}\right|_0 - \left|s_{(u,v)}\right|_1 + (2 - \frac{\alpha_{(u,v)}}{90}) \right) \\
=\ &\overbrace{\sum_{r_{(u,v)} \in K(f)} \left(x\left(a^L_{v,u}\right) + x\left(a^{FF}_{u,v}\right) + x\left(a^R_{u,v}\right) - x\left(a^{FF}_{v,u}\right) - x(a^{VF}_{u,v})\right)}^{=-(|f|-4)\ \text{(Gl.4.3)}} \\
&+ \sum_{r_{(u,v)} \in K(f)} \left(-x\left(a^L_{u,v}\right) - x\left(a^R_{v,u}\right) + x(a^{HV}_{u,v}) + 2 - 1\right) \\
=\ &-(|f|-4) + |f| + \overbrace{\sum_{r_{(u,v)} \in K(f)} \left(-x\left(a^L_{u,v}\right) - x\left(a^R_{v,u}\right) + x(a^{HV}_{u,v})\right)}^{=0\ \text{(Gl.4.5)}} \\
=\ &4 \text{ (bzw. } -4 \text{ ,wenn } f \text{ Außenfläche)} \ \checkmark
\end{aligned}$$

- (Kr4):

$$\begin{aligned}
& \sum_{(u,v)\in \mathrm{In}(v)} \alpha_{(u,v)} \\
= & \sum_{(u,v)\in \mathrm{In}(v)} \left(x(a^{VF}_{u,v}) - x(a^{HV}_{u,v}) + 1\right) \cdot 90 \\
= & \overbrace{\sum_{(u,v)\in \mathrm{In}(v)} \left(x(a^{VF}_{u,v}) - x(a^{HV}_{u,v})\right)}^{=4-\deg(v)\ \text{(Gl.4.2)}} \cdot 90 + \deg(v) \cdot 90 \\
= & \ 360 \ \checkmark
\end{aligned}$$

- (Kr5): Sei $\alpha_{(u,v)} = 0 \Longrightarrow x(a^{HV}_{u,v}) = x(h_{u,v}, n_v) = 1$. Aufgrund der Flusserhaltung bei dem Knoten $h_{u,v}$ gilt: $x\left(a^{L}_{u,v}\right) + x\left(a^{R}_{w,v}\right) = 1$, wobei $(v,w) = \mathrm{succ}_f(u,v)$ ist. Das bedeutet:

$$\begin{aligned}
s^{E}_{(u,v)} &= 0^{x(a^{R}_{u,v})} 1^{x(a^{L}_{u,v})} = 1 \text{ oder} \\
s^{A}_{(v,w)} &= 0^{x(a^{L}_{w,v})} 1^{x(a^{R}_{w,v})} = 1
\end{aligned}$$

Für den 0°-Winkel $\alpha_{(u,v)}$ existiert somit genau ein Knotenknick, der diesem Winkel eindeutig zugeordnet werden kann. ✓

▶ Die Anzahl der Knicke in K ist gleich den Kosten von x:

$$\begin{aligned}
\mathrm{bends}(K) &= \sum_{(u,v)\in \bar{E}} \left|s_{(u,v)}\right|_0 \\
&= \sum_{(u,v)\in \bar{E}} \left(x\left(a^{L}_{v,u}\right) + x\left(a^{FF}_{u,v}\right) + x\left(a^{R}_{u,v}\right)\right) \\
&= \mathrm{cost}(x) \ \checkmark
\end{aligned}$$

■

Der folgende Satz folgt direkt aus Lemma 4.3 und Lemma 4.4.

Satz 5 *Sei $G = (V, E, F)$ ein planarer Graph mit fixer Einbettung P. Jede Kandinsky Repräsentation von G mit k Knicken kann aus einem gültigen, ganzzahligen Fluss x auf dem Kandinsky Netzwerk $\mathcal{N}^K(P)$ mit Kosten k abgeleitet werden, der zugleich die Bündelkapazitätsbedingung einhält. Die An-*

zahl der Knicke in einer knickminimalen Kandinsky Repräsentation von G ist gleich den Kosten eines ganzzahligen minimalen Kostenflusses im Kandinsky Netzwerk $\mathcal{N}^K(P)$, der zugleich die Bündelkapazitätsbedingung einhält.

4.2.2 Das Alternativ Kandinsky Netzwerk

Sei $G = (V, E, F)$ ein planarer Graph mit gegebener Einbettung P. Das Alternativ Kandinsky Netzwerk, das wir mit $\mathcal{N}^{AK} = (N, A)$ bezeichnen, wird wie folgt aufgebaut.

Es wird zunächst wieder für jeden Knoten $v \in V$ und für jede Fläche $f \in F$ jeweils ein Knoten (n_v und n_f) in das Netzwerk eingeführt, jedoch mit einem anderen Bedarf als im Tamassia Netzwerk. Es ist nun

$$N = N_V \cup N_F$$

wobei

- $N_V = \{n_v \mid v \in V\} \quad b(n_v) = \mathbf{4} \quad \forall n_v \in N_V$
- $N_F = \{n_f \mid f \in F\} \quad b(n_f) = \begin{cases} -(\mathbf{2}|f| - 4) & f \text{ Innenfläche} \\ -(\mathbf{2}|f| + 4) & f \text{ Außenfläche} \end{cases}$

Die Kanten in A sind die gleichen wie in A^T, mit dem einzigen Unterschied, dass $v - f$ Kanten in $\mathcal{N}^{AK}$ eine unterschiedliche obere Kapazitätsgrenze haben:

$$A = A_{VF} \cup A_{FF}$$

wobei nun

- $A_{VF} = \{a^{VF}_{u,v} = (n_v, n_f) \quad \mid f = \text{face}(u, v),\ (u, v) \in \bar{E}\}$
  $\text{lcap}(a) = 0,\ \text{ucap}(a) = \mathbf{4},\ \text{cost}(a) = 0 \quad \forall a \in A_{VF}$

- $A_{FF} = \{a^{FF}_{u,v} = (n_f, n_g) \quad \mid f = \text{face}(u, v),$
  $g = \text{face}(v, u),\ (u, v) \in \bar{E}\}$
  $\text{lcap}(a) = 0,\ \text{ucap}(a) = \infty,\ \text{cost}(a) = 1 \quad \forall a \in A_{FF}.$

Der Fluss auf einer $v - f$ Kante repräsentiert weiterhin den Winkel, der an dem Knoten v in der Fläche f geformt wird, dieser ist jedoch gegeben durch:

$$x\left(a_{u,v}^{VF}\right) \cdot 90 \tag{4.7}$$

Anders als im originalen Kandinsky Netzwerk entspricht hier eine Einheit, die

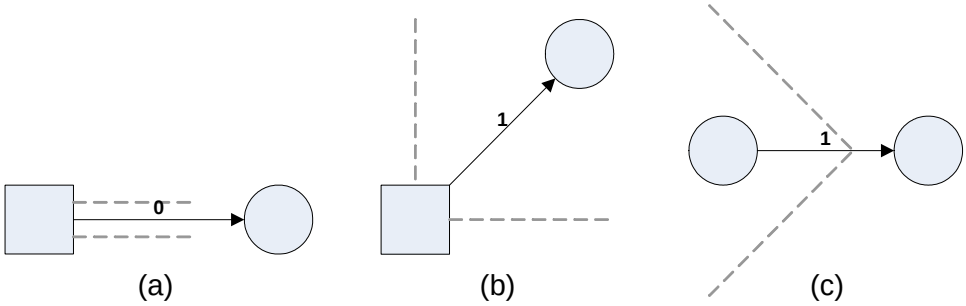

Abbildung 4.14: Ein Fluss von einer Einheit im Netzwerk entspricht einem 90°-Winkel bzw. einem 90°-Knick im Graph

von einem *Knoten* in eine *Fläche* fließt, einem 90°-Winkel, und nicht einem 180°-Winkel (Abb. 4.14(b)). 0°-Winkel werden durch 0-Flüsse auf $v - f$ Kanten modelliert (Abb. 4.14(a)). Ein Fluss auf einer $f - f$ Kante entspricht weiterhin einem konvexen Knick in der Ausgangsfläche(Abb. 4.14(c)).

Kriterium (*Kr5*)

Damit der Fluss auf $\mathcal{N}^{AK}$ einer korrekten Kandinsky Darstellungen entspricht, muss für jeden 0°-Winkel die Existenz des zugehörigen 270°-Knickes sichergestellt sein.

Die Einhaltung des Kriteriums (*Kr5*) wird durch eine ähnliche Hilfskonstruktion wie im originalen Kandinsky Netzwerk sichergestellt: Es wird ein neuer Hilfsknoten $h_{u,v}$ eingeführt und die Kante (n_v, f_1) (Abb. 4.15(a)) durch die zwei Kanten $(n_{v,}h_{u,v})$ und $(h_{u,v}, f_1)$ ersetzt (Abb. 4.15(b)). Nun beschreibt der Fluss auf der Kante $(n_{v,}h_{u,v})$ den Winkel, den die Kante (u, v) mit ihrem Nachfolger auf der Fläche f_1 formt. Die Kante $(h_{u,v}, f_1)$ hat eine untere Schranke von 1. Der Hilfsknoten $h_{u,v}$ muss folglich einen Zufluss

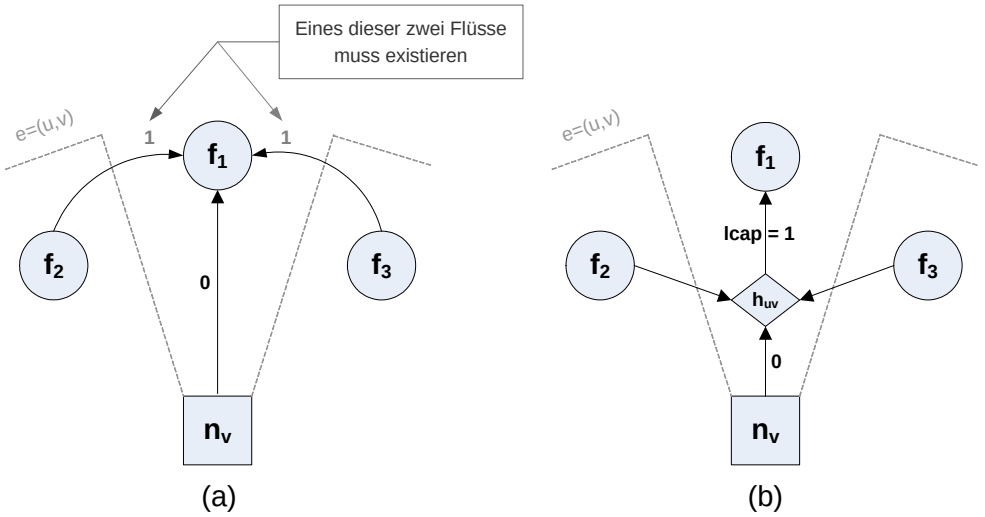

Abbildung 4.15: 0°-Winkel erzwingen Knotenknicke (a), die durch die Hilfskonstruktion modelliert werden (b).

von mindestens einer Einheit haben. Wenn die nötige Einheit nicht von n_v kommt, dann muss sie entweder von f_2 oder von f_3 kommen. Dadurch wird erreicht, dass ein 0°-Winkel nur mit einem zugehörigen 270°-Knick (Knotenknick) auftritt.

Auch in diesem Netzwerk entsprechen die Flüsse auf den Kanten $(f_1, h_{e'})$ und (f_2, h_e) (Abb.4.16(a)) einem Knotenknick am gleichen Ende der Kante in $e = (u, v) \in E$ und dürfen daher nicht beide den Wert 1 haben. Wir fassen daher zwei solche Kanten zu einem *Bündel* zusammen und setzen die Kapazität des Bündels auf 1.

Das so aufgebaute Netzwerk können wir nun wie folgt zusammenfassen:

Definition 4.3 ***Das Alternativ Kandinsky Netzwerk*** *Sei $G = (V, E, F)$ ein planarer Graph mit fixer Einbettung P. Die Knoten des Alternativ Kandinsky Netzwerkes $\mathcal{N}^{AK}(P) = (N, A)$ setzen sich wie folgt zusammen:*

$$N = N_V \cup N_F \cup N_H$$

wobei

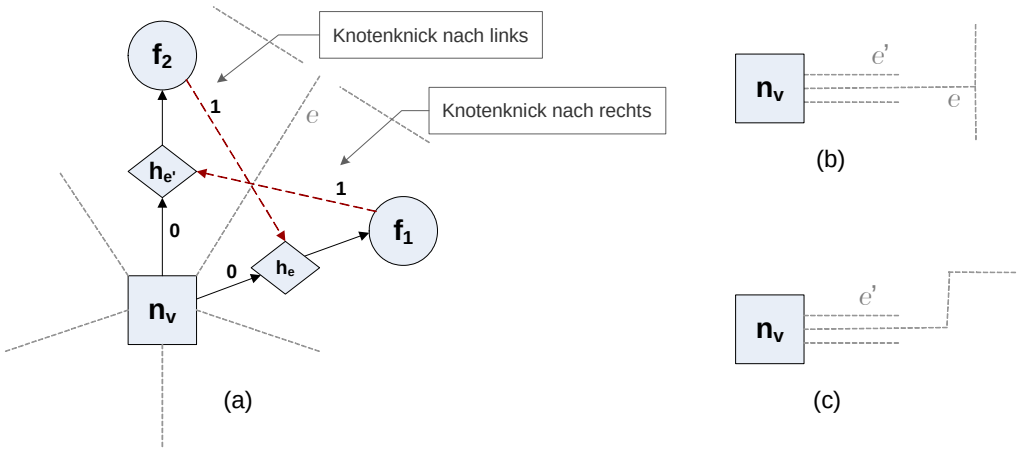

Abbildung 4.16: Um zwei Knotenknicke an demselben Ende einer Kante e in entgegengesetzte Richtungen zu verhindern (a), müssen Bündelkapazitäten gesetzt werden. Ungültige Zeichnungen wie in (b) und (c) werden dadurch verhindert.

- $N_V = \{n_v \mid v \in V\}$ $\quad b(n_v) = 4 \quad \forall n_v \in N_V$
- $N_F = \{n_f \mid f \in F\}$ $\quad b(n_f) = \begin{cases} -(2|f|-4) & f \textit{ Innenfläche} \\ -(2|f|+4) & f \textit{ Außenfläche} \end{cases}$
- $N_H = \{h_{u,v} \mid (u,v) \in \bar{E}\}$ $\quad b(h_{u,v}) = 0 \quad \forall h_{u,v} \in N_H$

Die Menge der Kanten ist gegeben durch

$$A = A_{VH} \cup A_{HF} \cup A_{FF} \cup A_{FH}$$

wobei

- $A_{VH} = \{a_{u,v}^{VH} = (n_v, h_{u,v}) \mid (u,v) \in \bar{E}\}$
  $\text{lcap}(a) = 0,\ \text{ucap}(a) = 4,\ \text{cost}(a) = 0 \quad \forall a \in A_{VH}$

- $A_{HF} = \{a_{u,v}^{HF} = (h_{u,v}, n_f) \mid f = \text{face}(u,v),\ (u,v) \in \bar{E}\}$
  $\text{lcap}(a) = 1,\ \text{ucap}(a) = \infty,\ \text{cost}(a) = 0 \quad \forall a \in A_{HF}$

- $A_{FF} = \{a^{FF}_{u,v} = (n_f, n_{f'}) \quad | \; f = \text{face}(u,v),$
  $f' = \text{face}(v,u),\ (u,v) \in \bar{E}\}$
  $\text{lcap}(a) = 0, \text{ucap}(a) = \infty,\ \text{cost}(a) = 1 \quad \forall a \in A_{FF}$
- $A_{FH} = \bigcup_{(u,v)\in\bar{E}} B_{u,v}$
  wobei
  $B_{u,v} = \quad \{a^R_{u,v} = (n_f, h_{(w,v)}) \mid f = \text{face}(u,v), (w,v) = \text{pred}_f(v,u)\} \cup$
  $\{a^L_{u,v} = (n_{f'}, h_{u,v}) \mid f' = \text{face}(v,u)\}$
  $\text{lcap}(a) = 0,\ \text{ucap}(a) = 1,\ \text{cost}(a) = 1 \quad \forall a \in A_{FH}$
  $\text{ucap}(B_{u,v}) = 1 \quad \forall (u,v) \in \bar{E}$

Lemma 4.5 *Sei $G = (V, E, F)$ ein planarer Graph mit fixer Einbettung P und $\mathcal{N}^{AK}(P)$ das zugehörige Alternativ Kandinsky Netzwerk. Für jeden ganzzahligen Fluss x in $\mathcal{N}^{AK}(P)$ gibt es eine Kandinsky Repräsentation K, deren Anzahl von Knicken gleich den Kosten des Flusses ist. Weiters kann K wie folgt aus dem Fluss berechnet werden:*

- *Für jedes $(u,v) \in \bar{E}$:*

$$\alpha_{(u,v)} = x(a^{HF}_{u,v}) \cdot 90$$

- *Für jedes $(u,v) \in \bar{E}$:*

$$s_{(u,v)} = \overbrace{0^{x(a^L_{v,u})}1^{x(a^R_{v,u})}}^{s^A_{(u,v)}}\overbrace{1^{x(a^{FF}_{v,u})}0^{x(a^{FF}_{u,v})}}^{s^M_{(u,v)}}\overbrace{0^{x(a^R_{u,v})}1^{x(a^L_{u,v})}}^{s^E_{(u,v)}}$$

Beweis. Es muss gezeigt werden, dass K den Kriterien (Kr1)-(Kr5) genügt. Für diesen Beweis verweisen wir auf den Beweis von Lemma 4.4, da diese sich nur geringfügig voneinander unterscheiden. ■

Lemma 4.6 *Sei $G = (V, E, F)$ ein planarer Graph mit fixer Einbettung P. Für jede Kandinsky Repräsentation K von G gibt es einen ganzzahligen Fluss x im Alternativ Kandinsky Netzwerk $\mathcal{N}^{AK}(P)$, dessen Kosten gleich der Anzahl der Knicke in K ist.*

Beweis. Wir konstruieren nun aus der Darstellung K einen Fluss x mit den geforderten Eigenschaften. Sei im folgenden $(u,v) \in \bar{E}$ und $(v,w) = \mathrm{succ}_f(u,v)$.

▶ Konstruktion des Flusses unter Einhaltung der Kapazitätsschranken:

- $\forall\ a^{VH}_{u,v} \in A_{VF}$

$$\begin{aligned} x\left(a^{VH}_{u,v}\right) &= \frac{\alpha_{(u,v)}}{90} \\ &\leq 4 = \mathrm{ucap}\left(a^{VH}_{u,v}\right) \quad \checkmark \end{aligned}$$

- $\forall\ a^{HF}_{u,v} \in A_{HV}$

$$\begin{aligned} x\left(a^{HF}_{u,v}\right) &= \left|s^{A}_{(v,w)}\right|_1 + \left|s^{E}_{(u,v)}\right|_1 + \frac{\alpha_{(u,v)}}{90} \\ &\geq 1 = \mathrm{lcap}\left(a^{HF}_{u,v}\right) \quad \checkmark \end{aligned}$$

$\left|s^{A}_{(v,w)}\right|_1$ und $\left|s^{E}_{(u,v)}\right|_1$ entsprechen Knotenknicken, die dem Winkel $\alpha_{(u,v)}$ zugeordnet sind. Ist $\alpha_{(u,v)} = 0$, dann hat genau eines dieser Strings die Länge 1. Daher gilt: $\left|s^{A}_{(v,w)}\right|_1 + \left|s^{E}_{(u,v)}\right|_1 + \frac{\alpha_{(u,v)}}{90} \geq 1$.

- $\forall\ a^{FF}_{u,v} \in A_{FF}$

$$\begin{aligned} x\left(a^{FF}_{u,v}\right) &= \left|s^{M}_{(u,v)}\right|_0 \\ &\leq \infty = \mathrm{ucap}\left(a^{FF}_{u,v}\right) \quad \checkmark \end{aligned}$$

- $\forall\ a^{R}_{u,v},\ a^{L}_{u,v} \in A_{FH}$

$$\begin{aligned} x\left(a^{R}_{u,v}\right) &= \left|s^{E}_{(u,v)}\right|_0 \leq 1 \\ x\left(a^{L}_{u,v}\right) &= \left|s^{E}_{(u,v)}\right|_1 \leq 1 \text{ und} \\ x\left(a^{R}_{u,v}\right) + x\left(a^{L}_{u,v}\right) &= \left|s^{E}_{(u,v)}\right|_0 + \left|s^{E}_{(u,v)}\right|_1 \\ &\leq 1 = \mathrm{ucap}\left(B_{u,v}\right) \quad \checkmark \end{aligned}$$

▶ Für den Beweis der Flusserhaltung an allen Knoten in $\mathcal{N}^{AK}$ verweisen wir auf den entsprechenden Teil des Beweises von Lemma 4.3, da diese sich nur

geringfügig voneinander unterscheiden.
▶ Für den Beweis von cost $(x) =$ bends (K) sei ebenfalls auf den Beweis von Lemma 4.3 verwiesen. ■

Der folgende Satz folgt direkt aus diesen beiden Lemmata:

Satz 6 *Sei $G = (V, E, F)$ ein planarer Graph mit fixer Einbettung P. Jede Kandinsky Repräsentation von G mit k Knicken kann aus einem ganzzahligen gültigen Fluss x auf dem Alternativ Kandinsky Netzwerk $\mathcal{N}^{AK}(P)$ mit Kosten k abgeleitet werden, der zugleich die Bündelkapazitätsbedingung einhält. Die Anzahl der Knicke in einer knickminimalen Kandinsky Repräsentation von G ist gleich den Kosten eines ganzzahligen minimalen Kostenflusses im Alternativ Kandinsky Netzwerk $\mathcal{N}^{AK}(P)$, der zugleich die Bündelkapazitätsbedingung einhält.*

4.2.3 Das Kandinsky MCF Problem

Die Berechnung einer knickminimalen Kandinsky Zeichnung kann nun auf das folgende, spezielle Min-Cost Flow Problem zurückgeführt werden:

Definition 4.4 ***Das Kandinsky Min-Cost Flow (KMCF) Problem***
Das Problem, auf einem gegebenen Kandinsky Netzwerk $\mathcal{N}^{K}(P)$ (bzw. Alternativ Kandinsky Natzwerk $\mathcal{N}^{AK}(P)$) einen gültigen, ganzzahligen Fluss x mit minimalen Kosten zu berechnen, der zugleich die Bündelkapazitätsbedingung einhält, wird als das Kandinsky MCF Problem bezeichnet.

Durch die zusätzliche Bündelkapazitätsbedingung handelt es sich nicht mehr um ein gewöhnliches MCF Problem. Die klassischen MCF Algorithmen können daher nicht ohne weiteres auf das KMCF Problem angewendet werden.

Kapitel 5

Lösungsansätze für das KMCF Problem

In [Foe97] stellen Fößmeier und Kaufmann einen MCF basierten Lösungsansatz für das KMCF Problem vor. Sie behaupten, dass nach Einführung einer Hilfskonstruktion in das Kandinsky Netzwerk das MCF Problem auf diesem mit einer etwas modifizierten Version des Successive Shortest Path (SSP) Algorithmus (Alg. 2.2, S. 17) in polynomialer Zeit gelöst werden kann. Wie Eiglsperger in [Eig03] festgestellt hat, ist das jedoch nicht der Fall. In Abschn. 5.1 fassen wir den Fößmeier-Kaufmann Ansatz kurz zusammen und gehen näher auf die Probleme ein, die bei der Anwendung des SSP Algorithmus auf das KMCF Problem auftreten. Es wird sich herausstellen, dass diese Schwierigkeiten auch bei den anderen *Optimal Infeasible Flow* Algorithmen bestehen, weshalb sie nicht ohne weiteres auf das KMCF Problem anwendbar sind. Daher ist die Komplexität dieses Problems bisher unbekannt.

In Abschn. 5.2 stellen wir den von Eiglsperger in [Eig03] eingeführten Transformationslagorithmus mit Güte 2 vor. In Abschn. 5.4 führen wir unseren neu entwickelten Cyclic-Shift Algorithmus ein, der ebenfalls den Approximationsfaktor 2 besitzt, aber in der Praxis bessere Ergebnisse liefert. Zwischen diesen beiden Ansätzen besteht ein grundsätzlicher Unterschied, den wir in Abschn. 5.5 aufzeigen und der eine Erklärung für die bessere Qualität der CS Lösungen liefert.

5.1 Der Fößmeier-Kaufmann Ansatz

Um auf einem Kandinsky Netzwerk $\mathcal{N}^K = (N, A)$ eingesetzt werden zu können, muss der SSP Algorithmus so modifiziert werden, dass die Einhaltung der Bündelkapazitätsbedingung sichergestellt ist. Dazu muss folgendes beachtet werden:

1. Sind in einem Iterationsschritt des SSP beide Kanten eines Bündels im aktuellen Restnetzwerk $\mathcal{N}(x)$ enthalten, darf ein kürzester Pfad $\mathrm{SP}(s, l) \in \mathcal{N}(x)$ nicht beide Kanten dieses Bündels zugleich enthalten.

2. Hat eine der Bündelkanten bereits den Flusswert eins, darf ein kürzester Pfad $\mathrm{SP}(s, l)$ die andere Kante dieses Bündels nicht enthalten.

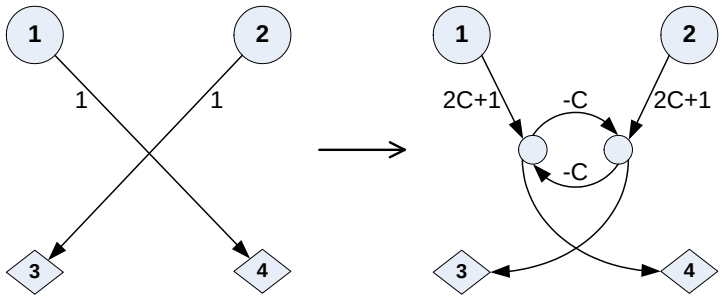

Abbildung 5.1: Jedes Bündel wird durch eine Hilfskonstruktion ersetzt, der die gleichzeitige Verwendung beider Kanten des Bündels verhindern soll.

In [FK96] schlagen Fößmeier und Kaufmann vor, jedes Bündel im Netzwerk durch die in Abb. 5.1 dargestellte Konstruktion zu ersetzen. Der Kreis mit negativen Kosten hat die Kapazität 1. C ist eine hinreichend große Zahl (z.B. obere Schranke für das Optimum). Diese Konstruktion soll verhindern, dass beide Kanten eines Bündels gleichzeitig verwendet werden. Da der Kreis die Gesamtkapazität 1 besitzt, können nur die hohen Kosten einer zum Kreis adjazenten Kante damit aufgehoben werden. Das Problem, dass durch beliebig häufiges Durchlaufen eines negativen Kreises ein beliebig kurzer Kantenzug gefunden werden kann, wird dadurch gelöst, indem man sich auf kürzeste Wege beschränkt (Kantenfolgen, die jede Kante höchstens

einmal enthalten). Ist der Kreis einmal voll, würde ein Weg über die andere Kante an dem Kreis vorbei müssen, ohne sie durchlaufen zu können. So ein Weg hätte Kosten $> 2C + 1$ und wäre mit Sicherheit nicht in der optimalen Lösung enthalten.

Bemerkung 5.1 *Eine Kantenfolge kann in einem solchen Netzwerk Knoten mehrfach enthalten (nämlich dann, wenn sie Kreise durchläuft) und wird daher als Weg bezeichnet.*

Nun muss aber das Shortest Path Problem auf einem Netzwerk mit negativen Kreisen gelöst werden, was im allgemeinen NP schwierig ist [GJ79]. In [FK96] wird jedoch behauptet, dass hier aufgrund der speziellen Struktur der negativen Kreise ein einfacherer Fall vorliegt und eine etwas modifizierte Version von Dijkstra's Algorithmus das Shortest Path Problem auf einem solchen Kandinsky Netzwerk in polynomialer Zeit lösen kann. Im folgenden fassen wir zunächst die Argumente in [FK96] zusammen.

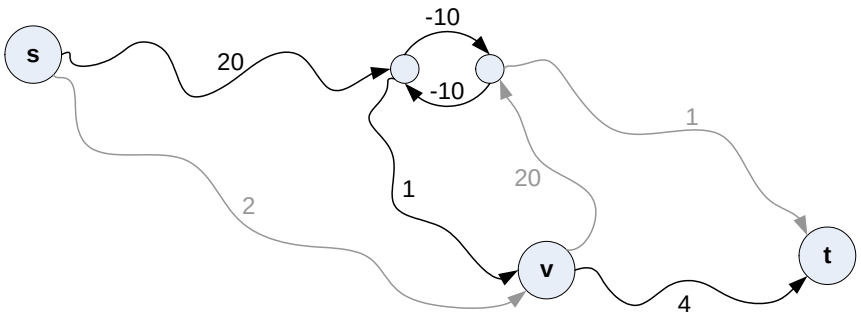

Abbildung 5.2: Der negative Kreis kann nicht von den kürzesten Teilwegen $\mathrm{SP}(s, v)$ und $\mathrm{SP}(v, t)$ zugleich benutzt werden.

Im allgemeinen Fall kann in einem Netzwerk mit negativen Kreisen folgendes Problem auftreten. Sei $SP(s, t)$ ein kürzester Weg von s nach t und sei v ein Knoten in diesem Weg (siehe Abb. 5.2). Dann setzt sich $SP(s, t)$ aus zwei Teilwegen zusammen: $SP(s, v)$ und $SP(v, t)$. Der kürzeste Weg von s nach v benutzt den negativen Kreis und hat Kosten 1*. Danach kann der kürzeste Weg von v nach t den Kreis nicht mehr durchlaufen und man erhält*

den in der Abbildung schwarz markierten kürzesten(!) Weg von s nach t mit Gesamtkosten 5. Würde man hingegen den grau markierten Weg benutzen, könnte nun der Teilweg von v nach t den negativen Kreis durchlaufen und man hätte die Gesamtkosten 3. Ein Shortest Path Algorithmus findet diesen Weg nicht, weil er nur die kürzesten Wege von s nach v weiterverfolgt. Die Schwierigkeit des Problems rührt also daher, dass die Kosten des zweiten Teilweges $SP(v,t)$ von dem Verlauf des ersten Teilweges $SP(s,v)$ abhängen.

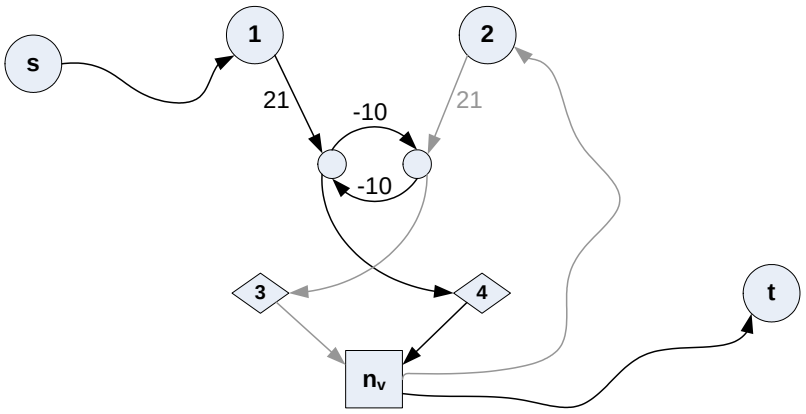

Abbildung 5.3: Auf dem Startnetzwerk $\mathcal{N}^K(x^0)$ kann das Shortest Path Problem trotz der negativen Kreise korrekt gelöst werden.

Bei Kandinsky kann das nicht passieren, da jede Flusseinheit, die den negativen Kreis durchläuft, anschließend in den Knoten n_v fließt (siehe Abb. 5.3). Wird der negative Kreis von dem kürzesten Weg von s nach n_v benutzt (schwarz markierter Weg), dann ist garantiert, dass er nicht Teil eines kürzesten Weges von n_v nach t ist. Damit arbeitet der Shortest Path Algorithmus korrekt.

Im Start-Netzwerk stimmt diese Argumentation, in einem Restnetzwerk in den weiteren Schritten des SSP muss es nicht mehr stimmen. Es kann zu einem Zeitpunkt im Restnetzwerk ein Fall wie in Abb. 5.4 vorliegen. Eine Flusseinheit, die den leeren negativen Kreis benutzt, fließt hier anschließend nicht in den Knoten n_v. Liegt der negative Kreis auf einem kürzesten Weg von s nach 4 (grau markiert), ist nicht bewiesen, dass er zugleich nicht Teil eines

kürzesten Weges von 4 nach t (rot markiert) ist. Somit ist nicht bewiesen, dass das kürzeste Wege Problem in allen Iterationen des SSP korrekt gelöst werden kann.

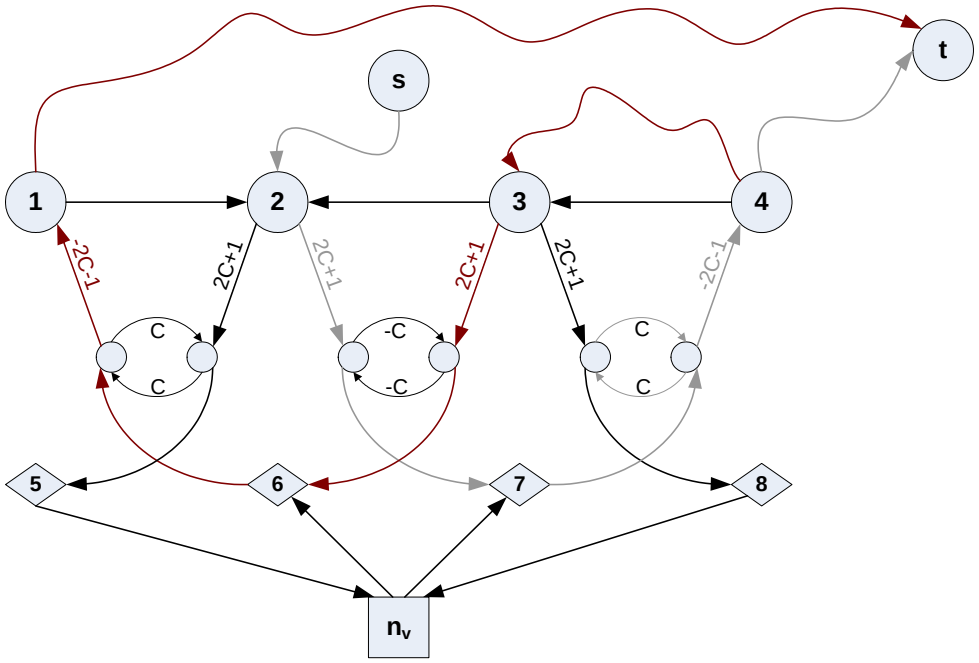

Abbildung 5.4: In den Zwischenschritten des SSP kann es sein, dass negative Kreise im Restnetzwerk eine korrekte Berechnung der kürzesten Wege verhindern.

Selbst wenn dem so wäre, ist das kein Beweis dafür, dass das MCF Problem korrekt gelöst werden kann. Wie aus dem folgenden Beispiel aus [Eig03] hervorgeht, führt das sukzessive Augmentieren entlang kürzester Wege nicht unbedingt zu einer optimalen Lösung. In Abb. 5.5(a) ist das Bündel $((1,4),(2,3))$ durch die Konstruktion mit negativen Kreisen ersetzt, um hervorzuheben, dass sie nichts an der nicht optimalen Lösung ändert. Die Kanten, bzw. Pfade sind mit ihren Kosten, die Knoten sind mit ihren kürzesten Distanzen im aktuellen Restnetzwerk beschriftet. Nicht beschriftete Kanten haben Kosten 0. Alle Kanten und Bögen haben Kapazität 1, $b(\mathsf{s}) = 2$ und $b(\mathsf{t}) = -2$.

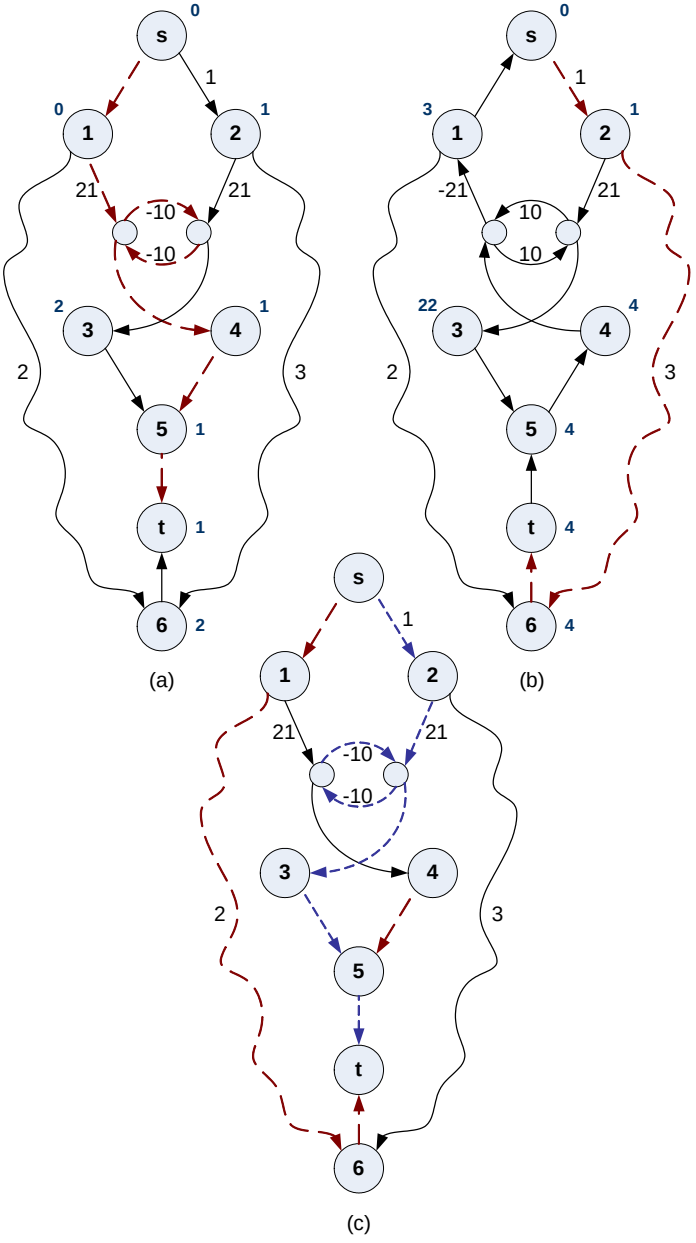

Abbildung 5.5: Der von SSP berechnete, nicht optimale Flus mit Kosten 5 ((a),(b)) und der optimale Fluss mit Kosten 4 (c).

Beispiel 5.1 *Im Startnetzwerk Abb. 5.5(a) ist (s,1,4,5,t) der kürzeste augmentierende Weg von s nach t mit Kosten* 1*, der den negativen Kreis durchläuft. Im nächsten Schritt (Abb. 5.5(b)) ist (s,2,6,t) der kürzeste Weg und hat die Kosten* 4*. Somit liefert SSP einen Fluss mit Kosten* 5 *zurück. Würde man hingegen im ersten Schritt den in Abb. 5.5(c) rot markierten Weg von s nach t mit Kosten* 2 *verwenden, wäre im zweiten Schritt der negative Kreis frei und der blau markierte Weg (s,2,3,5,t) mit Kosten* 2 *könnte benutzt werden. Dieser Fluss hätte insgesamt Kosten* 4 *und wäre die optimale Lösung gewesen. SSP kann diesen Fluss nicht finden, da er in jedem Schritt entlang eines kürzesten Weges von s nach t augmentiert.*

Es liegt ein ähnliches Problem wie bei Netzwerken mit negativen Kreisen vor: die Kosten eines kürzesten Weges in einem Iterationsschritt des SSP hängen von den kürzesten Wegen in den vorherigen Schritten ab.

Bei allen Netzwerk-Fluss basierten MCF Algorithmen, die auf dem Prinzip beruhen '*den kürzesten Weg zuerst zu füllen*', wird dasselbe Problem auftreten.

5.2 Eiglsperger's Ansatz für das KMCF Problem

Eiglsperger hat das Problem mit den bisherigen Ansätzen zur Lösung des KMCF- Problems erkannt. Er fand jedoch keinen korrekten Lösungsansatz für das KMCF Problem, sondern einen 2-Approximationsalgorithmus, der in diesem Abschnitt vorgestellt wird.

Sei $G = (V, E, F)$ ein planarer Graph mit einer fixen Einbettung P und sei $\mathcal{N}^K(P) = \left(N^K, A^K\right)$ das dazugehörige Kandinsky Netzwerk, das wie in Def. 4.2 definiert ist.

5.2.1 Ein Transformationsalgorithmus

In dem von Eiglsperger in [Eig03] vorgestellten Algorithmus wird zunächst das Min-Cost Flow Problem auf dem Kandinsky Netzwerk gelöst, ohne da-

bei die Bündelkapazitätsbedingung zu berücksichtigen. Die Lösung x kann dadurch Bündel enthalten, deren Kapazität überschritten ist.

Definition 5.1 *Ein Bündel* $B_{u,v} = \left(a^R_{u,v}, a^L_{u,v}\right)$ *wird als* ***überfüllt*** *bezeichnet, wenn*

$$x\left(a^R_{u,v}\right) + x\left(a^L_{u,v}\right) = 2$$

ist. $B_{u,v}$ *wird als* ***leer*** *bezeichnet, wenn beide Bündelkanten den Flusswert* 0 *haben und als* ***nicht leer****, wenn genau eine Bündelkante den Flusswert* 1 *hat.*

x entspricht genau dann einer gültigen Lösung des Kandinsky MCF Problems, wenn es keine überfüllten Bündel enthält. Ansonsten werden die überfüllten Bündel wie folgt durch Flussverschiebungen entlang geeigneter Kreise in nicht überfüllte Bündel transformiert.

In den folgenden Zeichnungen sind aus Übersichtlichkeitsgründen von den zwei $f - f$ Kanten zwischen zwei *Flächen* jeweils nur eine eingezeichnet, die $v - f$ Kanten sind nicht eingezeichnet.

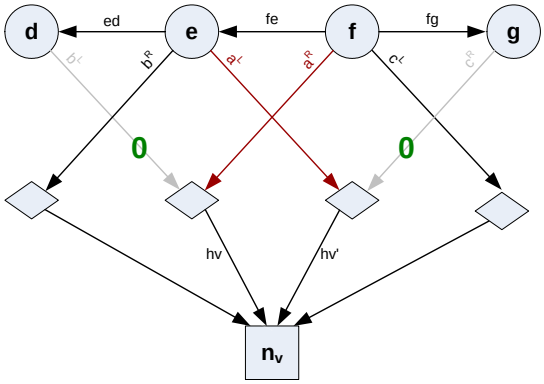

Abbildung 5.6: Das Nachbarbündel von einem überfüllten Bündel ist nicht überfüllt.

Sei $B(v)$ die zyklische Liste der um den *Knoten* $n_v \in N^K$ angeordneten Bündel und $A = \left(a^R, a^L\right)$ ein überfülltes Bündel in dieser Liste. Weiters seien $B = \left(b^R, b^L\right)$ das linke und $C = \left(c^R, c^L\right)$ das rechte Nachbarbündel von A

(Abb. 5.6). Dann gilt:

$$x\left(b^L\right) = 0 \quad \text{wegen ucap}\left(hv\right) = 1 \text{ und } x\left(a^R\right) = 1 \text{ und} \tag{5.1}$$

$$x\left(c^R\right) = 0 \quad \text{wegen ucap}\left(hv'\right) = 1 \text{ und } x\left(a^L\right) = 1. \tag{5.2}$$

Für B gibt es zwei Möglichkeiten:

- Ist $x\left(b^R\right) = 0$, dann ist B ein leeres Bündel.
- Ist $x\left(b^R\right) = 1$, dann ist B nicht leer *und* für den linken Nachbarn von B liegt derselbe Fall wie für B vor.

Geht man also von A aus nach links, trifft man solange auf *nicht leere* Bündel, bis man zuletzt auf ein leeres Bündel stößt. Das ist spätestens dann der Fall, wenn man den Knoten n_v umrundet und C erreicht hat, für den dann $x\left(c^R\right) = x\left(c^L\right) = 0$ ist.

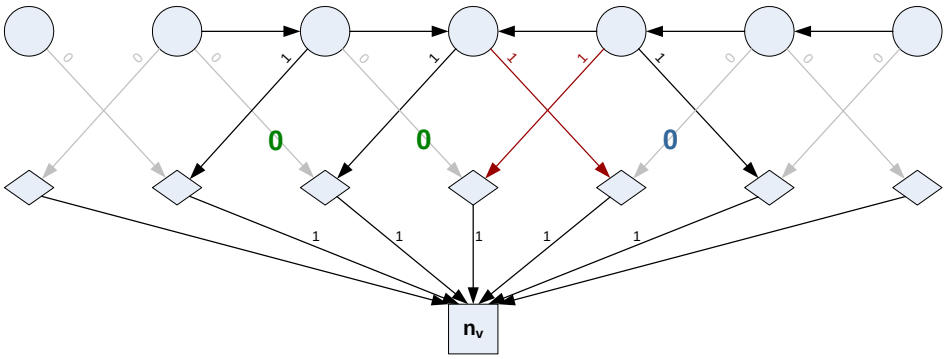

Abbildung 5.7: Eine maximale Teilliste I mit $r(I) = 1$, $l(I) = 2$.

Dieselben Überlegungen gelten für die Bündel rechts von A. Folglich enthält jede Teilliste von $B\left(v\right)$, die aus nicht leeren Bündeln besteht, höchstens ein überfülltes Bündel. Eine solche Teilliste von nicht leeren Bündeln und einem überfüllten Bündel, das von beiden Seiten an ein leeres Bündel angrenzt, wird als *maximale Teilliste* bezeichnet.

Sei $I\left(v\right)$ die Menge aller maximalen Teillisten in $B\left(v\right)$. Für jedes $I \in I\left(v\right)$ sei $r\left(I\right)$ (bzw. $l\left(I\right)$) die Anzahl der nicht leeren Bündel rechts (bzw. links)

von dem überfüllten Bündel in I (Abb. 5.7). Es gibt zwei Möglichkeiten, das überfüllte Bündel in I zu korrigieren:

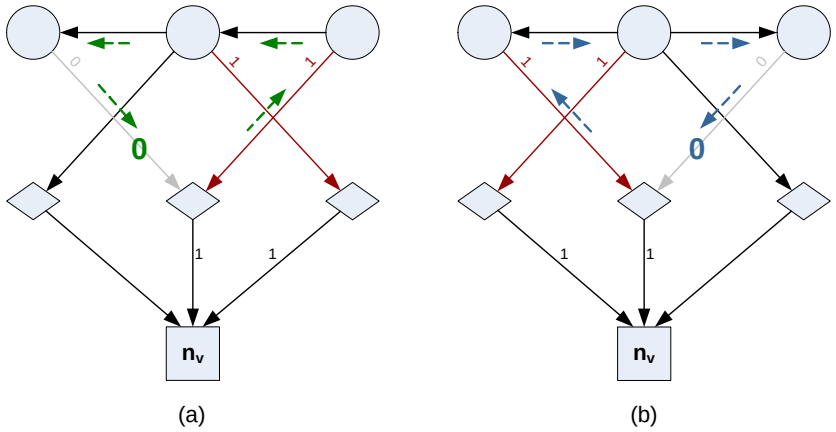

Abbildung 5.8: Links- und Rechts-Transformation bei einem überfüllten Bündel.

Links-Transformation: Der Fluss kann entlang des in Abb. 5.8,(a) mit grünen Pfeilen markierten Kreises mit Kosten **2** um eine Einheit verschoben werden. Ist das Bündel links von dem überfüllten Bündel nicht leer, dann wird es nach diesem Kreisfluss zu einem überfüllten Bündel. In dem Fall wird derselbe Kreisfluss für dieses Bündel wiederholt. Nach insgesamt $l(I) + 1$ Schritten kann man aufhören, da das leere Bündel links von I erreicht ist und kein neues, überfülltes Bündel mehr entsteht. Insgesamt hat eine solche Links-Transformation die Kosten $2 \cdot (l(I) + 1)$.

Eine **Rechts-Transformation** kann auf die gleiche Weise, nur spiegelverkehrt und in die andere Richtung durchgeführt werden (Abb. 5.8,(b)) und hat die Gesamtkosten $2 \cdot (r(I) + 1)$.

Es kann nun alles wie folgt zusammengefasst werden:

Algorithmus 5.1 ***Der Transformationsalgorithmus von Eiglsperger (TE)***

TE1 *Löse das Kandinsky MCF Problem auf dem Kandinsky Netzwerk ohne Bündelkapazitätsbeschränkung.*

$\longrightarrow$ *Zu diesem Problem existiert immer eine ganzzahlige Lösung.*

TE2 Für alle Knoten $v \in V$, korrigiere alle überfüllten Bündel in $I(v)$

- *mittels einer Links-Transformation, wenn*

$$\sum_{I \in I(v)} l(I) \leq \sum_{I \in I(v)} r(I)$$

- *mittels einer Rechts-Transformation, wenn*

$$\sum_{I \in I(v)} l(I) > \sum_{I \in I(v)} r(I).$$

Satz 7 *Der Transformationsalgorithmus berechnet eine 2-Approximation für das Kandinsky MCF Problem.*

Beweis. [Eig03] Sei x die Lösung des KMCF Problems ohne Bündelkapazitätsbeschränkung mit dem Zielfunktionswert $z_{TE1} = z(x)$. Es ist klar, dass $z_{TE1} \leq z_{opt}$ ist. Es ist auf jeden Fall

$$z_{TE1} \geq \sum_{v \in V} \sum_{I \in I(v)} (|I| + 1)$$

da jedes nicht leere Bündel in I den Kostenfaktor 1 und das eine überfüllte Bündel den Kostenfaktor 2 hat.

Da $|I| = l(I) + r(I) + 1$ ist, gilt außerdem

$$2 \cdot (l(I) + 1) \leq |I| + 1 \quad \text{wenn } l(I) \leq r(I) \text{ und} \tag{5.3}$$

$$2 \cdot (r(I) + 1) \leq |I| + 1 \quad \text{wenn } r(I) < l(I). \tag{5.4}$$

Seien

$$V_l = \left\{ v \in V : \sum_{I \in I(v)} l(I) \leq \sum_{I \in I(v)} r(I) \right\}$$

$$V_r = \left\{ v \in V : \sum_{I \in I(v)} l(I) > \sum_{I \in I(v)} r(I) \right\}$$

Für die Transformationskosten gilt daher

$$
\begin{aligned}
c_T &= \sum_{v \in V_l} \sum_{I \in I(v)} 2 \cdot (l(I) + 1) + \sum_{v \in V_r} \sum_{I \in I(v)} 2 \cdot (r(I) + 1) && (5.5)\\
&\leq \sum_{v \in V_l} \sum_{I \in I(v)} (|I| + 1) + \sum_{v \in V_r} \sum_{I \in I(v)} (|I| + 1) && (5.6)\\
&= \sum_{v \in V} \sum_{I \in I(v)} (|I| + 1) \leq z_{TE1} && (5.7)
\end{aligned}
$$

Sei $\hat{x}$ der von dem Transformationsalgorithmus berechnete Fluss. Für seine Kosten $z_{TE} = z(\hat{x})$ gilt

$$z_{TE} = z_{TE1} + c_T \leq 2 \cdot z_{TE1} \leq 2 \cdot z_{opt} \ .$$

■

5.2.2 Eine verbessernde Heuristik

Der obige Algorithmus transformiert einen Fluss mit überfüllten Bündeln in eine gültige Lösung des KMCF Problems, indem es in einem Durchlauf explizit die Flusswerte von Kanten innerhalb maximaler Teillisten verändert. Eiglsperger führt an dieser Stelle eine verbessernde Heuristik ein, den wir als den Sukzessiven Transformationsalgorithmus bezeichnen und die eine bessere Leistung erzielen soll.

Algorithmus 5.2 ***Der Sukzessive Transformationsalgorithmus (ST)***

ST1 *Löse das Kandinsky MCF Problem auf dem Kandinsky Netzwerk ohne die Bündelkapazitätsbeschränkung.*
Enthält die Lösung überfüllte Bündel, gehe zu ST2. Andernfalls, gehe zu ST4.

ST2 *Für jeden Knoten $v \in V$ korrigiere die überfüllten Bündel in $B(v)$ (wenn vorhanden) mittels einer Links- oder Rechtstransformation (Alg. 5.1, TE2).*

ST3 *Setze die obere Kapazität jener Bündelkanten auf Null, die vor der Transformation den Flusswert* 1*, nach der Transformation den Flusswert* 0 *besitzen.*
Gehe zu ST1.

ST4 *Terminiere mit einer Näherungslösung für das KMCF Problem.*

In den einzelnen Iterationen kann die Lösung in ST1 jedoch überfüllte Bündel enthalten, die nicht ohne weiteres korrigiert werden können. Aus dieser Beschreibung der Heuristik geht nicht hervor, wie in solchen Situationen vorgegangen werden soll. Betrachten wir dazu folgenden Fall: Sei $A = \left(a^R, a^L\right) \in I$ ein überfülltes Bündel in $B\left(v\right)$, für den in der transformierten Lösung die Kante a^L den Flusswert Null hat (Abb. 5.9(a)), sodass im 2. Schritt $\text{ucap}\left(a^L\right) = 0$ gesetzt wird.Sei in einem späteren Durchlauf

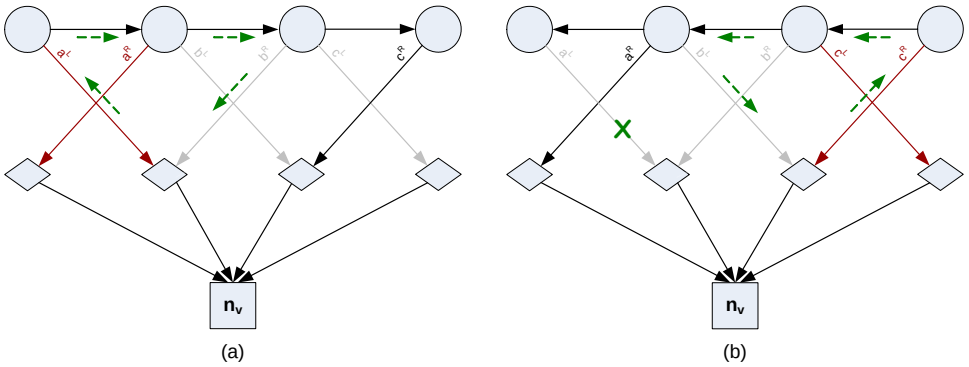

Abbildung 5.9: (a) Nach einer Rechts-Transformation hat die Kante a^L den Flusswert 0. Sie wird daher gesperrt. (b) Die Kante c^R wird gesperrt.

$C = \left(c^R, c^L\right)$ ein überfülltes Bündel, für den in der transformierten Lösung die Kante c^R den Flusswert 0 hat. Es wird daher $\text{ucap}\left(c^R\right) = 0$ gesetzt (Abb. 5.9,(b)). Ist in einem späteren Durchlauf das Bündel $B = \left(b^R, b^L\right)$ überfüllt, kann es nicht korrigiert werden, da sowohl Links- als auch Rechts-Transformation nicht möglich sind, ohne dabei Fluss durch die gesperrten Bündelkanten zu schicken (Abb. 5.10).

In unserer Implementierung korrigieren wir in dem Fall das Bündel B wie gewohnt (TE2), indem wir dabei alle gesperrten Bündelkanten, die nach

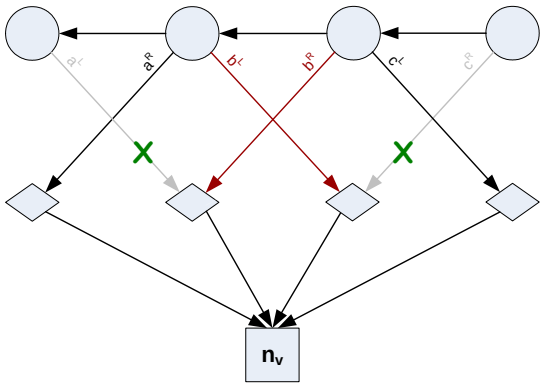

Abbildung 5.10: Das überfüllte Bündel B kann nicht korrigiert werden.

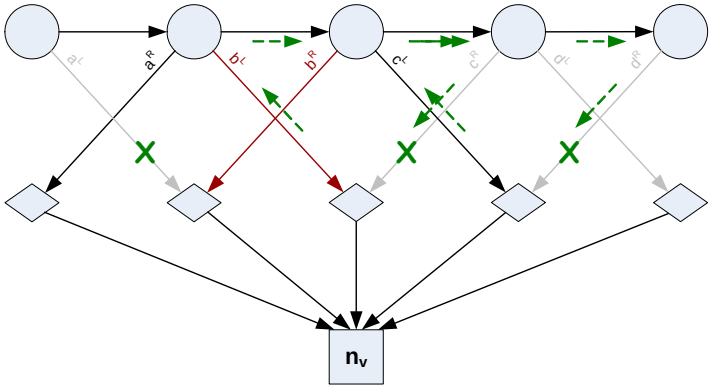

Abbildung 5.11: Es werden die zuvor gesperrten Kanten der Bündel C und D freigegeben und jeweils die anderen Kanten gesperrt.

der Transformation den Flusswert 1 haben, wieder freigeben und die andere Kante des jeweiligen Bündels sperren (Abb. 5.11). Die Heuristik terminiert weiterhin, da in einem solchen Schritt kein gesperrtes Bündel (ein Bündel mit mindestens einer gesperrten Kante) wieder freigegeben und mindestens ein weiteres Bündel gesperrt wird (Abb. 5.12). Die Gesamtkosten dieses *Modifizierten Sukzessiven Transformationsalgorithmus* sind im schlimmsten Fall quadratisch in der Anzahl der Bündel, da jedes Bündel bei Bedarf beliebig oft umgeschaltet werden kann.

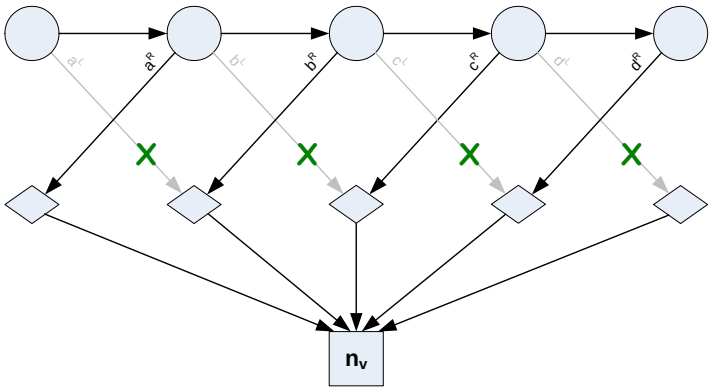

Abbildung 5.12: Nach der Korrektur ist das Bündel B gesperrt.

5.3 Simple Kandinsky Zeichnungen

In [BDD00] stellen die Autoren das Simple Kandinsky Modell vor, in der knickminimale Zeichnungen mit Hilfe eines viel einfacheren Netzwerkes berechnet werden können.

Die durch die Bündel verursachte Schwierigkeit wird umgangen, in dem Knotenknicke nur in eine Richtung erlaubt werden. Das führt zu einem Bündelfreien Netzwerk.

Definition 5.2 *Eine* ***Simple Kandinsky Zeichnung*** *ist eine Kandinsky Zeichnung, die zusätzlich folgende Bedingungen erfüllt:*

1. *Knoten mit Grad* > 4 *haben auf jeder Seite mindestens eine inzidente Kante.*

2. *An Knoten mit Grad* ≤ 4 *sind keine* $0°$*-Winkel erlaubt.*

3. *Es sind nur Knotenknicke nach rechts erlaubt.*

Die zu einem Knoten inzidenten Kanten werden also möglichst auf die vier Seiten des Knotens verteilt und 0°-Winkel nur dort eingeführt, wo sie unbedingt nötig sind. Das macht das Problem aber nicht einfacher. Erst die Einschränkung, dass alle Knotenknicke in eine Richtung, nämlich nach

rechts, knicken dürfen, führt zu einer Vereinfachung im Kandinsky Netzwerk und ermöglicht so die Verwendung von klassischen MCF Algorithmen.

Vereinfachen wir das in Def. 4.2 eingeführte Kandinsky Netzwerk soweit, dass es zunächst nur die 3. Einschränkung einhält:

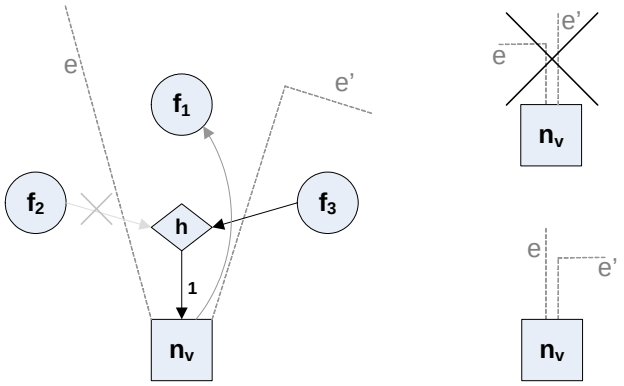

Abbildung 5.13: Knotenknicke im Simple-Kandinsky Modell

Wenn zwei aufeinanderfolgende Kanten e und $e\prime$ auf der Fläche f_1 einen 0°-Winkel formen (Abb. 5.13) , muss die Kante $e\prime$, die in der zyklischen Liste $K(f_1)$ nach der Kante e kommt, nach außen knicken. Die $f - h$ Kante, die einem Links-Knotenknick auf der Kante e entspricht, wird nicht eingeführt. Das gilt für alle Kanten $e \in \bar{E}$.

Wir entfernen folglich alle Kanten in der Menge $\{a^L_{u,v} = (n_f, h_{u,v}) : f = \text{face}(v, u)\}$ aus dem Kandinsky Netzwerk, wodurch es keine Bündel mehr enthält. Somit kann ein minimaler Kostenfluss auf diesem vereinfachten Kandinsky Netzwerk mit den gewöhnlichen MCF Algorithmen in polynomialer Zeit berechnet werden. Die Einschränkungen 1. und 2. sind für die Vereinfachung des Problems daher nicht unbedingt notwendig.

Definition 5.3 ***One Way Nodebend (OWN) Kandinsky Netzwerk*** *Sei $G = (V, E, F)$ ein planarer Graph und sei $\mathcal{N}^K = (N^K, A^K)$ das nach der Def. 4.2 aufgestellte Kandinsky Netzwerk. Wir bezeichnen das Netzwerk $(N^K, A^K - \{a^L_{u,v}\})$ (bzw. $(N^K, A^K - \{a^R_{u,v}\})$) als das OWN Kandinsky Netz-*

werk, in der die Teilmenge A_{FH} von Kanten gegeben ist durch

$$\begin{aligned} A_{FH} = \{a^R_{u,v} = (n_f, h_{w,v}) \quad | \quad & f = \text{face}(u,v),\ (w,v) = \text{pred}_f(v,u) \\ & (u,v),(v,u),(w,v) \in \bar{E}\} \\ \textit{(bzw. } A_{FH} = \{a^L_{u,v} = (n_f, h_{u,v}) \quad | \quad & f = \text{face}(v,u),\ (u,v),(v,u) \in \bar{E}\} \textit{)} \end{aligned}$$

Satz 8 *Sei $G = (V, E, F)$ ein planarer Graph. Jeder gültige Fluss x im OWN Kandinsky Netzwerk, das wie in der Def. 5.3 beschrieben aufgebaut ist, entspricht einer gültigen Kandinsky Zeichnung des zugrunde liegenden Graphen G ohne Links-Knotenknicke (bzw. ohne Rechts-Knotenknicke).*

Beweis. Wir müssen zeigen, dass jeder gültige Fluss im OWN Kandinsky Netzwerk einer Repräsentation entspricht, die den Kriterien (*Kr1*)-(*Kr4*) (S. 24) und (*Kr5*) (S. 38) genügt.

- **Kr1** ist automatisch erfüllt, da wir das Netzwerk entsprechend der planaren Repräsentation des Graphen G aufstellen. ✓
- **Kr2** Die Bit-Strings erstellen wir auf folgende Weise: Für jedes Paar $(u,v),\ (v,u) \in \bar{E}$:

$$\begin{aligned} s_{(u,v)} &= 1^{x(a^R_{v,u})} 1^{x(a^{FF}_{v,u})} 0^{x(a^{FF}_{u,v})} 0^{x(a^R_{u,v})} \\ s_{(v,u)} &= 1^{x(a^R_{u,v})} 1^{x(a^{FF}_{u,v})} 0^{x(a^{FF}_{v,u})} 0^{x(a^R_{v,u})} \end{aligned}$$

  (bzw. analog für den Fall ohne-Rechts-Knotenknicke). Somit ist (*Kr2*) ebenfalls erfüllt. ✓

▶ **Kr3** Sei $f \in F$ eine innere Fläche.

$$\begin{aligned}
& \sum_{r_{(u,v)} \in K(f)} \rho(u,v) \\
= & \sum_{r_{(u,v)} \in K(f)} (|s_{(u,v)}|_0 - |s_{(u,v)}|_1 + 2 - \frac{\alpha_{(u,v)}}{90}) \\
= & \; 2\,|f| + \overbrace{\sum_{r_{(u,v)} \in K(f)} \left(x\left(a^{FF}_{u,v}\right) + x\left(a^{R}_{u,v}\right) - x\left(a^{FF}_{v,u}\right) - x(a^{VF}_{u,v})\right)}^{=-(|f|-4) \text{ (Gl. 4.3)}} \\
& - \sum_{r_{(u,v)} \in K(f)} \left(x\left(a^{R}_{v,u}\right) - x\left(a^{HV}_{u,v}\right) + 1\right)
\end{aligned}$$

$$\begin{aligned}
= & \; 2\,|f| - |f| + 4 - |f| - \overbrace{\sum_{r_{(u,v)} \in K(f)} \left(x\left(a^{R}_{v,u}\right) - x\left(a^{HV}_{u,v}\right)\right)}^{=0 \text{ (Gl. 4.5)}} \\
= & \; 4 \;\checkmark
\end{aligned}$$

Für den Fall ohne Rechts-Knotenknicke ersetzt man oben $a^R_{u,v}$ durch $a^L_{v,u}$ und $a^R_{v,u}$ durch $a^L_{u,v}$. Der Rest bleibt gleich und ist gültig. Auf die gleiche Weise kann man zeigen, dass für die Außenfläche f_o gilt:

$$\sum_{r_{(u,v)} \in K(f_o)} \rho(u,v) = -4 \;\checkmark$$

▶ **Kr4**

$$\begin{aligned}
& \sum_{(u,v) \in \mathrm{In}(v)} \alpha_{(u,v)} \\
= & \sum_{(u,v) \in \mathrm{In}(v)} \left(x(a^{VF}_{u,v}) - x(a^{HV}_{u,v}) + 1\right) \cdot 90 \\
= & \overbrace{\sum_{(u,v) \in \mathrm{In}(v)} \left(x(a^{VF}_{u,v}) - x(a^{HV}_{u,v})\right)}^{=4-\deg(v) \text{ (Gl. 4.2)}} \cdot 90 + \deg(v) \cdot 90 \\
= & \; 360 \;\checkmark
\end{aligned}$$

- **Kr5** Ist erfüllt, da jedem 0°-Winkel ein Rechts-Knotenknick $a^R_{u,v}$ (bzw. Links-Knotenknick) zugeordnet werden kann. ✓

■

Definition 5.4 *Mit **One Way Nodebend Kandinsky Zeichnungen** bezeichnen wir Kandinsky Zeichnungen ohne Links-Knotenknicke (bzw. ohne Rechts-Knotenknicke).*

Das in [BDD00] eingeführte Netzwerk zur Berechnung knickminimaler Simple Kandinsky Zeichnungen ist aufgrund der zusätzlichen zwei Einschränkungen viel einfacher gebaut als das OWN Kandinsky Netzwerk. Das MCF Problem kann aber auf beiden in polynomialer Zeit gelöst werden. Solange man Knotenknicke nur in eine Richtung erlaubt ist es Geschmackssache, ob man die Kanten um einen Knoten möglichst auf alle vier Seiten verteilt haben möchte oder nicht.

5.4 Der Cyclic-Shift Ansatz

In diesem Abschnitt stellen wir einen neuen Algorithmus mit Gütegarantie 2 vor, der in der Praxis sehr gute Ergebnisse liefert. Dazu wird das Kandinsky MCF Problem zunächst als Ganzzahliges Lineares Programm formuliert. Dabei verwenden wir das in Abschn. 4.2.2 beschriebene Alternative Kandinsky Netzwerk als Grundlage. Ausgehend von der nicht ganzzahligen Lösung der LP-Relaxierung wird eine ganzzahlige Lösung konstruiert, die im schlimmsten Fall höchstens zwei Mal so schlecht wie die LP-Lösung ist.

5.4.1 Die ILP-Formulierung des KMCF Problems

Es ist möglich, eine ILP-Formulierung für das Kandinsky Knickminimierungsproblem aufzustellen, die direkt auf der Kandinsky Repräsentation basiert [EFK00]. Wir nehmen in unserer ILP-Formulierung das Alternativ Kandinsky Netzwerk als Basis.

Sei $G = (V, E, F)$ ein planarer Graph mit fixer Einbettung P und sei $\mathcal{N}^{AK}(P) = (N, A)$ das Alternativ Kandinsky Netzwerk, das wie in Def. 4.3 (S. 53) definiert ist.

Mit jeder Kante $a \in A$ im Netzwerk sind drei ganze Zahlen assoziiert: Kosten: $\text{cost}(a) \geq 0$, eine untere Kapazitätsgrenze $\text{lcap}(a) \geq 0$ und eine obere Kapazitätsgrenze $\text{ucap}(a) \geq 0$

Mit jedem Knoten $n_v \in N$ ist eine ganze Zahl assoziiert: Bedarf $b(n_v) \in \mathbb{Z}$.

Für jede Kante $a \in A$ führen wir eine Flussvariable $x(a)$ ein. Das Ganzzahlige Lineare Programm für das Kandinsky Min-Cost Flow Problem (KMCF-ILP) ist wie folgt definiert:

$$\min \sum_{a \in A} \text{cost}(a) \cdot x(a) \tag{5.8}$$

sodass

$$\sum_{a \in \text{Out}(n)} x(a) - \sum_{a \in \text{In}(n)} x(a) = b(n) \qquad \forall n \in N \tag{5.9}$$

$$x(a^L_{u,v}) + x(a^R_{u,v}) \leq 1 \qquad \forall (u, v) \in \bar{E} \tag{5.10}$$

$$\text{lcap}(a) \leq x(a) \leq \text{ucap}(a) \qquad \forall a \in A \tag{5.11}$$

$$x(a) \text{ ganzzahlig} \qquad \forall a \in A \tag{5.12}$$

$\bar{E}$ ist hier die Menge der Richtungskanten (siehe Def. 2.11, S. 11,). Die Ungleichung 5.10 bezeichnen wir als die Bündelkapazitätsbedingung, wobei $B_{u,v} = \left(a^L_{u,v}, a^R_{u,v}\right)$ ein Bündel im Netzwerk ist (siehe Def. 4.3, S. 53).

5.4.2 Der Cyclic-Shift Ansatz

Wir lassen nun die Ganzzahligkeitsbedingung 5.12 weg und lösen die dadurch resultierende LP-Relaxierung des Problems. Wegen der Bedingung 5.10 ist die Lösung x der LP-Relaxierung im allgemeinen nicht ganzzahlig. Es gibt also Kanten mit nicht ganzzahligen Flusswerten, insbesondere Bündel $B_{u,v}$, für die $a^L_{u,v}$ und $a^R_{u,v}$ beide nicht ganzzahlige Flusswerte haben, die in der

Summe ≤ 1 sind.

Definition 5.5 *Sei x die Lösung der LP-Relaxierung des KMCF-ILPs für einen Graphen $G = (V, E, F)$. Jene Bündel $B_{u,v}$, für die*

$$x(a^L_{u,v}) > 0 \ \wedge \ x(a^R_{u,v}) > 0$$

*ist, bezeichnen wir als **kritische Bündel**. Die Bündel, in denen mindestens eine Kante den Flusswert 0 hat, nennen wir **unkritisch**.*

Der Cyclic-Shift Algorithmus, der im weiteren detailliert beschrieben wird, kann wie folgt zusammengefasst werden:

Algorithmus 5.3 ***Cyclic-Shift Algorithmus (CS)***

Eingabe: *Planarer Graph $G = (V, E, F)$ mit fixer Einbettung P.*

Ausgabe: *Eine regionerhaltende Kandinsky Repräsentation des Graphen G mit z_{CS} Knicken für die folgende Abschätzung gilt:*

$$z_{CS} \leq 2 \cdot z_{opt}$$

wobei z_{opt} die minimale Knickanzahl der optimalen Lösung ist.

CS1 *Es wird die LP-Relaxierung des Kandinsky MCF Problems gelöst.*
→ Es entsteht eine optimale, jedoch nicht ganzzahlige Lösung mit Zielfunktionswert z_{CS1}. Es gilt $z_{CS1} \leq z_{opt}$.

CS2 *Es werden geeignete Kreisflüsse angebracht, durch die in einer Kante jedes Bündels der Fluss 0 erzeugt wird.*
→ Dieser Schritt verursacht zusätzliche Kosten, der Zielfunktionswert erhöht sich auf z_{CS2}, die durch

$$z_{CS2} \leq 2 \cdot z_{CS1}$$

abgeschätzt werden kann (siehe Abschn. 5.4.4).
Die auf 0 zu setzenden Bündelkanten und die entstehenden Kosten können auch ohne Modifikation der Flüsse ermittelt werden (und nur darauf kommt es an!), sodass man letzteres auch lassen kann.

CS3 *Aus jedem Bündel wird für eine Kante mit dem Flusswert* 0 *die obere Kapazitätsgrenze auf* 0 *gesetzt.*
→ Danach kann von jedem Bündel nur mehr höchstens eine Kante Fluss aufnehmen, sodass die Bündelkapazitätsbeschränkung entfallen kann. Dadurch reduziert sich das KMCF Problem auf ein gewöhnliches MCF Problem (siehe Def. 2.15).

CS4 *Die LP-Relaxierung des in CS3 entstandenen MCF Problems wird von neuem mit ganzzahligen Startwerten gelöst.*
→ Es entsteht eine ganzzahlige, zulässige Näherungslösung für das KMCF Problem mit dem Zielfunktionswert z_{CS}*, sodass* $z_{CS} \leq z_{CS2}$ *ist. Daher gilt die Abschätzung*

$$z_{CS} \leq 2 \cdot z_{CS1} \leq 2 \cdot z_{opt}$$

CS5 *Transformiere die jetzt ganzzahlige Lösung in eine Kandinsky Repräsentation mit* $z_{CS} \leq 2 \cdot z_{opt}$ *Knicken.* □

Beim Auffinden einer optimalen Lösung des KMCF Problems kommt es im Grunde nur darauf an, in jedem Bündel die richtigen Kante auf Fluss 0 zu setzen. Im besten Fall werden in Schritt CS3 genau jene Bündelkanten gesperrt, die in einer optimalen Lösung keinen Fluss enthalten. Da uns leider keine Kriterien für eine solche optimale Auswahl zur Verfügung stehen, versuchen wir möglichst gute Auswahlkriterien festzulegen.

In einem unkritischen Bündel sperren wir die Kante, welches durch die LP-Lösung schon den Fluss 0 erhalten hat. Bei einem kritischen Bündel versuchen wir Kreisflüsse mit möglichst geringen Kosten zu finden, die es unkritisch machen (Schritt CS2) und sperren die Kante, die danach den Flusswert 0 hat.

Sei $\mathcal{B}(v)$ die zyklische, im Uhrzeigersinn um einen Knoten n_v angeordnete Liste von Bündeln. Wir zeigen nun, wie man alle kritischen Bündel in $\mathcal{B}(v)$ in unkritische Bündel transformieren kann. Dabei ist in den Fällen 1 und 2 $A = \left(a^L, a^R\right)$ ein kritisches Bündel, das rechts neben einem nicht kritischen Bündel $B = \left(b^L, b^R\right)$ steht. Sind alle Bündel in $\mathcal{B}(v)$ kritisch, liegt Fall 3 vor.

Aus Gründen der Übersichtlichkeit wird in den folgenden Zeichnungen von den zwei $f - f$ Kanten, die zwischen zwei *Flächen* verlaufen, immer nur eine eingezeichnet.

Fall 1: Wir setzen voraus, dass $x\left(b^L\right) = 0$ ist.

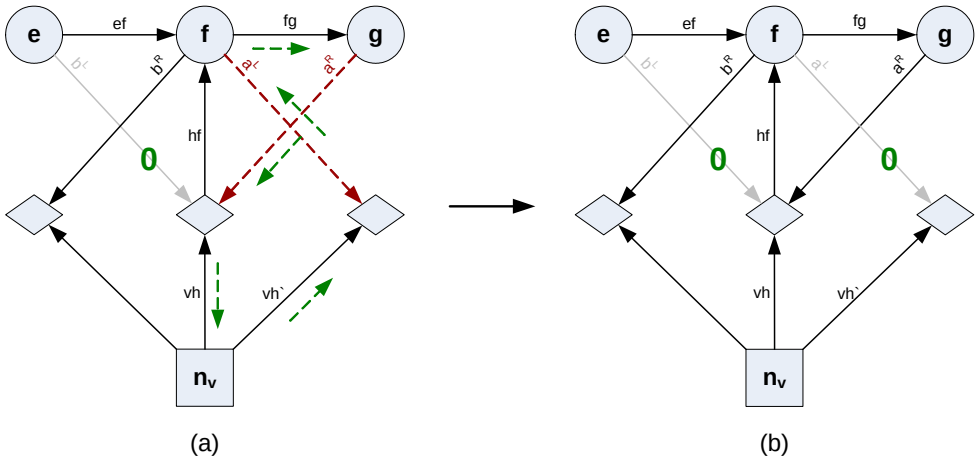

Abbildung 5.14: Eine Links-Korrektur vom Typ 1.

In diesem Fall ist

$$x\left(a^R\right) + x\left(vh\right) \geq 1 \text{ wegen } x\left(b^L\right) = 0 \text{ und } \text{lcap}(hf) = 1 \quad (5.13)$$

$$x\left(a^R\right) + x\left(a^L\right) \leq 1 \text{ wegen der Bündelkapazität} \quad (5.14)$$

$$\text{Daher gilt} \quad :$$

$$x\left(vh\right) \geq x(a^L)$$

Somit kann der Fluss entlang des in Abb. 5.14,(a) durch kurze, grüne Pfeile gekennzeichneten Kreises $\left(-a^L,\ fg,\ a^R,\ -vh,\ vh`\right)$ um $x(a^L)$ Einheiten augmentiert werden. Nach dieser Augmentierung ist der Fluss in a^L gleich 0 und das Bündel A somit unkritisch (Abb. 5.14,(b)). Am Bündel B hat sich nichts geändert.

Diese Korrektur, die wir als eine **Typ 1 Korrektur** bezeichnen, hat insgesamt Kosten

$$\begin{aligned} \mathrm{cost}_{\mathrm{L},1}(A) &= x\left(a^L\right) \cdot \left(\mathrm{cost}(fg) + \mathrm{cost}(a^R) - \mathrm{cost}(a^L)\right) \\ &= x\left(a^L\right) \end{aligned}$$

Dies ist eigentlich eine obere Schranke. Im günstigeren Fall hat die Kante gf eine positive Kapazität, sodass sie einen Teil des Kreisflusses aufnehmen kann. Ein Teil der von fg verursachten Kosten wird in dem Fall von der in negativer Richtung durchlaufenen Kante gf wieder aufgehoben und man kommt insgesamt auf geringere Kosten. Da wir aber im allgemeinen nicht davon ausgehen können, werden wir im folgenden diese (und ähnliche) Fälle außer Acht lassen. Wir verwenden weiterhin die Bezeichnung 'Kosten', behalten aber im Kopf, dass es sich in Wirklichkeit um eine Kostenschranke handelt.

Man beachte, dass die Bündelkanten und die Kanten, die zwischen zwei *Flächen* verlaufen, Kosten 1 haben. Alle anderen Kanten im Netzwerk haben Kosten 0 (Def. 4.3).
Im spiegelverkehrten Fall, in dem A links von dem nicht kritischen Bündel B mit $x\left(b^R\right) = 0$ steht, kann die *Typ 1* Korrektur analog in die andere Richtung durchgeführt werden und hat die Kosten

$$\mathrm{cost}_{\mathrm{R},1}(A) = x\left(a^R\right).$$

Fall 2: Wir setzen voraus, dass $x\left(b^R\right) = 0$ ist.
In diesem Fall augmentieren wir zunächst (falls $x(hf) > 1$) den Fluss entlang des Kreises $\left(-a^R,\ gf,\ -hf\right)$ mit Kosten 0 um $\min\left\{x\left(a^R\right),\ x(hf) - 1\right\}$ Einheiten (Abb. 5.15,(a)). Danach gibt es zwei Möglichkeiten:

- Fluss in a^R gleich 0: dann ist das Bündel A unkritisch, und es ist weiter nichts zu tun.
- Fluss in hf an der unteren Grenze 1: dann ist wegen der Flusserhaltungsbedingung der Zufluss von b^L und a^R in den Hilfsknoten ≤ 1.

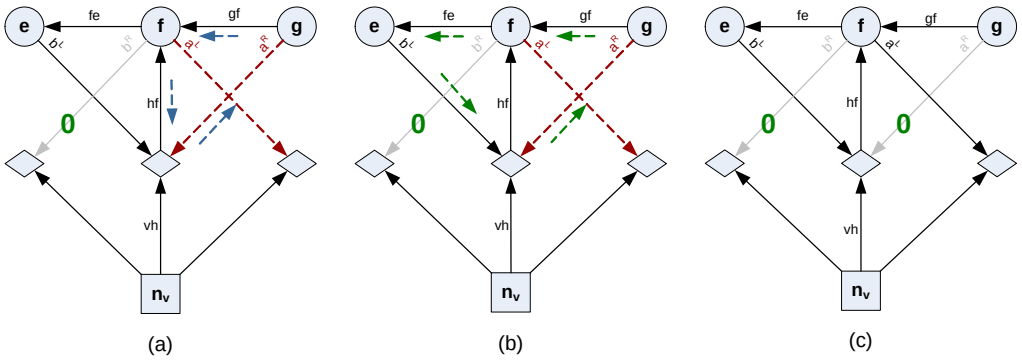

Abbildung 5.15: Eine Links-Korrektur vom Typ 2.

Daher können wir den Fluss entlang des in Abb. 5.15,(b) durch kurze, grüne Pfeile gekennzeichneten Kreises $\left(-a^R,\ gf,\ fe,\ b^L\right)$ so augmentieren, dass danach a^R den Fluss 0 hat und das Bündel A unkritisch ist (Abb. 5.15,(c)). Im Bündel B bleibt $x\left(b^R\right) = 0$ erhalten, der Fluss in b^L wird vergrößert, bleibt jedoch ≤ 1.

Diese Korrektur, die wir als eine **Typ 2 Korrektur** bezeichnen, hat insgesamt Kosten

$$\begin{aligned} \mathrm{cost}_{\mathrm{L},2}(A) &= x\left(a^R\right) \cdot \left(-\mathrm{cost}(a^R) + \mathrm{cost}(gf) + \mathrm{cost}(fe) + \mathrm{cost}(b^L)\right) \\ &= 2 \cdot x\left(a^R\right) \end{aligned}$$

Analog hat eine *Typ 2* Korrektur in die andere Richtung (sofern wieder Fall 2 vorliegt) die Kosten

$$\mathrm{cost}_{\mathrm{R},2}(A) = 2 \cdot x\left(a^L\right).$$

Bemerkung 5.2 *Steht rechts von Bündel A ein weiteres kritisches Bündel, können wir die gleiche Korrektur nun auf dieses anwenden. Es ist also leicht einzusehen, dass bei jedem Knoten n_v, dessen Bündelliste $\mathcal{B}(v)$ mindestens ein unkritisches Bündel enthält, alle kritischen Bündel um diesen Knoten in unkritische Bündel transformiert werden können.*

Fall 3: Wir setzen voraus, dass alle Bündel um den Knoten n_v kritisch sind. In diesem Fall benutzen wir bei den Bündeln A und B dieselben korri-

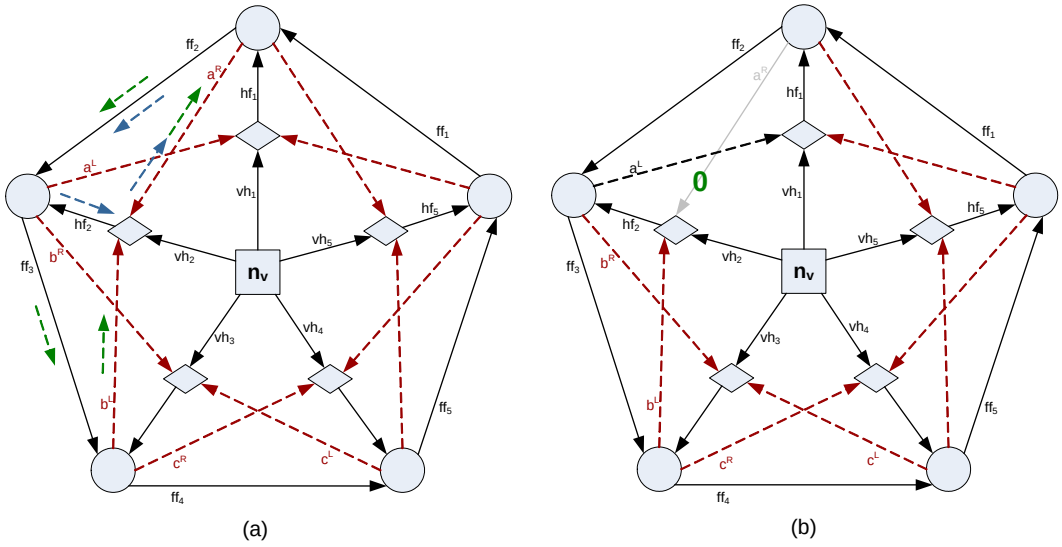

Abbildung 5.16: Alle Bündel um n_v sind kritisch.

gierenden Kreisflüsse wie bei der *Typ 2* Korrektur (Abb. 5.16). Das Bündel A wird dadurch unkritisch, das Bündel B bleibt kritisch. Da die *Typ 2* - Voraussetzung $x\left(b^R\right) = 0$ nicht erfüllt ist, wird dabei seine Bündelkapazität möglicherweise überschritten, nicht aber die Kapazität der einzelnen Kanten b^R und b^L.

Falls die Überschreitung der Bündelkapazität nicht auftritt, können wir alternativ mit dem Durchlauf aufhören und für die restlichen kritischen Bündel wie in Fall 1 oder Fall 2 vorgehen. Andernfalls müssen wir die oben für die Bündel A und B beschriebenen Operationen auf die Bündel B und C anwenden. Dadurch erreichen wir ein unkritisches Bündel B und beseitigen *auch* die Kapazitätsverletzung bei diesem Bündel.

Danach fahren wir entsprechend fort. Spätestens enden diese Fortsetzungen erfolgreich nach einer vollen Umrundung des Knotens n_v, weil beim letzten Bündelpaar (rechter Nachbar von A und A selbst) wegen $x\left(a^R\right) = 0$ eine gewöhnliche *Typ 2* Korrektur stattfindet (Abb. 5.17,(a)). Danach sind alle Bündel um n_v unkritisch (Abb. 5.17,(b)).

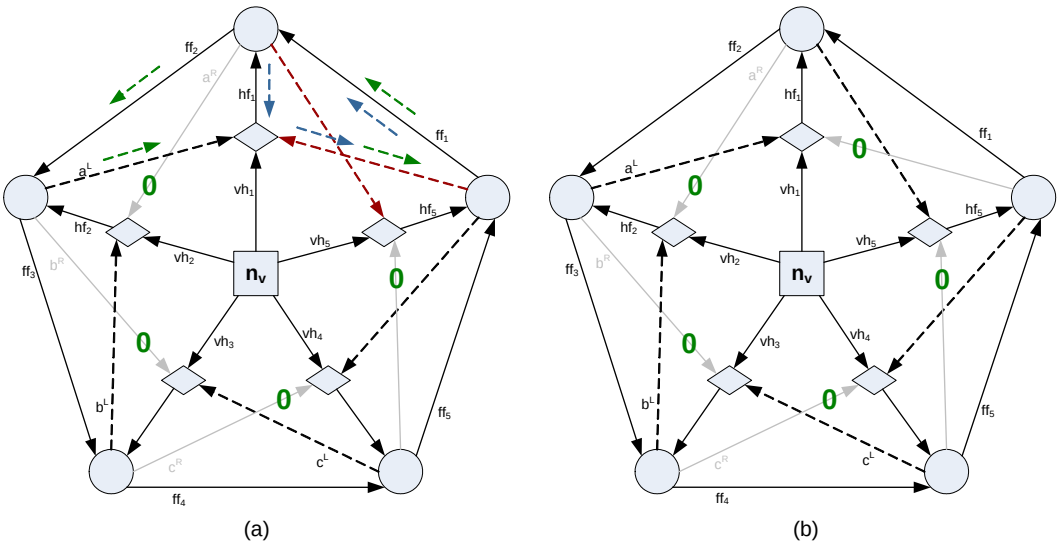

Abbildung 5.17: Bei dem letzten kritischen Bündel findet eine *Typ 2* Korrektur statt.

Die Gesamtkosten, die bei dieser Korrektur entstehen, sind

$$\mathrm{cost}_{\mathrm{L},3}\left(\mathcal{B}\left(v\right)\right) = \sum_{(a^L,a^R)\in\mathcal{B}(v)} 2\cdot x\left(a^R\right).$$

Diese Fall 3-Korrektur kann auf die gleiche Weise (nur Spiegelverkehrt) in die andere Richtung (von A aus nach rechts) durchgeführt werden und hat in dem Fall Gesamtkosten

$$\mathrm{cost}_{\mathrm{R},3}\left(\mathcal{B}\left(v\right)\right) = \sum_{(a^L,a^R)\in\mathcal{B}(v)} 2\cdot x\left(a^L\right)$$

Bemerkung 5.3 *Wir können nun mit den bisher eingeführten Korrektur-Methoden alle kritischen Bündel in einer nicht ganzzahligen Lösung in unkritische Bündel transformieren.*

Bei diesen Transformationen haben wir immer zwei Möglichkeiten: Bei *Typ 1* und *2* können wir eine Folge $\mathcal{K}$ von benachbarten, kritischen Bündeln

von links nach rechts (im UZS) korrigieren:

$$\textit{Typ1} \text{ Korrektur hat Kosten:} \quad \mathrm{cost}_{\mathrm{L},1}(\mathcal{K}) = \sum_{(a^L,a^R)\in\mathcal{K}} x\left(a^L\right)$$
$$\textit{Typ 2} \text{ Korrektur hat Kosten:} \quad \mathrm{cost}_{\mathrm{L},2}(\mathcal{K}) = \sum_{(a^L,a^R)\in\mathcal{K}} 2\cdot x\left(a^R\right)$$

oder von rechts nach links (gegen UZS):

$$\textit{Typ1} \text{ Korrektur hat Kosten:} \quad \mathrm{cost}_{\mathrm{R},1}(\mathcal{K}) = \sum_{(a^L,a^R)\in\mathcal{K}} x\left(a^R\right)$$
$$\textit{Typ 2} \text{ Korrektur hat Kosten:} \quad \mathrm{cost}_{\mathrm{R},2}(\mathcal{K}) = \sum_{(a^L,a^R)\in\mathcal{K}} 2\cdot x\left(a^L\right).$$

Bei *Typ 3* können wir den Knoten n_v im Uhrzeigersinn umkreisen

$$\text{mit Kosten:} \quad \mathrm{cost}_{\mathrm{R},3}(\mathcal{B}(v)) = \sum_{(a^L,a^R)\in\mathcal{B}(v)} 2\cdot x\left(a^L\right)$$

oder umgekehrt

$$\text{mit Kosten:} \quad \mathrm{cost}_{\mathrm{L},3}(\mathcal{B}(v)) = \sum_{(a^L,a^R)\in\mathcal{B}(v)} 2\cdot x\left(a^R\right).$$

In jedem Fall wählen wir die Richtung aus, in der die geringeren Kosten entstehen. Dadurch bleiben die gesamten zusätzlichen Kosten, die für diese Korrekturen höchstens entstehen können, begrenzt. Ihre Abschätzung erfolgt später in Abschn. 5.4.4.

Steht einmal fest, welche der oben beschriebenen Korrekturen durchgeführt werden soll, sind damit auch die zu sperrenden Bündelkanten festgelegt. Wir setzen deren obere Kapazitätsgrenze auf 0 und erhalten somit ein gewöhnliches MCF-Netzwerk, auf dem ein kostenminimaler Fluss in polynomialer Zeit berechnet werden kann (CS3). Auf diesem Netzwerk generieren wir eine neue, ganzzahlige Startlösung und lösen die LP-Relaxierung von neuem (CS4). Die Lösung ist jetzt ganzzahlig und kann in eine korrekte Kandinsky Repräsentation transformiert werden.

Die durch die Korrekturen ermittelten Flüsse brauchen wir weder für die darauffolgenden Schritte CS3 und CS4 noch für die Kostenabschätzungen. Es ist also nicht notwendig, sie explizit durchzuführen. Es kommt nur darauf an, welche Kanten in den einzelnen Bündeln gesperrt werden und welche Kostenerhöhungen dadurch für das neue MCF-Netzwerk entstehen. Das lässt sich mit den Korrekturen leichter erklären.

5.4.3 Ein Beispiel

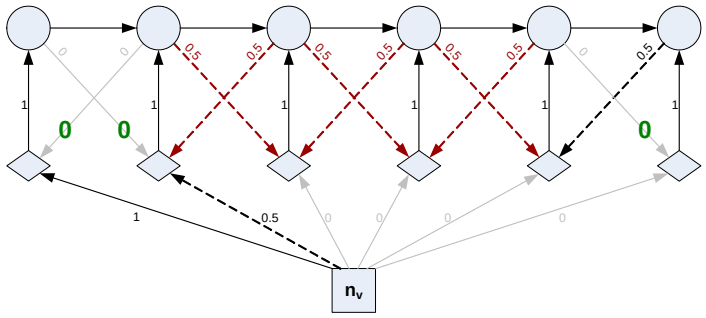

Abbildung 5.18: Hier entscheiden wir uns für eine *Typ 1* Korrektur im UZS.

In Abb. 5.18 sind drei benachbarte, kritische Bündel zu sehen, für die links beide Fälle zugleich vorliegen und rechts Fall 2 vorliegt. Eine *Typ1* Korrektur im UZS hat Kosten **1.5**, während die Kosten einer *Typ 2* Korrektur im oder gegen den UZS bei **3** liegen. Es wird also eine *Typ 1* Korrektur im UZS durchgeführt. Danach sind die drei Bündel unkritisch und ihre zu sperrenden Kanten somit festgelegt (siehe Abb. 5.19).

Für die zwei benachbarten, kritischen Bündel in Abb. 5.20 liegt auf beiden Seiten Fall 2 vor. Da beide Richtungen die gleichen Korrekturkosten haben ($2 \cdot 0.5 + 2 \cdot 0.5 = \mathbf{2}$), kann eine beliebige ausgewählt werden (Abb. 5.21).

An dieser Stelle kommt die Frage auf, ob unser Auswahlkriterium nicht *verfeinert* werden kann, sodass die Wahrscheinlichkeit, die richtige Wahl getroffen zu haben, erhöht wird. Dazu kommen wir in Abschn. 5.4.5 zurück.

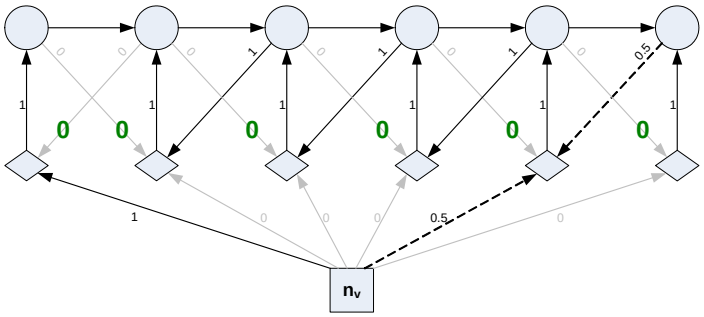

Abbildung 5.19: Nach der Korrektur liegen die zu sperrenden Bündelkanten fest.

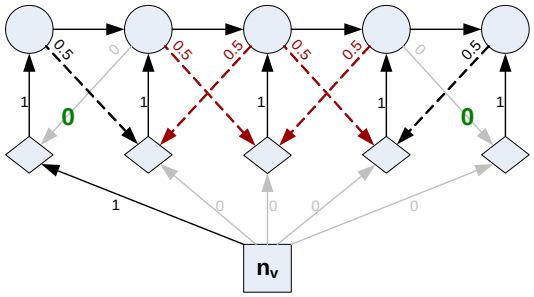

Abbildung 5.20: Beide Richtungen haben dieselben Korrekturkosten.

5.4.4 Eine Analyse des Algorithmus

Terminierung

Alle kritischen Bündelfolgen werden nacheinander wie in Abschn. 5.4.2 beschrieben, in unkritische Bündel transformiert. Dabei ist für jedes kritische Bündel genau eine Operation erforderlich. Die Termination ist selbstverständlich.

Korrektheit

Es ist zu zeigen, dass im 2. Schritt des Alg. 5.3 tatsächlich alle kritischen Bündel mit Hilfe der oben beschriebenen Korrekturen in unkritische Bündel transformiert werden.

Es ist klar, dass eine Folge von benachbarten, kritischen Bündeln für sich

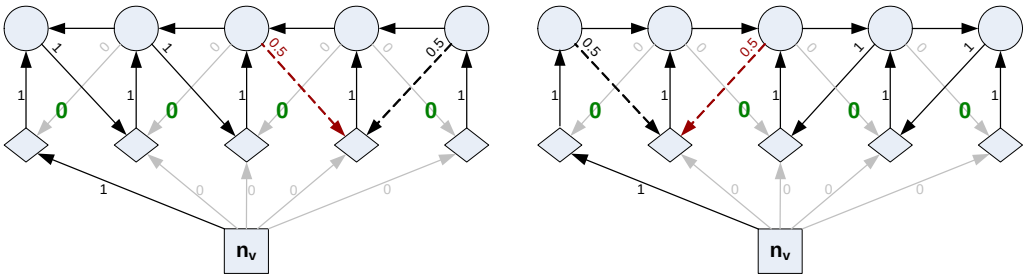

Abbildung 5.21: Beide Korrekturen haben dieselben Kosten.

ohne Probleme korrigiert werden kann. Da solche kritischen Bündelfolgen nacheinander bearbeitet werden, stellt sich die Frage, ob und inwiefern die Korrektur einer Bündelfolge die einer anderen beeinflusst.

Sind zwei Bündelfolgen um unterschiedliche Knoten angeordnet, hat die Korrektur der einen keinen Einfluss auf die Korrektur der anderen, da die jeweiligen Kreisflüsse keine gemeinsamen Kanten mit beschränkter Kapazität haben, die eine Augmentierung entlang beider Pfade verhindern könnten. Ein Problem kann nur dann auftreten, wenn beide Bündelfolgen um denselben Knoten n_v angeordnet sind und -eine von rechts, eine von links- an dasselbe nicht kritische Bündel angrenzen.Hat dabei eines der Kanten des nicht kri-

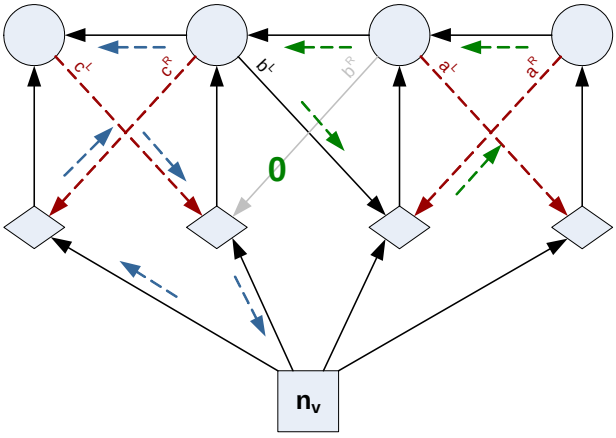

Abbildung 5.22: Zwei kritische Bündel, die an dasselbe nicht kritische Bündel angrenzen.

tischen Bündels einen positiven Flusswert, liegt für eines der benachbarten

Bündel Fall 1, für den anderen Fall 2 vor (Abb. 5.22). Die korrigierenden Kreisflüsse der Bündel A (grün eingezeichnet) und C (blaue Pfeile) schneiden sich nicht, was deshalb auch zu keinem Problem führt.

Haben hingegen beide Kanten des Bündels B den Flusswert 0, liegen für A und C die Fälle 1 und 2 zugleich vor (Abb. 5.23). Es dürfen jedoch nicht beide

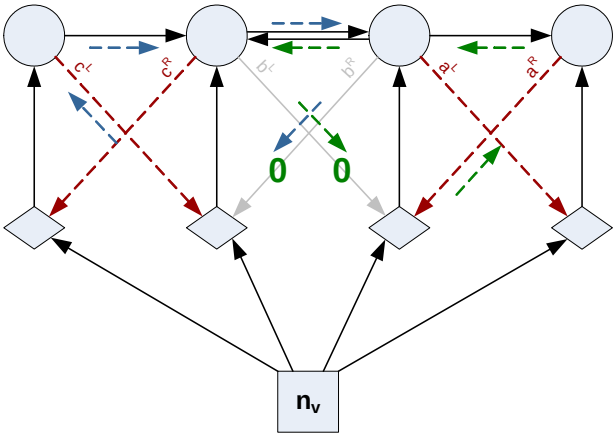

Abbildung 5.23: Zwei kritische Bündel, die an dasselbe nicht kritische Bündel angrenzen, dessen Kanten beide den Flusswert 0 haben.

mit einer *Typ 2* Korrektur repariert werden, da dadurch ein neues kritisches Bündel (B) entstehen würde. Dieser Fall tritt in unserem Algorithmus auch nicht ein: Wird für eines der kritischen Bündel, z.B. A, eine *Typ 2* Korrektur durchgeführt, ist nach den entsprechenden Augmentierungen $x\left(b^L\right) \neq 0$. Danach liegt für C nur mehr Fall 1 vor, womit eine *Typ 2* Korrektur für dieses Bündel ausgeschlossen wird.

Die Gütegarantie

Satz 9 *Während der Bündelkorrekturen wird der Zielfunktionswert des LPs höchstens auf das doppelte erhöht.*

Beweis. Sei x die Lösung der LP-Relaxierung des Kandinsky MCF Problems mit dem Zielfunktionswert $z(x)$. Da auf jeder Bündelkante der Kostenfaktor

1 vorliegt, gilt jedenfalls

$$z(x) \geq \sum_i \left(x\left(a_i^L\right) + x\left(a_i^R\right) \right) \tag{5.15}$$

wobei über alle Bündel $\left(a_i^L, a_i^R\right)$ addiert wird.
Sei $\mathcal{B}_K = \{\mathcal{K}_1, \mathcal{K}_2, ..., \mathcal{K}_l\}$ die Menge aller kritischen Bündel in dem nicht ganzzahligen Teilnetzwerk, wobei jedes $\mathcal{K}_j = \left(B_1, B_2, ..., B_{k_j}\right)$ eine Folge von benachbarten kritischen Bündeln ist. Bei seinen Bündeln $B_i = \left(a_i^L, a_i^R\right)$ verzichten wir zur Vereinfachung der Schreibweise auf eine weitere Indizierung mit j. Für alle möglichen Fälle bezüglich der Korrektur (Fälle 1, 2 oder 3), betrachten wir die Kosten beim Abarbeiten von links und von rechts; da wir immer die günstigere Richtung auswählen, ist davon jeweils das Minimum zu nehmen:

- Links und rechts von $\mathcal{K}_j$ liegt Fall 2 vor:

$$\text{cost}\left(\mathcal{K}_j\right) = \min\left\{ \sum_{i=1}^{k_j} 2 \cdot x\left(a_i^L\right),\ \sum_{i=1}^{k_j} 2 \cdot x\left(a_i^R\right) \right\}$$

- Links von $\mathcal{K}_j$ liegt Fall 2 vor, rechts Fall 1:

$$\text{cost}\left(\mathcal{K}_j\right) = \min\left\{ \sum_{i=1}^{k_j} x\left(a_i^R\right),\ \sum_{i=1}^{k_j} 2 \cdot x\left(a_i^R\right) \right\}$$

- Links von $\mathcal{K}_j$ liegt Fall 1 vor, rechts Fall 2:

$$\text{cost}\left(\mathcal{K}_j\right) = \min\left\{ \sum_{i=1}^{k_j} 2 \cdot x\left(a_i^L\right),\ \sum_{i=1}^{k_j} x\left(a_i^L\right) \right\}$$

- Auf beiden Seiten von $\mathcal{K}_j$ liegt Fall 1 vor:

$$\text{cost}\left(\mathcal{K}_j\right) = \min\left\{ \sum_{i=1}^{k_j} x\left(a_i^R\right),\ \sum_{i=1}^{k_j} x\left(a_i^L\right) \right\}$$

- Für $\mathcal{K}_j$ liegt Fall 3 vor ($\mathcal{K}_j$ ist die zyklische Liste aller Bündel um einen Knoten n_v):

$$\text{cost}\left(\mathcal{K}_j\right) = \min\left\{\sum_{i=1}^{k_j} 2 \cdot x\left(a_i^R\right),\ \sum_{i=1}^{k_j} 2 \cdot x\left(a_i^L\right)\right\}$$

  Für jeden dieser Fälle gilt die Abschätzung:

$$\text{cost}\left(\mathcal{K}_j\right) \leq \sum_{i=1}^{k_j} \left(x\left(a_i^L\right) + x\left(a_i^R\right)\right)$$

  Für die Gesamtkosten der Korrektur sind diese Kosten über alle kritischen Bündel zu addieren:

$$\text{cost}\left(\mathcal{B}_K\right) = \sum_{j=1}^{l} \text{cost}\left(\mathcal{K}_j\right) \leq \sum_{j=1}^{l} \sum_{i=1}^{k_j} \left(x\left(a_i^L\right) + x\left(a_i^R\right)\right)$$

Sei $\hat{x}$ der Fluss nach der Korrektur aller kritischen Bündel. Für seine Kosten $z(\hat{x})$ gilt:

$$z(\hat{x}) \leq z(x) + \text{cost}\left(\mathcal{B}_K\right) \leq 2 \cdot z(x)$$

Das ist die Kostenabschätzung für das korrigierte Netzwerk, das allerdings weiterhin nicht ganzzahlige Flüsse enthalten kann, aber nur unkritische Bündel. Im 4. Schritt des Alg. 5.3 werden auf diesem Netzwerk Kanten gesperrt, die ohnehin keinen Fluss enthalten, und danach ein optimaler kostenminimaler Fluss berechnet, dessen Kosten jedenfalls $\leq z\left(\hat{x}\right)$ sind. ■

5.4.5 Verbesserungen

Kombinieren von Links- und Rechts-Korrektur

Sei $\mathcal{K} = (B_1, B_2, ..., B_k)$ eine Folge von k benachbarten, kritischen Bündeln, die auf beiden Seiten an ein unkritisches Bündel angrenzt. Bisher haben wir stets die gesamte Folge in dieselbe Richtung (im oder gegen den UZS) korrigiert. Es kann allerdings günstiger sein, die ersten j Bündel von links aus, die restlichen $k-j$ Bündel von rechts aus "abzuarbeiten". Die Gesamtkosten einer solchen *kombinierten* Korrektur sind gegeben durch:

$$\mathrm{ccost}_j(\mathcal{K}) = \sum_{i=1}^{j} \mathrm{cost}_{\mathrm{L}}(B_i) + \sum_{i=j+1}^{k} \mathrm{cost}_{\mathrm{R}}(B_i)$$

Neben den Kosten, die bei der Anwendung einer Links- bzw. Rechts-Korrektur auf alle Bündel entstehen, berechnen wir nun zusätzlich die Kosten $\mathrm{scost}_j(\mathcal{K})$ für alle $j \in 1, ..., k-1$ und führen danach die Korrektur mit den geringsten Kosten aus:

$$\mathrm{cost}\,(\mathcal{K}) = \min\left\{\mathrm{cost}_{\mathrm{L}}(\mathcal{K}),\ \mathrm{cost}_{\mathrm{R}}(\mathcal{K}),\ \min_{j=1}^{k-1}\{\mathrm{scost}_j(\mathcal{K})\}\right\}$$

Betrachten wir zum Beispiel die drei kritischen Bündel in Abb. 5.24. Eine Links-Korrektur vom *Typ 1* hätte die Kosten **1.8**, eine Rechts-Korrektur vom *Typ 1* die Kosten **1.2**. Wendet man hingegen auf das erste Bündel von links eine Links-Korrektur und auf die restlichen zwei eine Rechts-Korrektur an, so liegen die Gesamtkosten bei **0.6**.

Einige Sonderfälle

Bei der Korrektur kritischer Bündelfolgen, die auf beiden Seiten an nicht kritische Bündel angrenzen, haben wir bisher zwei Fälle in betracht gezogen: Fall 1 und Fall 2. Die Kosten der zugehörigen Korrekturen (Typ 1 und Typ 2) betrachten wir nun als obere Schranken. Es *können* nämlich Sonderfälle vorliegen, in denen eine günstigere Korrektur möglich ist. Wir verfeinern nun unseren Algorithmus, indem wir bei einigen ausgewählten Sonderfällen

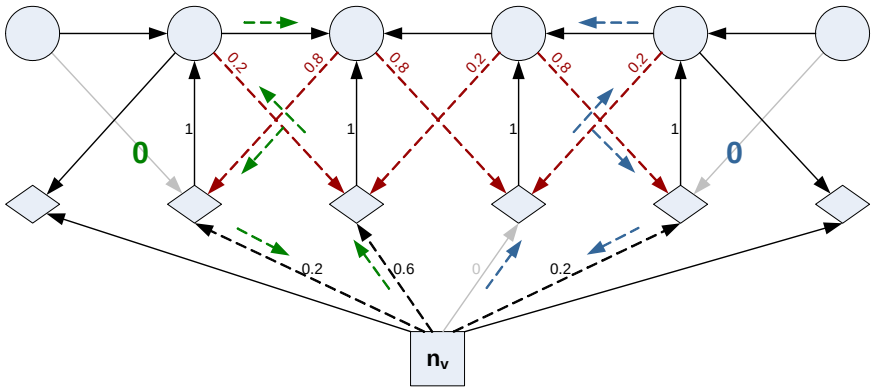

Abbildung 5.24: Kombinieren von Links- und Rechts-Korrektur.

eine solche Verbesserung implementieren. Es wird aber nicht angestrebt, alle möglichen Verbesserungen zu erfassen.

Die theoretische Schranke von 2 für die Güte des Algorithmus bleibt unverändert, da im schlimmsten Fall keiner dieser Sonderfälle vorliegt und für alle kritischen Bündel weiterhin nur eine Typ 1 (bzw. Typ 2) Korrektur möglich ist.

Sonderfälle zu Fall 1 Sei $A = \left(a^L, a^R\right)$ ein kritisches Bündel, das rechts neben einem nicht kritischen Bündel $B = \left(b^L, b^R\right)$ mit $x\left(b^L\right) = 0$ (Fall 1) steht.

Fall 1.1 Wir setzen voraus, dass $x\left(vh'\right) > 0 \quad \wedge \quad x\left(b^R\right) < 1$ ist (siehe Abb. 5.25).
In dem Fall gilt für die Kapazität des mit blauen Pfeilen gekennzeichneten Kreises:

$$cap_0 = \min\left\{1 - x(b^R),\ x\left(vh'\right),\ x\left(a^L\right)\right\} > 0 \tag{5.16}$$

und der Fluss kann entlang dieses 0-Kosten Kreises um cap_0 Einheiten augmentiert werden. Ist $x\left(a^L\right) > cap_0$, wird der Rest des Flusses mittels einer *Typ 1* Korrektur rückgängig gemacht.
Dadurch, dass weniger Fluss entlang des Pfades $\left(a^R, -vh\right)$ geschickt wird, erhöht sich die Kapazität des 0-Kosten Kreises für das rechts an A anschlie-

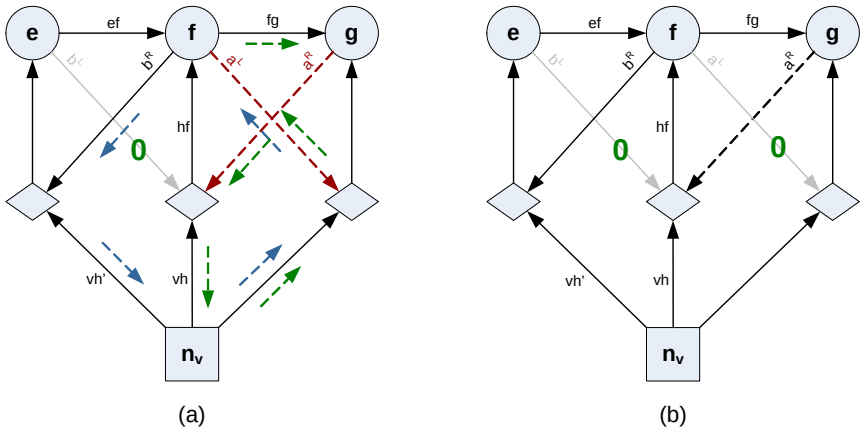

Abbildung 5.25: Fall 1.1: $x\left(vh'\right) > 0 \quad \wedge \quad x\left(b^R\right) < 1.$

ßende Bündel, sodass auch für dieses Fall 1.1 vorliegt. Auf diese Weise werden die Korrekturkosten aller restlichen Bündel rechts von A, im Vergleich zu einer Typ 1 Korrektur, verringert.

Fall 1.2 Wir setzen voraus, dass $x\left(c^L\right) = 0 \ \wedge \ x(c^R) < 1 \wedge x\left(vh''\right) > 0$ ist.

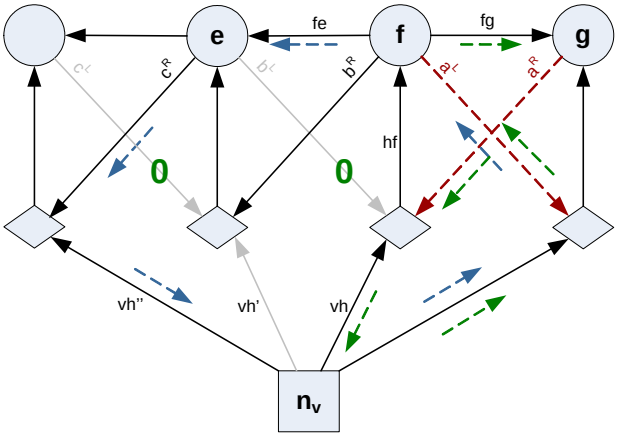

Abbildung 5.26: Fall 1.2

Dann hat der in Abb. 5.26 mit blauen Pfeilen gekennzeichnete Kreis mit

Kosten 1 die Kapazität

$$cap = \min\left\{x\left(vh''\right),\ 1 - x\left(c^R\right),\ x\left(a^L\right)\right\} > 0.$$

Daher kann der Fluss entlang dieses Kreises um cap Einheiten augmentiert werden. Ist $x\left(a^L\right) > cap$, wird der Rest des Flusses mittels einer *Typ 1* Korrektur rückgängig gemacht.
Hier wird ebenfalls weniger Fluss entlang des Pfades $\left(a^R, -vh\right)$ geschickt, was aufgrund derselben Überlegungen wie in Fall 1.1 zu Ersparnissen bei der Korrektur der restlichen Bündel rechts von A führt.

Fall 1.3 Wir setzen voraus, dass $x\left(b^R\right) \leq x\left(vh\right)$ ist.

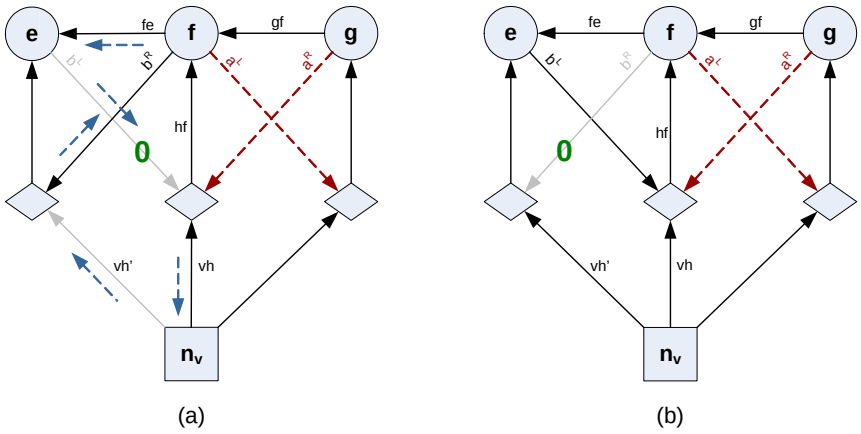

Abbildung 5.27: Fall 1.3: $x\left(b^R\right) \leq x\left(vh\right)$.

Dann kann der Fluss entlang des in Abb. 5.27,(a) blau gekennzeichneten Kreises um $x\left(b^R\right)$ Einheiten augmentiert werden. Danach liegt links von Bündel A Fall 2 vor (Abb. 5.27,(b)), wodurch eine Alternative eröffnet wird, die unter Umständen besser ist.

Sonderfälle zu Fall 2 Sei $A = \left(a^L, a^R\right)$ ein kritisches Bündel, das rechts von einem nicht kritischen Bündel $B = \left(b^L, b^R\right)$ mit $x\left(b^R\right) = 0$ (Fall 2) steht.

Fall 2.1 Wir setzen voraus, dass $x\left(a^L\right) \leq x\left(vh\right)$ ist. In dem Fall kann

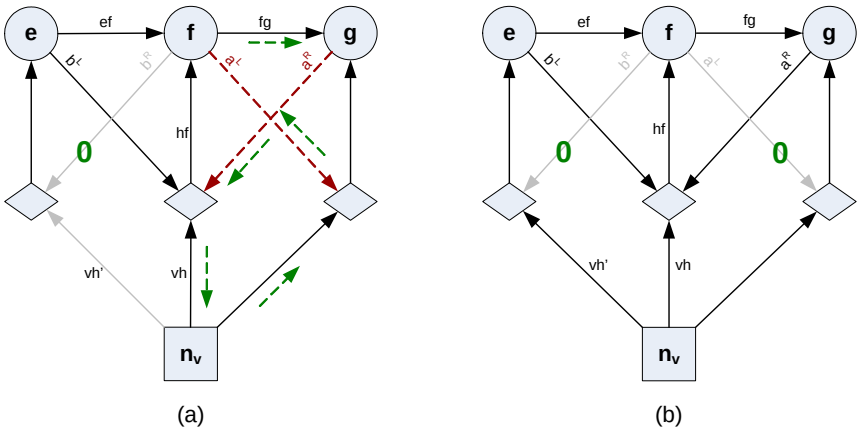

Abbildung 5.28: Fall 2.1: $x\left(a^L\right) \leq x\left(vh\right)$.

auf das Bündel A eine *Typ 1* Korrektur angewendet werden, obwohl links Fall 2 vorliegt (Abb. 5.28,(a)). Für den rechten Nachbarn von A liegt danach Fall 1 vor (Abb. 5.28,(b)).

Fall 2.2 Wir setzen voraus, dass $x\left(c^R\right) = x\left(c^L\right) = 0 \quad \wedge \quad x\left(b^L\right) \leq x\left(vh''\right)$ ist.

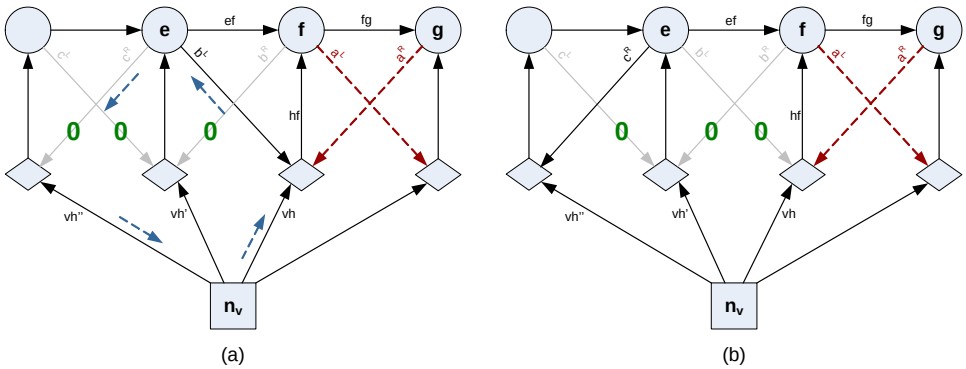

Abbildung 5.29: Fall 2.2: $x\left(c^R\right) = x\left(c^L\right) = 0 \quad \wedge \quad x\left(b^L\right) \leq x\left(vh''\right)$.

Dann kann der Fluss entlang des in Abb. 5.29,(a) blau dargestellten 0-Kosten Kreises um $x\left(b^L\right)$ Einheiten augmentiert werden. Für das Bündel A ist danach auch eine *Typ 1* Korrektur möglich, da der Fluss auf Kante b^L

gleich Null ist Abb. 5.29,(b).

Fall 2.3 Wir setzen voraus, dass $x\left(b^L\right) \leq x\left(vh'\right)$ ist.

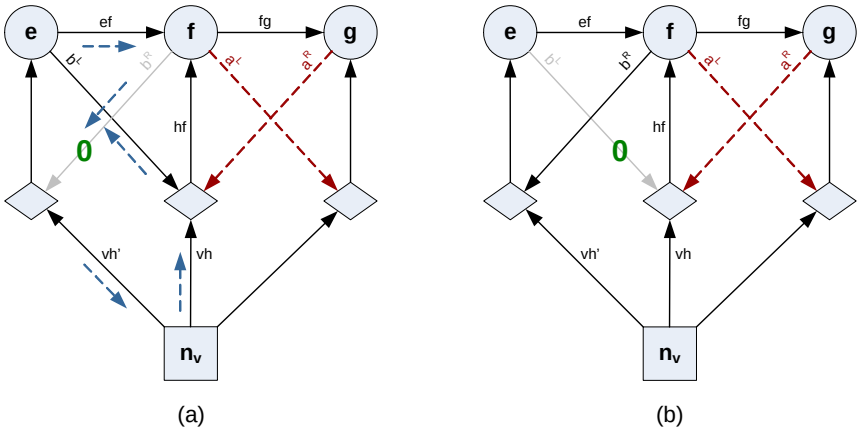

Abbildung 5.30: Fall 2.3: $x\left(b^L\right) \leq x\left(vh'\right)$.

Dann können wir den Fluss entlang des in Abb. 5.30,(a) blau markierten Kreises augmentieren. Danach liegt links von A Fall 1 vor (Abb. 5.30,(b)).

5.5 Vergleich von Cyclic-Shift und Eiglsperger's Ansatz

Bei dem Kandinsky Min–Cost Flow ILP spielen vier Einschränkungen (bzw. Nebenbedingungen) eine Rolle:

1. Flusserhaltung: $\sum_{a\in \mathrm{Out}(n)} x(a) - \sum_{a\in \mathrm{In}(n)} x(a) = b(n) \quad \forall n \in N$
2. Kapazitätsbeschränkung (Kanten): $\mathrm{lcap}(a) \leq x(a) \leq \mathrm{ucap}(a) \quad \forall a \in A$
3. Die Bündelkapazitätsbedingung: $x(a^L) + x(a^R) \leq 1 \quad \forall \left(a^L, a^R\right) \in A^{FH}$
4. Ganzzahligkeit der Flusswerte

Das gesuchte Optimum ist ein Fluss mit minimalen Kosten, der alle vier Bedingungen erfüllt.

Der Cyclic-Shift Algorithmus und der Transformationsalgorithmus von Eiglsperger haben einen grundsätzlichen Unterschied:

- Bei CS erhält man die erste, jedoch unzulässige Lösung vor der Korrektur (CS1) durch das Weglassen der Ganzzahligkeitsbedingung. Es gelten also nur die Bedingungen 1, 2 und 3.
- Bei TE wird bei der Berechnung der ersten, unzulässigen Lösung (TE1) die Bündelkapazitätsbeschränkung 3 weggelassen. Die Ganzzahligkeit der Lösung entsteht dadurch automatisch, bedeutet also keine Einschränkung. Wirksam sind daher nur die Bedingungen 1 und 2.

Wegen dieser Abschwächung der Einschränkungen bei TE gilt für die Lösungen von CS1 und TE1:

$$z_{TE1} \leq z_{CS1} \leq z_{opt}.$$

Das bedeutet, dass die Ausgangssituation vor der Korrektur bei CS näher am Optimum liegt, als die bei TE. Eine Auswirkung dieser *Nähe* werden wir im nächsten Kapitel, bei den Testergebnisse feststellen.

Kapitel 6

Testergebnisse und Auswertungen

Wir testen unseren Cyclic-Shift Algorithmus (Alg. 5.3), den Transformationsalgorithmus von Eiglsperger (Alg. 5.1) und den modifizierten sukzessiven Transformationsalgorithmus (Alg. 5.2). Wir lösen außerdem für alle Instanzen das ganzzahlige Lineare Programm des zugehörigen Kandinsky MCF Problems (Abschn. 5.4.1), um die Näherungslösungen der drei Algorithmen mit den exakten Lösungen vergleichen zu können.

6.1 Implementierung und Testumgebung

Unsere Implementierung des Cyclic-Shift (*CS*) Algorithmus besteht grundsätzlich aus vier Stufen:

CS1 Das Lösen der LP-Relaxierung des KMCF-ILPs.

CS2 Das Korrigieren der kritischen Bündel.

CS3 Das Sperren einer Kante aus jedem Bündel.

CS4 Das erneute Lösen der LP-Relaxierung (entspricht gewöhnlichem MCF und liefert ganzzahlige Lösung).

Der Transformationsalgorithmus von Eiglsperger (*TE*) ist ebenfalls als vierstufiges Verfahren implementiert:

TE1 Das Lösen der LP-Relaxierung des KMCF-ILPs ohne die Bündelkapazitätsbeschränkung (entspricht gewöhnlichem MCF und liefert ganzzahlige Lösung, die allerdings überfüllte Bündel enthalten kann).

TE2 Das Korrigieren der überfüllten Bündel mittels geeigneter Transformationen.

TE3 Das Sperren einer Kante aus *jedem* Bündel.

TE4 Das erneute Lösen der LP-Relaxierung des KMCF-ILPs (entspricht ebenfalls gewöhnlichem MCF und liefert eine ganzzahlige Lösung des KMCF Problems).

Der auf S. 67 definierte Transformationsalgorithmus berechnet in 2 Schritten (TE1 und TE2) eine Näherungslösung mit dem Zielfunktionswert z_{TE2}. Wir sperren in unserer Implementierung jene Bündelkanten (d.h. setzen ihre obere Kapazitätsgrenze auf Null), die in dieser Näherungslösung den Flusswert 0 besitzen und lösen das modifizierte Lineare Programm von neuem. Da nun keine überfüllten Bündel entstehen können erhalten wir eine gültige Näherungslösung mit dem Zielfunktionswert $z_{TE} \leq z_{TE2}$.

In der Implementierung des modifizierten Sukzessiven Transformationsalgorithmus (ST) werden die folgenden Schritte solange wiederholt, bis die Lösung in Schritt ST1 keine überfüllten Bündel enthält:

ST1 Das Lösen der LP-Relaxierung des KMCF-ILPs ohne die Bündelkapazitätsbeschränkung
Enthält die Lösung überfüllte Bündel: weiter zu ST2.
Sonst: weiter zu ST4.

ST2 Korrigieren der überfüllten Bündel durch geeignete Transformationen.

ST3 Das Sperren *jener Bündelkanten*, die vor der Transformation den Flusswert 1 und danach den Flusswert 0 besitzen. Freigabe der geperrten

Bündelkanten, die nach der Transformation den Flusswert 1 besitzen. Zurück zu ST1.

ST4 Abbruch mit einer gültigen Näherungslösung des KMCF Problems.

Zum Lösen der ganzzahligen Linearen Programme (KMCF-ILP) benutzen wir den *Mixed Integer Optimizer*, für das Lösen der LPs in den Schritten TE1, TE4 und ST1 den *Network Optimizer* von ILOG CPLEX 8.1. Die LPs in den Schritten CS1 und CS4 werden mit der Standardeinstellung gelöst, in der CPLEX den Optimierer bestimmt. Alle Tests werden auf einem Pentium IV 2.8GHz Rechner mit 2GB Arbeitsspeicher durchgeführt.

Für unsere Tests verwenden wir neben den Rome Graphen [DGLTTV95] zusätzlich insgesamt 14000 zufällig generierte, planare Graphen aus unterschiedlichen Klassen sowie 29 vollständige Graphen.

6.2 Rome Graphen

Die Rome Graphen Sammlung besteht aus 11529 ungerichteten Graphen. Die Knotenanzahl n variiert zwischen 10 und 100, die Dichte m/n zwischen 0.5 und 1.9, wobei m die Kantenanzahl angibt. Die durchschnittliche Dichte liegt bei 1.22.

8249 dieser Graphen sind nicht planar und werden planarisiert. Dabei wird der Graph zunächst durch das Entfernen von Kanten in einen planaren Graphen transformiert. Anschließend werden die entfernten Kanten wieder so eingefügt, dass die Anzahl der Kreuzungen gering ist. Dabei wird das Subgraph Planarizer Modul der Algorithmenbibliothek [AGD] benutzt. Die Kreuzungspunkte werden durch künstliche Knoten ersetzt, sodass der resultierende Graph planar ist. Abb. 6.1 zeigt, wie sich die Anzahl der Knoten und die durchschnittliche Dichte der Rome Graphen nach der Planarisierung verhalten. (Man beachte, dass auf der x-Achse ab $n = 204$ nicht jede ganze Zahl repräsentiert ist, weshalb die Intervallabstände danach nicht mehr gleichmäßig sind. Nach einem Graphen mit 308 Knoten enthält der nächstkleine Graph beispielsweise nur 254 Knoten.)

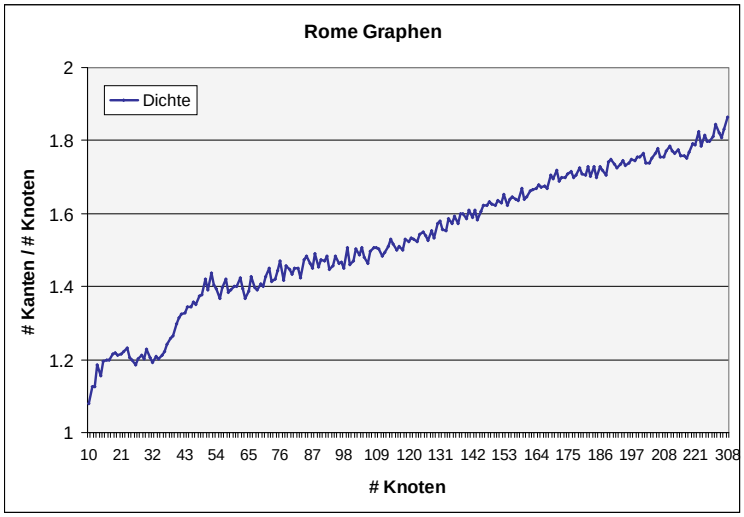

Abbildung 6.1: Die durchschnittliche Dichte der planarisierten Rome Graphen in Bezug auf die Anzahl der Knoten im planarisierten Graph.

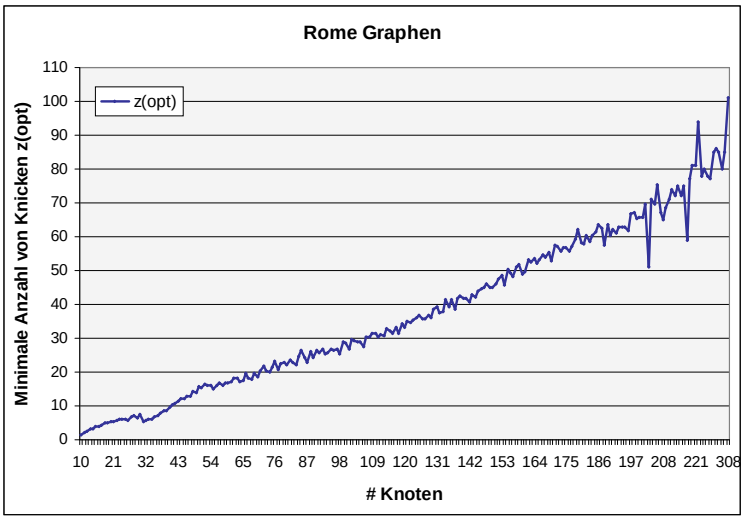

Abbildung 6.2: Die durchschnittliche minimale Anzahl von Knicken (z_{opt}) bei den Rome Graphen in Bezug auf die Anzahl der Knoten im planarisierten Graph.

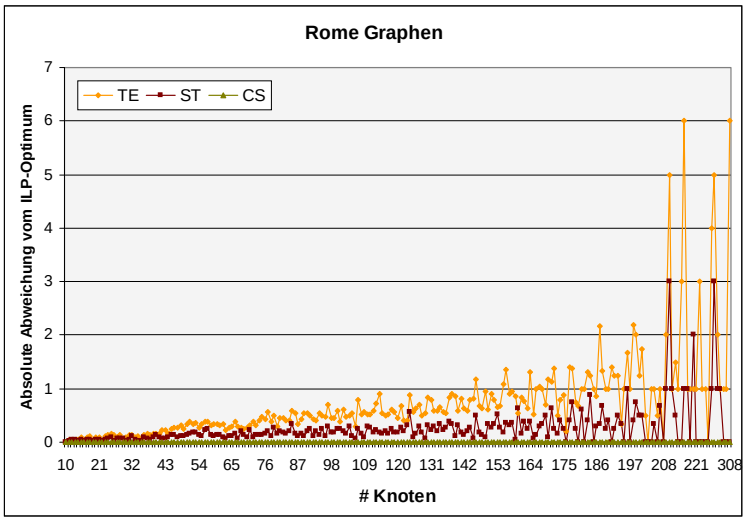

Abbildung 6.3: Die durchschnittliche absolute Abweichung der TE, ST und CS Lösungen vom ILP-Optimum bei den Rome Graphen in Bezug auf die Anzahl der Knoten im planarisierten Graph.

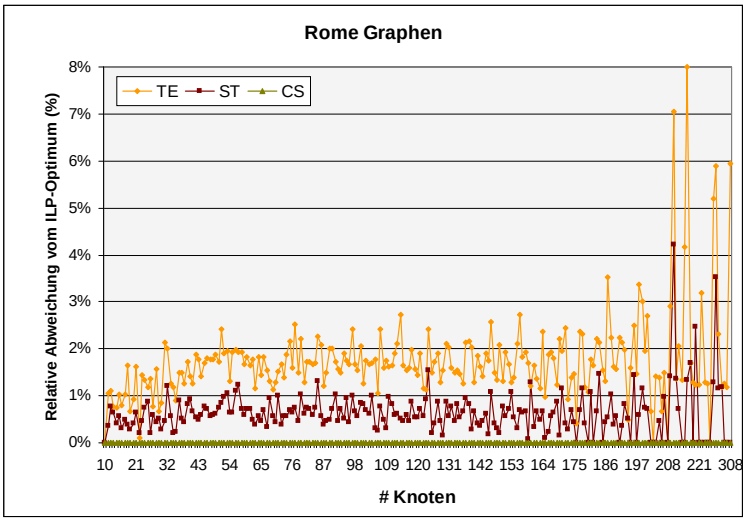

Abbildung 6.4: Die durchschnittliche relative Abweichung der TE, ST und CS Lösungen vom ILP-Optimum bei den Rome Graphen mit $z_{opt} \neq 0$ in Bezug auf die Anzahl der Knoten im planarisierten Graph.

6.2.1 Qualitätsanalyse

Der Cyclic-Shift Algorithmus liefert für alle Rome Instanzen die optimale Lösung. Dabei ist für 10783 Graphen die in Schritt CS1 berechnete Lösung bereits ganzzahlig. Bei 675 der restlichen 746 Graphen mit nicht ganzzahliger CS1 Lösung entstehen in Schritt CS2 keine zusätzlichen Korrekturkosten, bei weiteren 61 Graphen liegen die Korrekturkosten unter 1. Für diese 736 Graphen ist die Lösung in Schritt CS4 daher mit Sicherheit optimal ($z_{CS1} \leq z_{opt} = z_{CS4} \leq z_{CS1} + k, 0 < k < 1$). Die restlichen zehn Graphen (Tab. 6.2) sind die einzigen, für die eine nicht optimale Lösung überhaupt in Frage kommt. Wie gesagt werden diese aber ebenfalls optimal gelöst.

	CS1 ganzz.	CS1 nicht ganzzahlig		
Korrekturkosten k	-	$k = 0$	$0 < k < 1$	$k \geq 1$
# Graphen	10783	675	61	10
∅ # Knicke (z_{opt})	17.73	31.8	32.77	40.9
∅ # Knoten (plan.Gr.)	64	99.7	98.3	125.5
∅ Dichte	1.36	1.52	1.52	1.59
∅ # hochgr. Knoten	4.4	7.6	7.3	9.1
∅ max. Knotengrad	6.6	7.9	8.5	9.2
∅ # Bündel	177	299	296	396
∅ # nicht g. Flussw.	–	14.73	19.26	25.4
∅ # kritischer Bündel	–	1.46	1.46	2.3

Tabelle 6.1: Vergleich einiger Parameter bezüglich der Lösung von CS.

Wir betrachten nun die 746 Graphen mit nicht ganzzahliger CS1 Lösung, da CS nur bei diesen Graphen in Einsatz kommt und eine Aussage über das Verhalten des Algorithmus dadurch erst möglich wird. Das Diagramm in Abb. 6.5 zeigt, dass die durchschnittliche Anzahl der kritischen Bündel bei diesen Graphen mit wachsender Knotenanzahl nicht zunimmt. Das erklärt die geringen Korrekturkosten (bei allen Instanzen unter 1.5) und das optimale Ergebnis des CS bei allen Instanzen. Denn offensichtlich hängen beide stark von der Anzahl der kritischen Bündel ab: Je mehr kritische Bündel, desto höher die Korrekturkosten, desto höher die Wahrscheinlichkeit einer nicht optimalen CS Lösung.

Es stellt sich nun die Frage, wovon die Anzahl der kritischen Bündel ab-

Instanz	n/m	Dichte	z_{opt}	k	max.Kn.gr.	# hochgr.Kn.
grafo3995	184/319	1.73	64	1	8	13
grafo4445	71/105	1.48	22	1	8	5
grafo6978	88/160	1.82	37	1	9	7
grafo7294	59/77	1.3	16	1	10	4
grafo8802	151/251	1.66	50	1	10	8
grafo9289	144/236	1.64	51	1	10	12
grafo9424	81/121	1.49	25	1.5	9	6
grafo10931	103/135	1.31	17	1	8	4
grafo10945	200/346	1.73	70	1	9	18
grafo10949	174/293	1.68	57	1	11	14

Tabelle 6.2: Die 10 Rome Graphen mit Korrekturkosten ≤ 1 (CS)

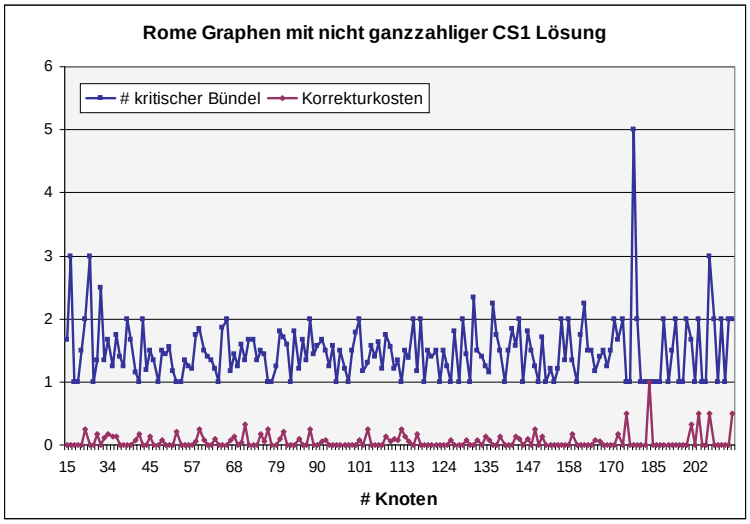

Abbildung 6.5: Die durchschnittliche Anzahl der kritischen Bündel in der CS1 Lösung und die durchschnittlichen Korrekturkosten bei den Rome Graphen mit nicht ganzzahliger CS1 Lösung in Bezug auf die Anzahl der Knoten im planarisierten Graph.

hängt. Dazu betrachten wir nun das Verhalten der durchschnittlichen Anzahl der kritischen Bündel in der CS1 Lösung bei *allen* Rome Graphen (Abb. 6.6): sie steigt von 0 bis zu maximal 3 an, während die durchschnittliche Anzahl aller Bündel im Netzwerk von ca.19 auf 1138 steigt. Die Größe der Graphen (und in Zusammenhang damit die Größe des Netzwerkes) hat offensichtlich keinen so starken Einfluss auf die Anzahl der kritischen Bündel.

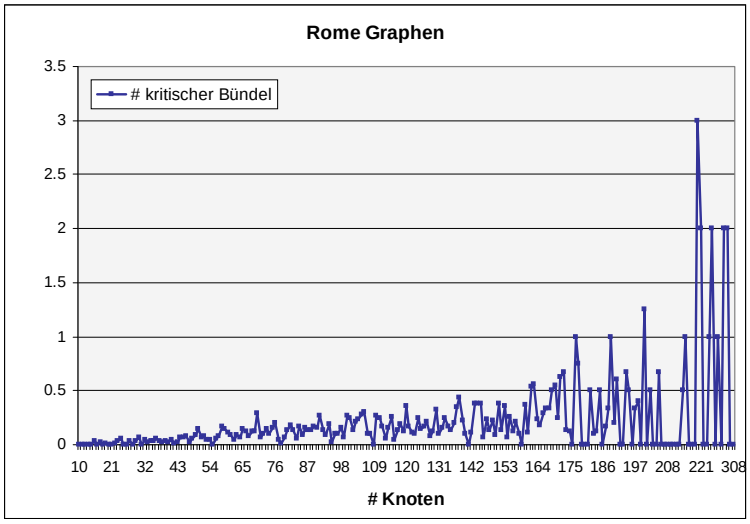

Abbildung 6.6: Die durchschnittliche Anzahl der kritischen Bündel in der CS1 Lösung bei den Rome Graphen in Bezug auf die Anzahl der Knoten im planarisierten Graph.

Der Tabelle 6.1 kann man entnehmen, dass mit den Korrekturkosten und der Anzahl der kritischen Bündel auch die durchschnittliche Dichte und der durchschnittliche Maximalgrad der Graphen eine ähnlich geringe Steigung zeigen (vergleiche auch die Abbildungen 6.6 und 6.7). Das lässt vermuten, dass ein Zusammenhang zwischen der Qualität der CS Lösung und diesen Parametern besteht. Die Testergebnisse auf den zufällig generierten Graphen im nächsten Abschnitt werden zeigen, dass Knoten mit sehr hohem Grad tatsächlich einen erheblichen Einfluss auf die CS Lösung haben.

Der Transformationsalgorithmus von Eiglsperger löst 8547 Instanzen optimal, wobei für 5325 dieser Instanzen die TE1 Lösung keine überfüllten Bündel enthält, also bereits gültig ist. Bei 3222 Graphen enthält die TE1

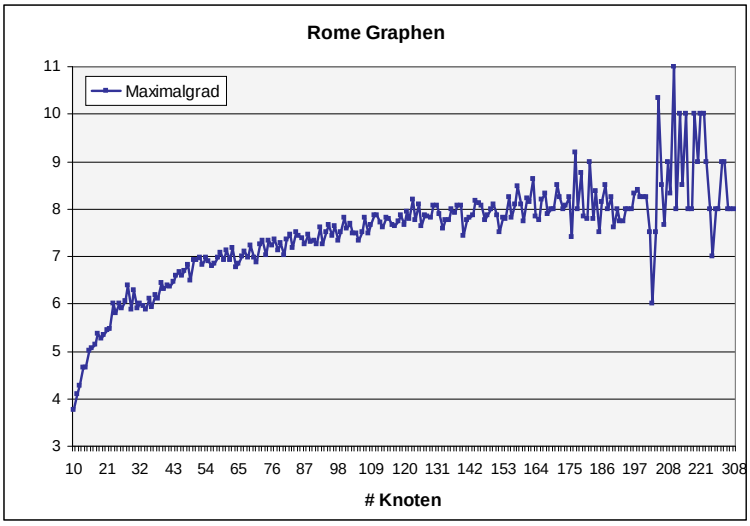

Abbildung 6.7: Der durchschnittliche Maximalgrad der planarisierten Rome Graphen in Bezug auf die Anzahl der Knoten im Graph.

Lösung überfüllte Bündel, die TE Lösung ist aber optimal. Bei den restlichen 2982 Graphen liegt die absolute Abweichung vom ILP-Optimum zwischen eins und sechs. Dabei liegt die durchschnittliche absolute Abweichung bei 0.34 (Abb. 6.3) , die durchschnittliche relative Abweichung bei 1.6% (Abb. 6.4).

Betrachten wir nun die insgesamt 6204 Graphen mit ungültiger TE1 Lösung. Anders als die kritischen Bündel in der CS1 Lösung (vergleiche Abb. 6.5) zeigt die Anzahl der überfüllten Bündel in der TE1 Lösung eine, im Vergleich zu der Anzahl aller Bündel, geringe Steigung (Abb. 6.8).

Den Tabellen 6.3 und 6.4 kann man entnehmen, dass zugleich auch die durchschnittliche Dichte und der Maximalgrad der Graphen einen ähnlich geringen Anstieg aufweisen. Die Anzahl der hochgradigen Knoten steigt ebenfalls an. Die Vermutung liegt Nahe, dass auch die Qualität der TE Lösung weniger von der Größe des Graphen, als von den oben genannten Parametern abhängt.

Der Sukzessive Transformationsalgorithmus löst 10205 Instanzen optimal. Darunter befinden sich auch die 5325 Graphen mit bereits gültiger TE1 Lösung, bei denen natürlich auch die ST1 Lösung im ersten Iterationsschritt gültig ist und der Algorithmus nicht zum Tragen kommt. Bei den

	TE3 optimal		TE3 nicht optimal
	TE1 gültig	TE1 ungültig	
# Graphen	5325	3222	2982
∅ # Knicke (z_{opt})	10.16	22.72	29.44
∅ Transformationskosten	–	3	4.58
∅ Abs. Abw. vom Opt.	–	–	1.32
∅ # Knoten (plan.Gr.)	65	85	97
∅ Dichte	1.26	1.43	1.51
∅ # hochgr. Knoten	2.67	5.55	7
∅ max. Knotengrad	5.7	7.3	7.8
∅ # überf. Bündeln	–	1.46	2.12

Tabelle 6.3: Vergleich einiger Parameter bezüglich der Lösung von TE.

$z_{TE} - z_{opt}$	1	2	3	4	5	6
# Graphen	2262	548	132	32	6	2
∅ # Knicke (z_{opt})	27	34	40	47	63	88
∅ Transformationskosten	4	5.8	7.6	9.2	9.3	14
∅ # Knoten (plan.Gr.)	87	104	116	133	182	262
∅ Dichte	1.49	1.55	1.61	1.66	1.74	1.81
∅ # hochgr. Knoten	6.6	8	9	10.8	13	17.5
∅ max. Knotengrad	7.7	8	8.3	8.5	9	8
∅ # überf. Bündeln	1.86	2.68	3.52	4.25	4.5	6.5

Tabelle 6.4: Vergleich einiger Parameter bezüglich der Absoluten Abweichung der TE Lösung vom ILP-Optimum

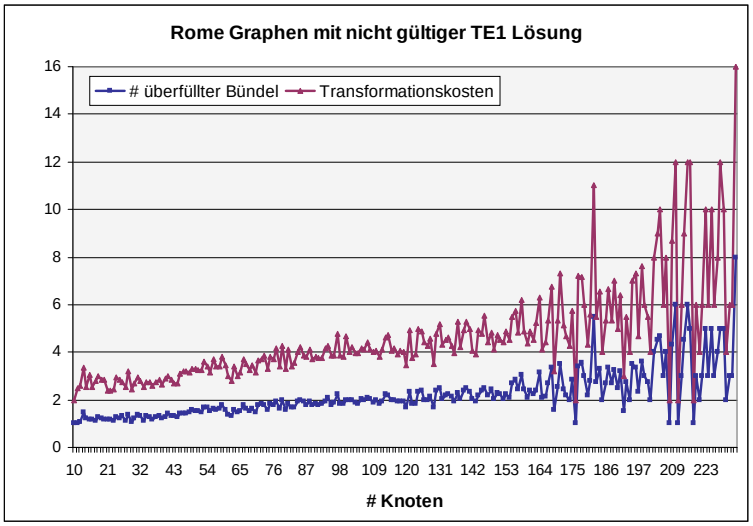

Abbildung 6.8: Die durchschnittliche Anzahl der überfüllten Bündel in der TE1 Lösung und der Transformationskosten bei den Rome Graphen mit ungültiger TE1 Lösung in Bezug auf die Anzahl der Knoten im planarisierten Graph.

restlichen Graphen enthält die ST1 Lösung überfüllte Bündel, bei 4880 davon ist die endgültige ST Lösung aber optimal, bei 1324 liegt ihre absolute Abweichung zwischen eins und vier. Dabei liegt die durchschnittliche absolute Abweichung bei 0.13 und die durchschnittliche relative Abweichung bei 0.62%. (Abb. 6.3, 6.4). Insgesamt löst ST also mehr Instanzen optimal und führt im Durchschnitt auch zu einer niedrigeren absoluten und relativen Abweichung als TE.

In der Tabelle 6.5 fällt auf, dass für 224 der Instanzen mit *nicht optimaler* ST Lösung die TE Lösung optimal ist. Würde man bei diesen Instanzen nach der Transformation im ersten Iterationsschritt *alle* Bündelkanten mit dem Flusswert 0 sperren, wäre die erfolgte Auswahl optimal. Es wird aber nur ein Teil gesperrt, die Auswahl bei den anderen Bündeln ändert sich natürlich in den darauffolgenden Iterationen und es entsteht eine nicht optimale Auswahl. Das muss aber nicht sein: bei den restlichen 2998 Graphen mit optimaler TE Lösung ist auch die ST Lösung optimal. Somit ist bei ca. 93% der von TE optimal gelösten Rome Instanzen die ST Lösung ebenfalls optimal.

	ST optimal			ST nicht optimal	
	ST1 gültig	ST1 ungültig			
		TE n.opt.	TE opt.	TE opt.	TE n.opt.
# Graphen	5325	1882	2998	224	1100
∅ # Knicke (z_{opt})	10.16	30	22	30.6	28.8
∅ ($z_{ST} - z_{opt}$)	–	–	–	1.1	1.17
∅ ($z_{TE} - z_{opt}$)	–	1.27	–	–	1.4
∅ Tr.kost. in 1. Iter.	–	4.29	3	3.5	5.1
∅ Tr.kost. insgesamt	–	11.3	7.9	16.4	14
∅ # Iterationen	–	4	3.5	5.52	4.46
∅ # Knoten	45	95	76	94	88
∅ Dichte	1.26	1.51	1.42	1.52	1.51
∅ # hochgr. Knoten	2.67	7.13	5.4	7.19	6.84
∅ max. Knotengrad	5.7	7.7	7.3	8	7.97
∅ # Bündel	113	285	214	282	263
∅ # überf. Bündeln	–	2.1	1.44	1.7	2.16

Tabelle 6.5: Vergleich einiger Parameter bezüglich der Lösungen von ST und TE.

$z_{ST} - z_{opt}$	1	2	3	4
# Graphen	1136	164	21	3
∅ # Knicke (z_{opt})	28	34	38	46
∅ Tr.kosten (ST)	13.5	19.3	24.1	38
∅ Tr.kosten (TE)	4.7	5.3	6.9	8
∅ # Iterationen	4.4	5.7	6.3	8.3
∅ # Knoten (plan.Gr.)	92	104	111	123
∅ Dichte	1.51	1.55	1.55	1.64
∅ # hochgr. Knoten	6.7	8	9.1	10.3
∅ max. Knotengrad	7.9	8.1	8.3	9
∅ # überf. Bündeln	2	2.3	3.1	3.6

Tabelle 6.6: Vergleich einiger Parameter bezüglich der Absoluten Abweichung der ST Lösung vom ILP Optimum.

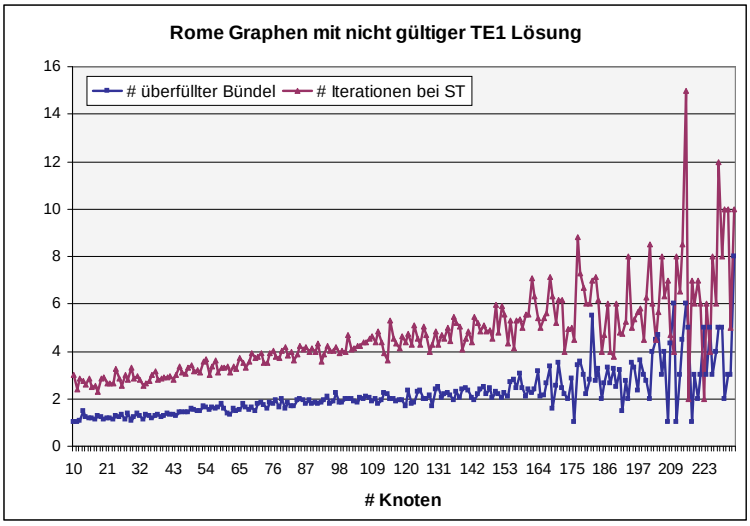

Abbildung 6.9: Die durchschnittliche Anzahl der überfüllten Bündel in der TE1 Lösung und die durchschnittliche Anzahl der von ST durchgeführten Iterationen bei den Rome Graphen mit ungültiger TE1 Lösung in Bezug auf die Anzahl der Knoten im planarisierten Graph.

Die durchschnittliche Anzahl der Iterationen hängt unter anderem auch von der Anzahl der Bündel im Netzwerk ab, allerdings nicht so stark (Abb. 6.9). Sie zeigt ein ähnliches Verhalten wie die Anzahl der überfüllten Bündel. Das deutet wieder auf einen Zusammenhang mit der Dichte und dem Maximalgrad der Graphen sowie der Anzahl der hochgradigen Knoten hin.

6.2.2 Laufzeitanalyse

Die Laufzeit liegt bei allen drei Verfahren unter 0.5 Sekunden (Abb. 6.10). Dabei hat ST im Durchschnitt die längste Laufzeit, was auf die mehrfachen Iterationen zurückzuführen ist. Damit übersteigt sie im Durchschnitt die Laufzeit des ILPs, da das ILP für diese einfachen Instanzen relativ schnell gelöst werden kann. CS hat eine etwas höhere Laufzeit als TE, da der Schritt CS1 mehr Rechenzeit benötigt als der Schritt TE1, das einem gewöhnlichen MCF Problem entspricht (Abb. 6.11).

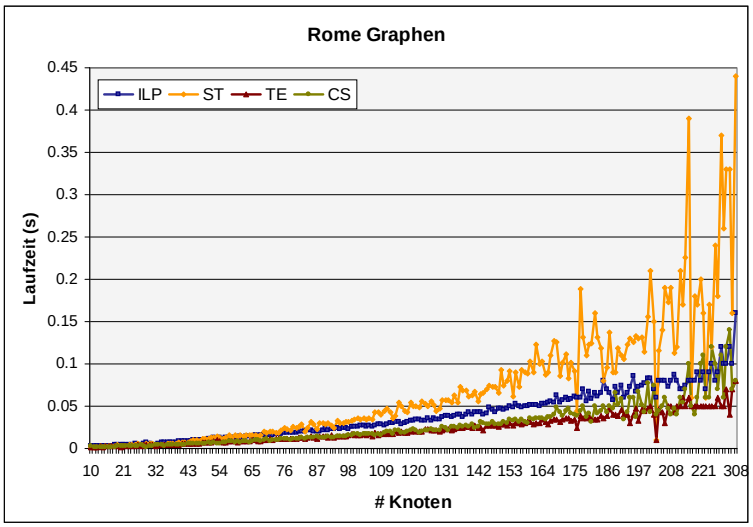

Abbildung 6.10: Die Laufzeiten von ST, TE, CS und dem KMCF-ILP bei den Rome Grapen in Bezug auf die Anzahl der Knoten im planarisierten Graph.

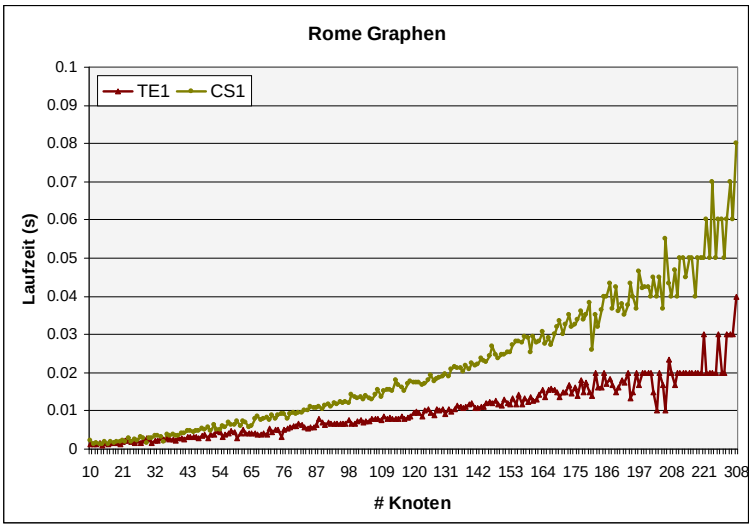

Abbildung 6.11: Die Laufzeiten von TE1 und CS1 bei den Rome Graphen in Bezug auf die Anzahl der Knoten im planarisierten Graph.

6.3 Die Random-Planar (RP) Graphen

Unsere RP-Graphen Sammlung setzt sich aus folgenden Graphen zusammen:

• Je 3000 *einfach, 2-* und *3-zusammenhängende zufällige* Graphen. Die Graphen jeder dieser drei Klassen können bezüglich ihrer Kantenanzahl in 3 Gruppen mit je 1000 Graphen unterteilt werden: $1.5n$, $2n$ und $2.5n$ Kanten. Jede dieser Gruppen besteht wiederum aus 10 Untergruppen mit je 100 Graphen mit n Knoten, wobei

$$n \in \{50, 100, 150, 200, 250, 300, 400, 500, 600, 700\}$$

ist. Bei den 3-zusammenhängenden Graphen der Klasse $m = 1.5n$ liegt die durchschnittliche Dichte bei 1.52, da der benutzte Generator (*Planar Triconnected Graph* Generator von [AGD]) manchmal einige Kanten mehr als $1.5n$ erzeugt.

• 3000 zufällige, maximal planare Graphen. Davon sind 1000 mit dem *Random Planar* Graph Generator von [LEDA], weitere 1000 mit dem *Planar Biconnected* und 1000 mit dem *Planar Triconnected* Graph Generator von AGD generiert. Die Graphen dieser Gruppen sind wie oben nach ihrer Knotenanzahl in 10 Untergruppen unterteilt.

• 1000 serien-parallele Graphen. Je 100 Graphen mit n Knoten, wobei n wie oben definiert ist. m liegt zwischen 1.80 und 1.86 wobei die durchschnittliche Dichte bei 1.84 liegt. Man beachte, dass ein serien-paralleler Graph mit n Knoten höchstens $2n - 3$ Kanten besitzt.

• 1000 Bäume. Je 100 Graphen mit n Knoten und m Kanten, wobei n ebenfalls wie oben definiert und $m = n - 1$ ist.

Zusätzlich haben wir die 29 vollständigen Graphen K_6-K_{34} getestet. Diese werden, auf die gleiche Weise wie die nicht planaren Rome Graphen, mit dem SubgraphPlanarizer Modul von AGD planarisiert.

6.3.1 Qualitätsanalyse

Schauen wir uns zunächst an, wie sich die einzelnen Verfahren bei den unterschiedlichen Testinstanzen verhalten.

Die Tab. 6.7 zeigt eine Zusammenfassung der Ergebnisse aller drei Verfahren auf den maximal planaren Graphen. Es fällt auf, dass die mit dem *Random Planar* oder dem *Planar Biconnected* Graph Generator erstellten maximal planaren Graphen für alle drei Verfahren schwieriger zu lösen sind, als die mit dem *Planar Triconnected* Graph Generator erstellten maximal planaren Graphen: die Verfahren weisen alle deutlich bessere Ergebnisse bei den letzteren auf.

	Maximal Planare Graphen		
Graph Generator	Rand. Pl.	Pl. Bicon.	Pl. Tricon.
# Graphen mit $z_{CS} = z_{opt}$	381	472	995
# Graphen mit $z_{ST} = z_{opt}$	–	2	238
# Graphen mit $z_{TE} = z_{opt}$	–	–	1
∅ Maximalgrad	58	55	13
∅ relative Abweichung CS (%)	0.21	0.16	~ 0
∅ relative Abweichung ST (%)	3.5	3.5	0.33
∅ relative Abweichung TE (%)	8.5	8.5	3.5

Tabelle 6.7: Vergleich der Ergebnisse von CS, ST und TE bei den unterschiedlich generierten maximal planaren Graphen

Dieser große Unterschied hängt mit den Maximalgraden der jeweiligen Graphen zusammen. Der größte auftretende Knotengrad bei den mit dem *Planar Triconnected* Generator erstellten Graphen liegt bei 22, während sie für die mit dem *Planar Biconnected* und *Random Planar* Generatoren erstellten Graphen jeweils bei 147 und 130 liegt. Abb. 6.15 zeigt die durchschnittlichen Maximalgrade der Graphen aller drei Klassen. Bei Graphen mit großem Maximalgrad können sehr lange Folgen von benachbarten kritischen Bündeln (bzw. lange maximale Teillisten mit einem überfüllten Bündel) um einen hochgradigen Knoten entstehen, deren Korrektur hohe Kosten verursacht. Das erhöht wiederum die Wahrscheinlichkeit einer nicht optimalen Lösung. Aus Abb. 6.15 können wir aber schließen, dass CS bei Instanzen mit einem durchschnittlichen Maximalgrad von bis zu ca. 13 noch gute Ergebnisse liefert.

Die Abb. 6.16 zeigt, dass die mit dem *Planar Triconnected* Generator erstellten Graphen die CS1 Lösung im Durchschnitt weniger kritische Bündel

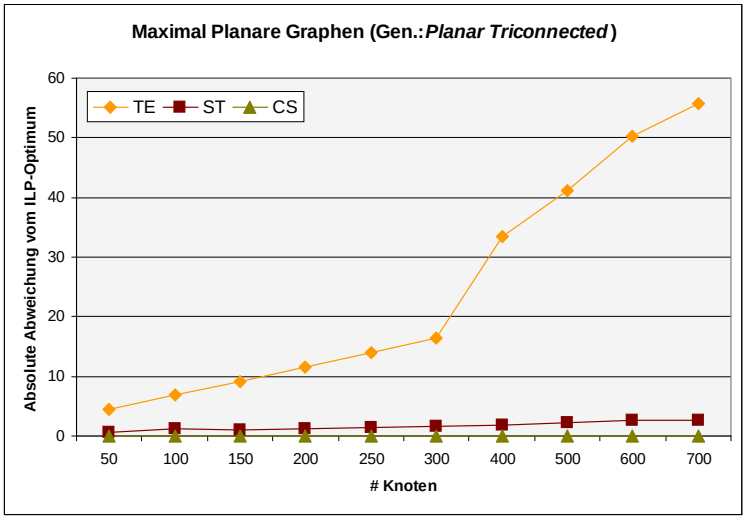

Abbildung 6.12: Die Absolute Abweichung der TE, ST und CS Lösungen vom ILP-Optimum bei den mit dem *Planar Triconnected* Generator erstellten maximal planaren Graphen in Bezug auf die Anzahl der Knoten im Graph.

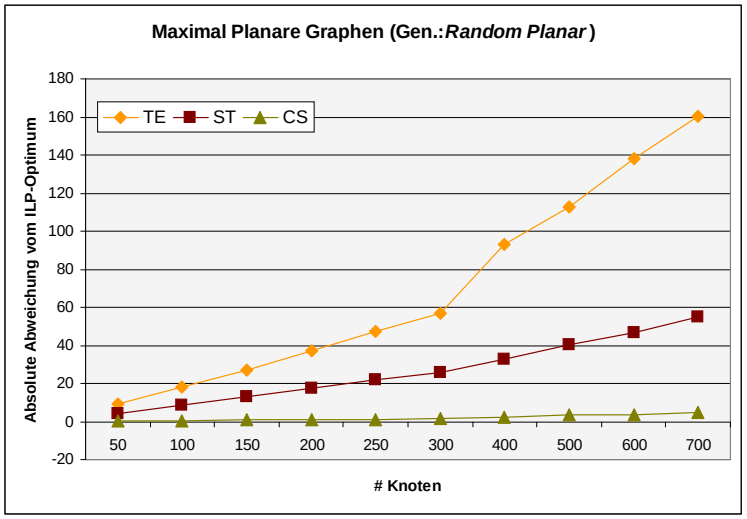

Abbildung 6.13: Die Absolute Abweichung der TE, ST und CS Lösungen vom ILP-Optimum bei den mit dem *Random Planar* Generator erstellten maximal planaren Graphen in Bezug auf die Anzahl der Knoten im Graph.

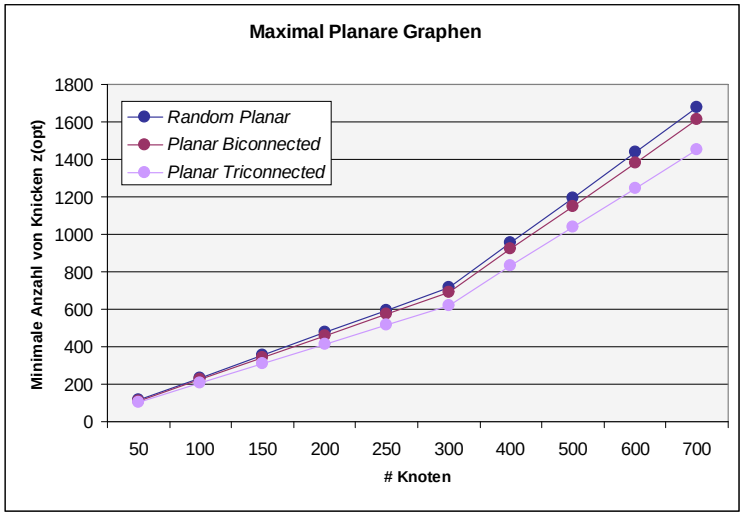

Abbildung 6.14: Die minimale Anzahl von Knicken (z_{opt}) bei den mit unterschiedlichen Generatoren erstellten maximal planaren Graphen in Bezug auf die Anzahl der Knoten im Graph.

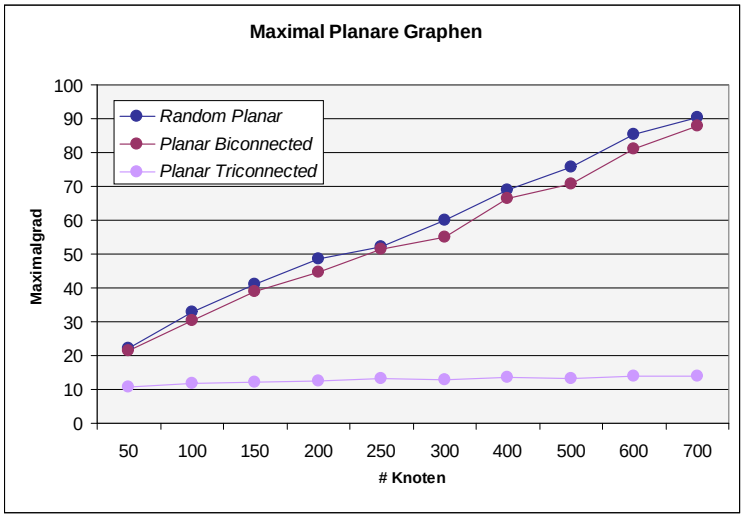

Abbildung 6.15: Der durchschnittliche Maximalgrad der unterschiedlich generierten maximal planaren Graphen in Bezug auf die Anzahl der Knoten im Graph.

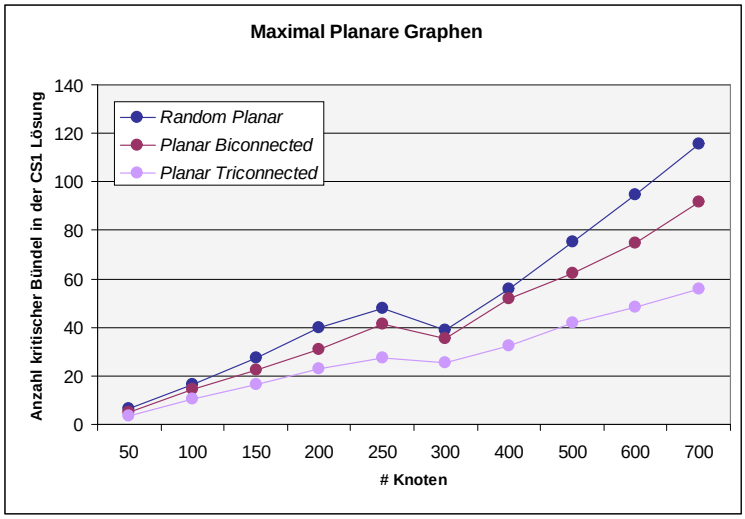

Abbildung 6.16: Die durchschnittliche Anzahl der kritischen Bündel in der CS1 Lösung bei den unterschidlich generierten maximal planaren Graphen in Bezug auf die Anzahl der Knoten im Graph.

enthält als bei den anderen maximal planaren Graphen. Im Vergleich zu den Maximalgraden dieser Graphen weist sie aber dennoch eine deutliche Steigung mit wachsender Knotenanzahl auf (Abb. 6.15), was auf den Einfluss der Größe der Graphen zurückzuführen ist. Interessanterweise verhalten sich die Kosten, die bei der Korrektur dieser kritischen Bündel in Schritt CS2 entstehen, anders: sie zeigen eine ähnliche Steigung wie bei dem Maximalgrad der Graphen (Abb. 6.17). Die kritischen Bündel, die bei Graphen mit kleinerem Maximalgrad entstehen, können folglich mit relativ geringeren Kosten korrigiert werden. Das bestätigt unsere obige Vermutung, dass lange Ketten von benachbarten kritischen Bündeln mehr Korrekturkosten verursachen. Somit haben kleinere Knotengrade nicht nur eine positive Auswirkung auf die Anzahl der kritischen Bündel, sondern zusätzlich auch auf die Korrekturkosten und somit eine *doppelte* positive Auswirkung auf die Qualität der CS Lösung.

Die durchschnittliche Anzahl der überfüllten Bündel zeigt ein ähnliches Verhalten wie die der kritischen Bündel in der CS1 Lösung, nur in einer anderen Größenordnung. Anders als bei CS verhalten sich die Transformationskosten in Schritt TE2 bei den mit *Planar Triconnected* Generator erstellten

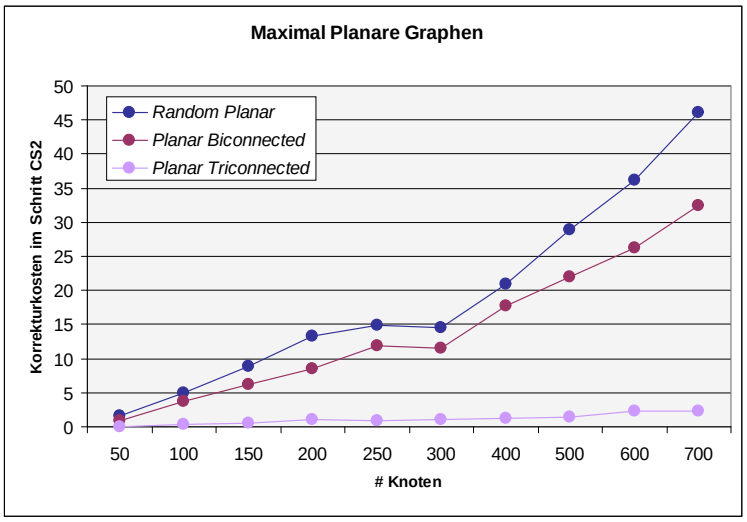

Abbildung 6.17: Die Korrekturkosten in Schritt CS2 bei den unterschidlich generierten maximal planaren Graphen in Bezug auf die Anzahl der Knoten im Graph.

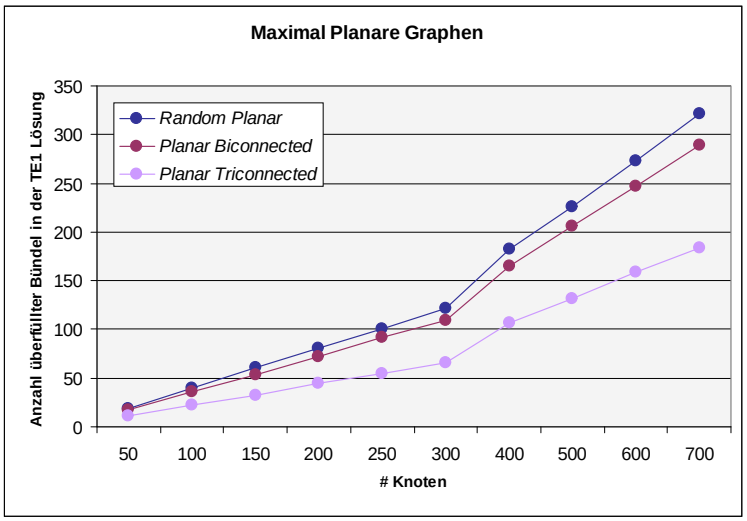

Abbildung 6.18: Die durchschnittliche Anzahl der überfüllten Bündel in der TE1 Lösung bei den unterschidlich generierten maximal planaren Graphen in Bezug auf die Anzahl der Knoten im Graph.

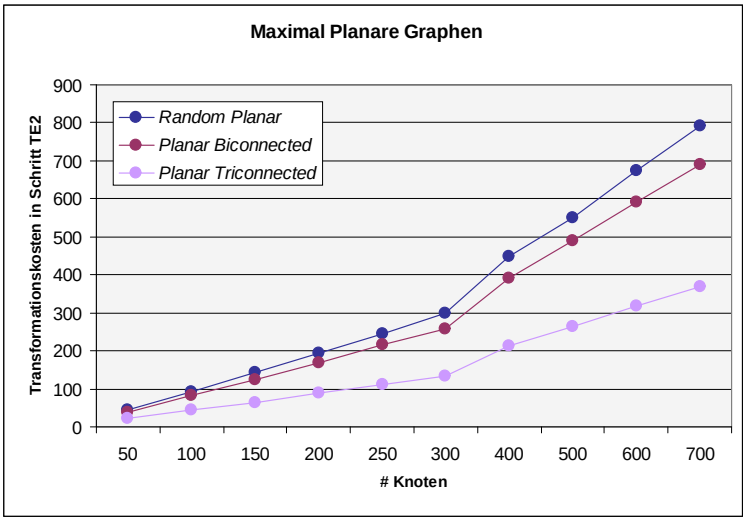

Abbildung 6.19: Die Transformationskosten in Schritt TE2 bei den unterschidlich generierten maximal planaren Graphen in Bezug auf die Anzahl der Knoten im Graph.

Graphen genau so, wie bei den anderen. Kleinere Knotengrade haben zwar einen Einfluss auf die Anzahl der überfüllten Bündel, aber keinen zusätzlichen Einfluss auf die Transformationskosten.

Die durchschnittliche Anzahl der Iterationen bei ST ist auf den mit *Planar Triconnected* Generator erstellten maximal planaren Graphen deutlich höher als auf den anderen (Abb. 6.20). Das hängt womöglich damit zusammen, dass bei Graphen mit kleinem Maximalgrad eher kurze maximale Teillisten entstehen und dadurch die Anzahl der in einem Iterationsschritt gesperrten Bündelkanten klein ist. Somit dauert es länger, bis ausreichend Bündel gesperrt sind, sodass der Algorithmus mit einer Lösung ohne überfüllte Bündel terminieren kann. Die Tatsache, dass ST bei solchen Graphen deutlich bessere Ergebnisse liefert zeigt, dass öftere Transformationen auf kurzen maximalen Teillisten zu einer besseren Auswahl von Bündelkanten führt als wenige Transformationen auf langen maximalen Teillisten.

Der Maximalgrad ist natürlich nicht der einzige Parameter, der sich auf die Qualität der Lösungen auswirkt. Ein weiterer Parameter ist der Anteil an hochgradigen Knoten im Graph. Er ist bis zu einem bestimmten Grad

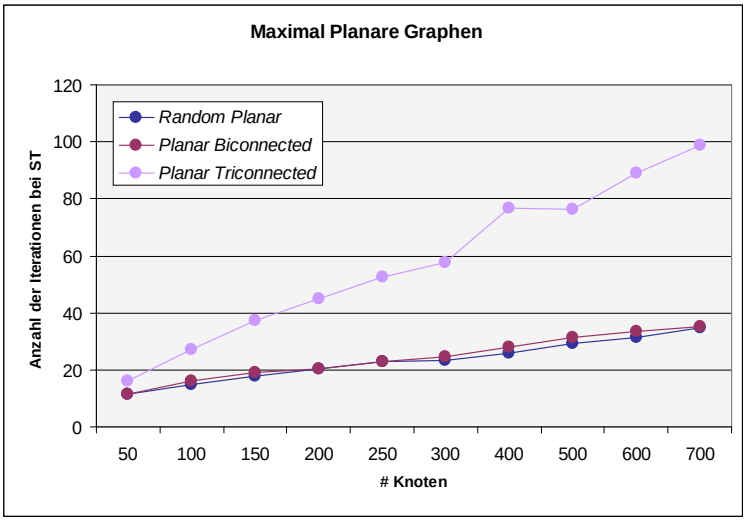

Abbildung 6.20: Die durchschnittliche Anzahl der Iterationen bei ST auf den unterschidlich generierten maximal planaren Graphen in Bezug auf die Anzahl der Knoten im Graph.

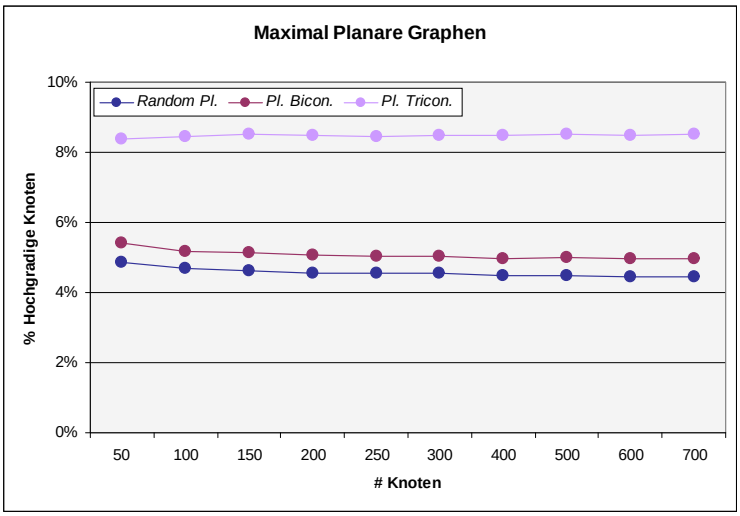

Abbildung 6.21: Der durchschnittliche Prozentanteil an hochgradigen Knoten bei den unterschiedlich generierten maximal planaren Graphen in Bezug auf die Anzahl der Knoten im Graph.

mit dem Maximalgrad verbunden, da bei gleichbleibender Dichte der Anteil an hochgradigen Knoten mit sinkendem Maximalgrad zunimmt (Abb. 6.21). Ein Anstieg im Anteil der hochgradigen Knoten kann daher bedeuten, dass die Instanz leichter geworden ist. Bleibt der Maximalgrad gleich und erhöht sich die Anzahl der hochgradigen Knoten aufgrund eines Anstiegs in der Dichte, führt das im allgemeinen zu schwierigeren Instanzen.

Die Dichte des Graphen hat zweifellos auch einen Einfluss auf das Ergebnis. Im allgemeinen kann man sagen, dass die Instanzen mit steigender Dichte schwieriger werden (siehe auch Tabellen 6.8 , 6.9 und 6.10). Man sollte beachten, dass bei gleichbleibender Knotenanzahl mit der Dichte im allgemeinen auch der Maximalgrad der Graphen steigt, was die Instanzen zusätzlich erschwert (Abb. 6.22).

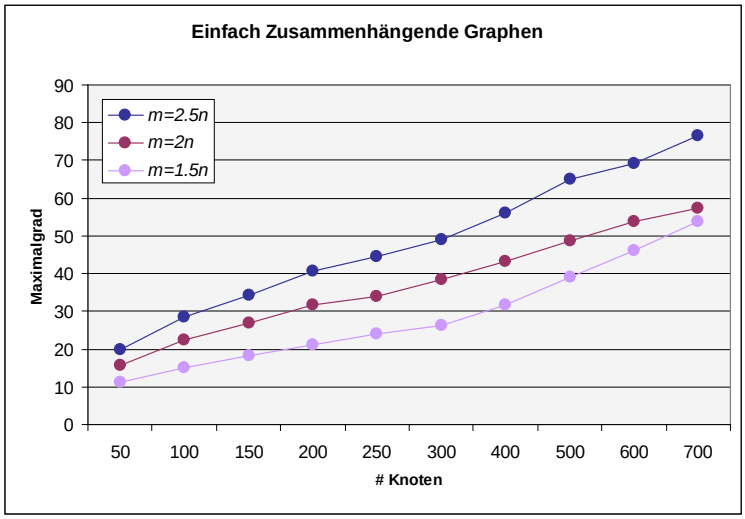

Abbildung 6.22: Der durchschnittliche Maximalgrad der einfach zusammenhängenden Graphen unterschiedlicher Dichte in Bezug auf die Anzahl der Knoten im Graph.

Die Abb. 6.23 zeigt, wie sich die Ergebnisse der drei Verfahren innerhalb der Klasse der einfach zusammenhängenden Graphen mit Dichte 1.5 verhalten. Es fällt auf, dass die durchschnittlichen Werte z_{CS1} und z_{CS} näher am Optimum liegen als jeweils z_{TE1}, z_{TE} und z_{ST}. Dieses Verhalten der durchschnittlichen absoluten Abweichungen (größere Steigung bei TE und ST im

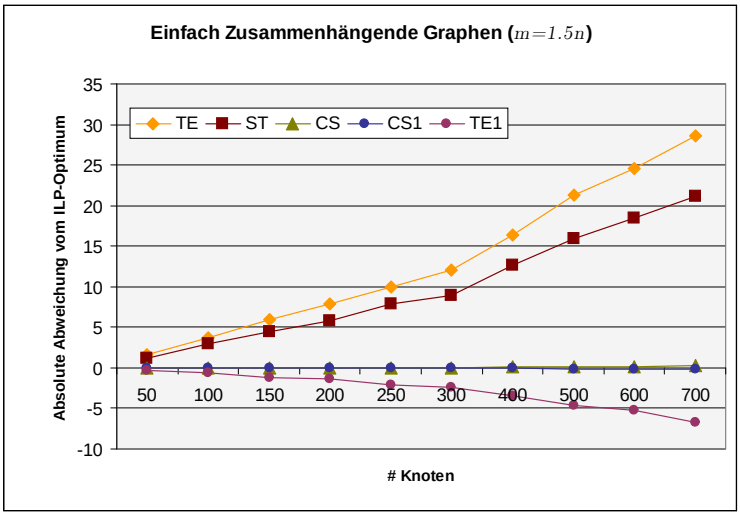

Abbildung 6.23: Die durchschnittliche absolute Abweichung der Lösungen z_{TE}, z_{ST}, z_{CS}, z_{CS1} und z_{TE1} vom ILP-Optimum (z_{opt}) bei einfach zusammenhängenden Graphen mit $m = 1.5n$ in Bezug auf die Anzahl der Knoten im Graph.

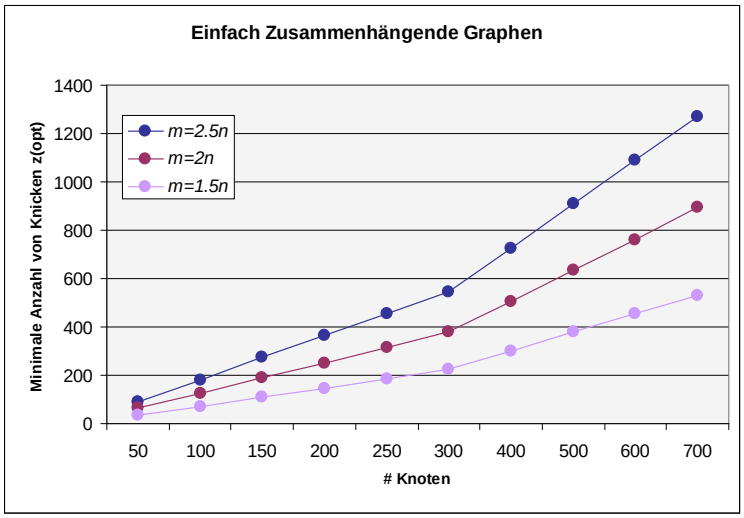

Abbildung 6.24: Die durchschnittliche minimale Anzahl von Knicken (z_{opt}) bei den einfach zusammenhängenden Graphen mit unterschiedlicher Dichte in Bezug auf die Anzahl der Knoten im Graph.

	Einfach zusammenhängende Graphen		
	$m = 1.5n$	$m = 2n$	$m = 2.5n$
# Graphen mit $z_{CS} = z_{opt}$	923	638	486
# Graphen mit $z_{ST} = z_{opt}$	24	3	1
# Graphen mit $z_{TE} = z_{opt}$	16	1	–
∅ Relative Abweichung CS (%)	0.02	0.11	0.17
∅ Relative Abweichung ST (%)	4	4.45	4.11
∅ Relative Abweichung TE (%)	5.3	7.85	9.09

Tabelle 6.8: Vergleich der Ergebnisse von CS, ST und TE bei den einfach zusammenhängenden planaren Graphen mit unterschiedlicher Dichte

	2-zusammenhängende Graphen		
	$m = 1.5n$	$m = 2n$	$m = 2.5n$
# Graphen mit $z_{CS} = z_{opt}$	974	777	568
# Graphen mit $z_{ST} = z_{opt}$	133	11	3
# Graphen mit $z_{TE} = z_{opt}$	56	3	–
∅ Relative Abweichung CS (%)	0.01	0.07	0.12
∅ Relative Abweichung ST (%)	2.47	3.368	3.375
∅ Relative Abweichung TE (%)	4.18	6.75	8.43

Tabelle 6.9: Vergleich der Ergebnisse von CS, ST und TE bei den 2-zusammenhängenden planaren Graphen mit unterschiedlicher Dichte

	3-zusammenhängende Graphen		
	$m = 1.5n$	$m = 2n$	$m = 2.5n$
# Graphen mit $z_{CS} = z_{opt}$	1000	1000	999
# Graphen mit $z_{ST} = z_{opt}$	1000	567	302
# Graphen mit $z_{TE} = z_{opt}$	983	102	4
∅ Relative Abweichung CS (%)	0	0	~ 0
∅ Relative Abweichung ST (%)	0	0.32	0.39
∅ Relative Abweichung TE (%)	0.09	1.8	3.6

Tabelle 6.10: Vergleich der Ergebnisse von CS, ST und TE bei den 3-zusammenhängenden planaren Graphen mit unterschiedlicher Dichte

Verhältnis zu CS) ist bei allen bisher betrachteten Graphen (mit Ausnahme der Bäume) mehr oder weniger gleich. Je nach Klasse und Dichte ändern sich die Absolutwerte. Steigt die Dichte z.B. bei den zusammenhängenden Graphen mit $n = 700$ auf 2, dann erhöht sich die durchschnittliche Abweichung bei TE von 29 auf 72, bei ST von 21 auf 39 und bei CS von 0.3 auf 1.54. Erhöht man die Dichte weiter auf 2.5, liegen diese Werte jeweils bei 117, 52 und 2.63.

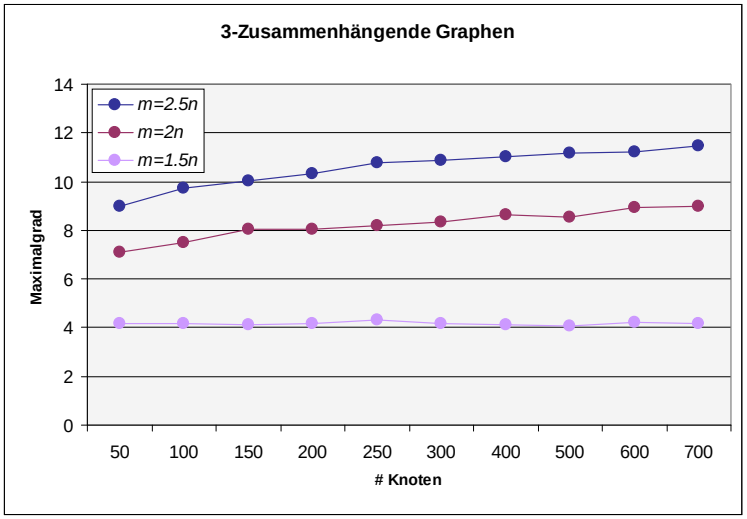

Abbildung 6.25: Der durchschnittliche Maximalgrad der 3-zusammenhängenden Graphen unterschiedlicher Dichte in Bezug auf die Anzahl der Knoten im Graph.

Bei den 3-zusammenhängenden Graphen liefern alle drei Verfahren deutlich bessere Ergebnisse als bei den einfach und 2-zusammenhängenden, was ebenfalls mit den geringen Maximalgraden dieser Graphen zusammenhängt. Bei den 3-zusammenhängenden Graphen mit Dichte 1.5 liegt der durchschnittliche Maximalgrad bei ca. 4 (Abb. 6.25), wodurch auch ST und TE einen Großteil dieser Instanzen optimal lösen können. Bei den Instanzen mit Dichte 2 und 2.5 steigt mit der Dichte auch der durchschnittliche Maximalgrad, bleibt allerdings unter 12. Die Ergebnisse von ST und TE werden von diesem Anstieg deutlich beeinflusst, während CS auch diese Instanzen (fast) alle optimal löst (1999 von 2000). Wir können daraus schließen, dass bei Gra-

phen mit einem Maximalgrad von bis zu ca. 12-13, die Dichte und Größe der Instanzen fast keinen Einfluss auf die CS Lösung hat.

	Serien-Parallele Graphen	Bäume
# Graphen mit $z_{CS} = z_{opt}$	491	1000
# Graphen mit $z_{ST} = z_{opt}$	18	1000
# Graphen mit $z_{TE} = z_{opt}$	10	1000
∅ Maximalgrad	99.75	8.85
∅ Relative Abweichung CS (%)	0.24	–
∅ Relative Abweichung ST (%)	2.02	–
∅ Relative Abweichung TE (%)	4.14	–

Tabelle 6.11: Vergleich der Ergebnisse von CS, ST und TE bei den serien-parallelen Graphen und bei den Bäumen

Die serien-parallelen Graphen haben eine durchschnittliche Dichte von 1.84 und können diesbezüglich mit den Graphen mit Dichte 2 verglichen werden. Für CS sind die serien-parallelen Graphen schwieriger zu lösen als die einfach und 2-zusammenhängenden, obwohl sie eine geringere Dichte haben. Die relative Abweichung liegt mit 0.24 sogar höher als bei den maximal planaren Graphen. Ein Blick in die Tab. 6.11 legt die Vermutung nahe, dass das mit dem höheren durchschnittlichen Maximalgrad der serien-parallelen Graphen zusammenhängt. Wenn der Maximalgrad eines Graphen steigt, obwohl seine Dichte abnimmt, dann wird die Qualität der CS Lösung mehr von dem Maximalgrad des Graphen als von seiner Dichte beeinflusst.

TE und ST liefern für die serien-parallelen Graphen bessere Ergebnisse, als für die einfach und 2-zusammenhängenden. Das bestätigt unsere Vermutung, dass das Ergebnis bei diesen beiden Verfahren nicht so stark vom Maximalgrad abhängt, wie es bei CS der Fall ist. Dennoch kann man sich folgende Frage stellen: Warum führt die Abnahme des Maximalgrades bei dem Übergang von den einfachen zu den 3-zusammenhängenden Graphen zu einer Verbesserung in der TE und ST Lösung, bei dem Übergang von den serien-parallelen zu den einfach zusammenhängenden Graphen aber zu einer Verschlechterung?

Eine mögliche Erklärung dafür wäre die folgende: Bei dem Übergang von serien-parallel zu einfach zusammenhängend verringert sich zwar der Ma-

	$n = 200$		
	$m \simeq 1.83n$	$m = 2n$	
	Serien-Parallel	Einf. Zus.	3-Zus.
∅ Rel. Abw. CS (%)	0.2	0.1	0
∅ Rel. Abw. ST (%)	2.21	4.51	0.23
∅ Rel. Abw. TE (%)	4.58	7.75	1.83
∅ Maximalgrad	70	32	8
gößter Knotengrad	110	53	11
∅ % hochgr. Knoten	2.07	3.87	4.43
% Knoten mit Grad ≥ 6	8%	12%	2%
% Knoten mit Grad ≤ 2	75%	38%	–

Tabelle 6.12: Vergleich der Ergebnisse von CS, ST und TE sowie einiger Parameter bei den einfach und 3-zusammenhängenden planaren Graphen mit $m = 2n$ und $n = 200$ und den serien-parallelen Graphen mit 200 Knoten

ximalgrad, aber nicht so stark, dass dadurch eine möglichst gleichmäßige Verteilung der Kanten auf die Knoten erzwungen wird. Für eine gleichmäßige Verteilung müsste eine Abnahme im Maximalgrad zusammen mit einer Abnahme im Anteil der Knoten mit sehr hohem Grad und im Anteil der Knoten mit Grad ≤ 2 erfolgen.

Die vorletzte Zeile in Tab. 6.12 zeigt, dass bei dem Übergang von serien-parallelen zu einfach zusammenhängenden Graphen der Anteil an Knoten mit *relativ hohem Grad* (wie z.B. ≥ 6) von 8% auf 12% steigt. Bei dem Übergang zu 3-zusammenhängenden Graphen hingegen sinkt dieser Anteil auf 2%. Der Anteil an Knoten mit Grad ≤ 2 sinkt bei beiden Übergängen, beim ersteren jedoch nicht genug, um das Problem für TE und ST zu vereinfachen.

Die Bäume sind die am einfachsten zu lösenden Instanzen und werden alle von allen drei Verfahren optimal gelöst. Das liegt daran, dass es bei Bäumen nur eine Fläche gibt und dadurch die zusätzlichen Kosten, die bei den Korrekturen und Transformationen aufgrund von Flussverschiebungen entlang $f - f$ Kanten entstehen, entfallen. Die korrigierte Lösung ist hier bereits optimal, die Schritte CS4 und TE4 brauchen nicht durchgeführt zu werden.

Bei allen Baum Instanzen ist die CS1 Lösung bereits ganzzahlig. Die TE1 Lösung ist bei 185 Instanzen gültig, bei den restlichen Bäumen enthält sie

überfüllte Bündel. Die durchschnittliche Anzahl der Iterationen bei ST liegt zwischen 1 und 41, wobei es natürlich keinen Sinn macht mehrere Iterationen durchzuführen, da die Optimalität der TE Lösung garantiert ist.

Die vollständigen planarisierten Graphen haben eine durchschnittliche Dichte von 2.15. Vergleicht man sie mit den anderen Graphen mit $m \simeq 2n$, so liegen sie bezüglich ihrer durchschnittlichen relativen Abweichung zwischen den einfach zusammenhängenden und den serien-parallelen Graphen. 16 von 29 Graphen werden von CS optimal gelöst. TE löst nur eine Instanz optimal (K_7), bei ST steigt die Zahl auf 7.

	Vollständige Graphen ($K_6 - K_{34}$)
# Graphen mit $z_{CS} = z_{opt}$	16
# Graphen mit $z_{TE} = z_{opt}$	1
# Graphen mit $z_{ST} = z_{opt}$	7
∅ Maximalgrad	19
∅ Relative Abweichung CS (%)	0.19
∅ Relative Abweichung TE (%)	9.5
∅ Relative Abweichung ST (%)	0.65

Tabelle 6.13: Vergleich der Ergebnisse von CS, TE und ST bei den vollständigen Graphen K6-K34

Trotz des beachtlichen Anstiegs in der Größe der Graphen bleibt die absolute Abweichung bei CS und ST relativ gering (Abb. 6.26). Das liegt an den geringen Knotengraden: alle Knoten in einem vollständigen Graphen K_n besitzen den Knotengrad $n - 1$. Dieser Maximalgrad bleibt nach der Planarisierung unverändert, da nur Knoten mit dem Grad 4 hinzukommen. Das führt zu sehr guten Ergebnissen bei CS und ST (Abb. 6.27). Die Anzahl der von ST durchgeführten Iterationen liegt bei diesen Instanzen zwischen 1 und 26. Im Vergleich zu den mit dem *Planar Triconnected* Generator erstellten maximal planaren Graphen, die zudem viel kleiner sind als die planarisierten vollständigen Graphen, ist das ziemlich gering (vergleiche Abb. 6.20). Das ist auf die spezielle Struktur dieser Instanzen zurückzuführen: z.B. haben 17744 der 17778 Knoten im planarisierten K_{34} den Grad 4. Auf die Lösung von TE wirkt sich dieser Umstand nicht so positiv aus. Der Anstieg in der Größe der Graphen hat hier einen größeren Einfluss: die Anzahl der Bündel

steigt an (von 42 auf 72098), ebenso die Anzahl der überfüllten Bündel (von 0 auf 236).

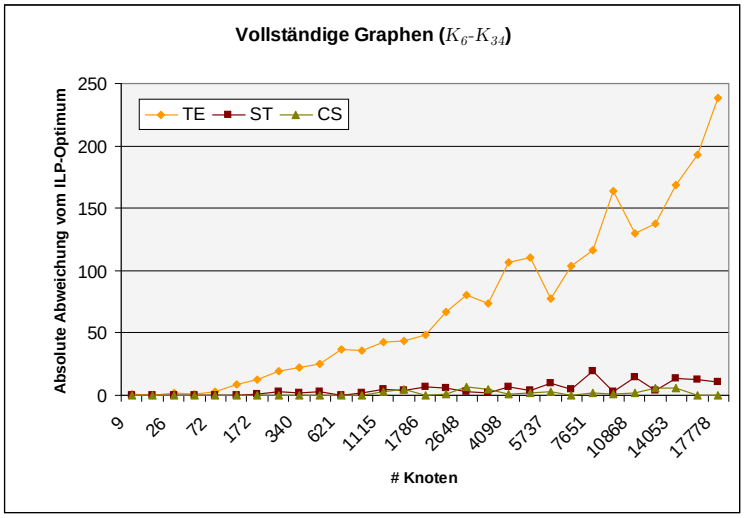

Abbildung 6.26: Die Absolute Abweichung der CS, TE und SP Lösungen vom ILP Optimum bei den vollständigen Graphen $K_6 - K_{34}$ in Bezug auf die Anzahl der Knoten im planarisierten Graph.

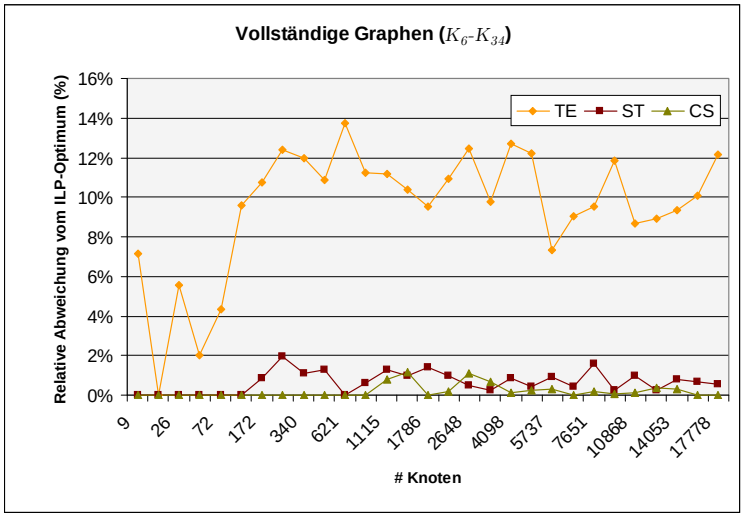

Abbildung 6.27: Die Relative Abweichung der CS, TE und SP Lösungen vom ILP Optimum bei den vollständigen Graphen $K_6 - K_{34}$ in Bezug auf die Anzahl der Knoten im planarisierten Graph.

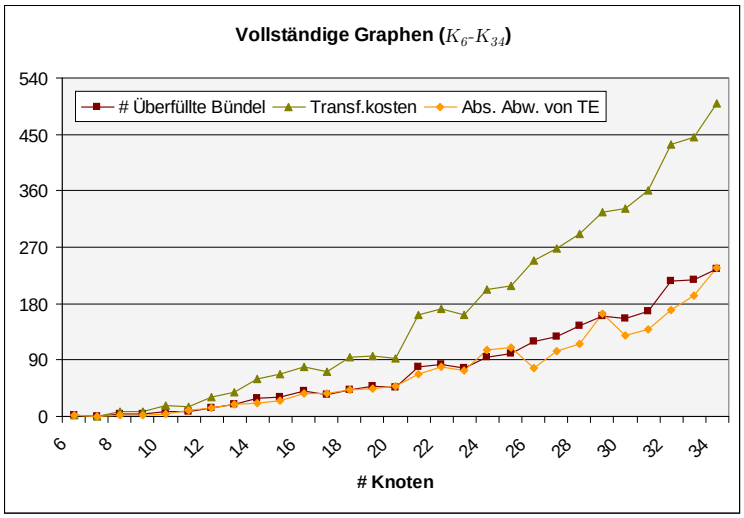

Abbildung 6.28: Ein Vergleich der Anzahl an überfüllten Bündel, der Transformationskosten die bei deren Korrektur entsthen und der absoluten Abweichung der TE Lösung vom ILP-Optimum in Bezug auf die Knoten im Graph.

6.3.2 Laufzeitverhalten

Der TE Algorithmus hat im allgemeinen die geringste Laufzeit (Abb. 6.29, 6.30). Das liegt daran, dass das LP in Schritt TE1 einem gewöhnlichen MCF Problem entspricht, während das LP in Schritt CS1 zusätzlich die Bündelkapazitätsbeschränkung enthält und daher eine längere Laufzeit benötigt (Abb. 6.31). Da CS1 fast $65 - 80\%$ der Gesamtlaufzeit von CS verbraucht (Abb. 6.32), bewirkt es einen deutlichen Unterschied zwischen den Gesamtlaufzeiten beider Verfahren. Bei TE haben TE1 und TE4 ungefähr die gleiche Laufzeit während TE2,3 vergleichsmäßig eine etwas geringere Laufzeit benötigt (Abb. 6.33). Aufgrund der mehrfachen Iterationen hat von den drei Algorithmen ST die längste Laufzeit. Sie übersteigt dadurch bei manchen Instanzen die Laufzeit des ILPs.

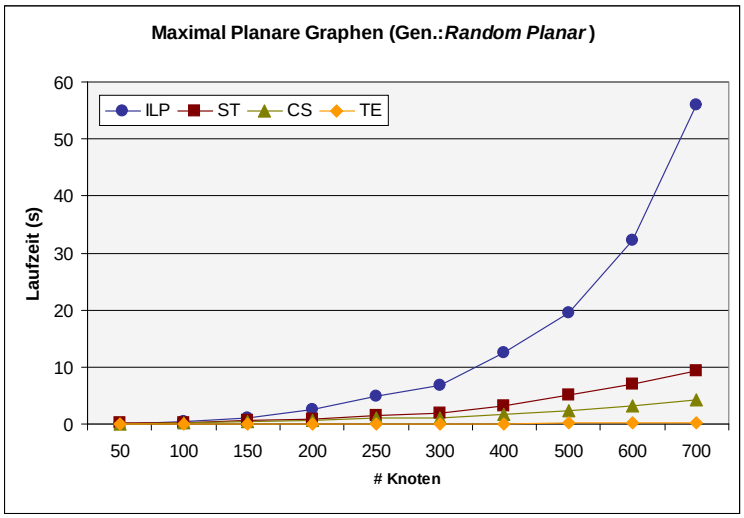

Abbildung 6.29: Die Laufzeiten der vier Verfahren bei den mit dem *Random Planar* Graph Generator erzeugten maximal planaren Graphen in Bezug auf die Anzahl der Knoten im Graph.

Bei den Instanzen, für die CS1 bereits eine ganzzahlige Lösung liefert, sodass die weiteren Schritte des Algorithmus entfallen, hat CS eine geringere Laufzeit (Abb. 6.34). Bei den relativ einfachen Instanzen ist die Laufzeit von ST aufgrund der mehrfachen Iterationen (z.B. durchschnittlich 23 Itera-

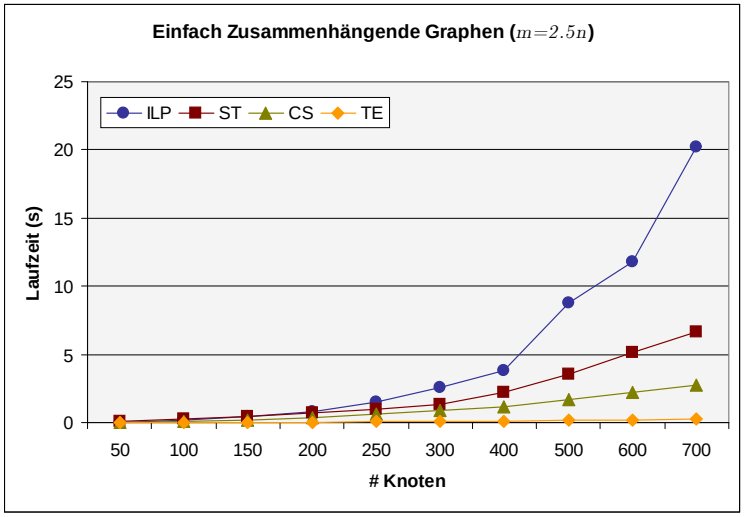

Abbildung 6.30: Die Laufzeiten der vier Verfahren bei den einfach zusammenhängenden Graphen mit $m = 2.5n$ in Bezug auf die Anzahl der Knoten im Graph.

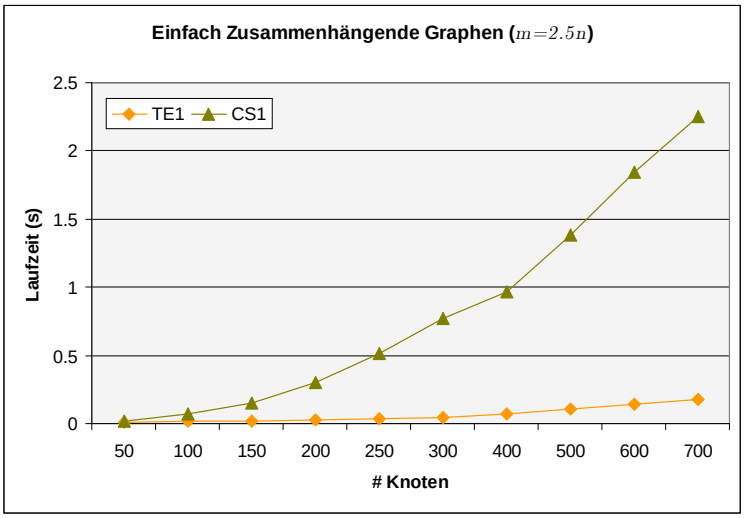

Abbildung 6.31: Die Laufzeiten von CS1 und TE1 bei den einfach zusammenhängenden Graphen mit $m = 2.5n$ in Bezug auf die Anzahl der Knoten im Graph.

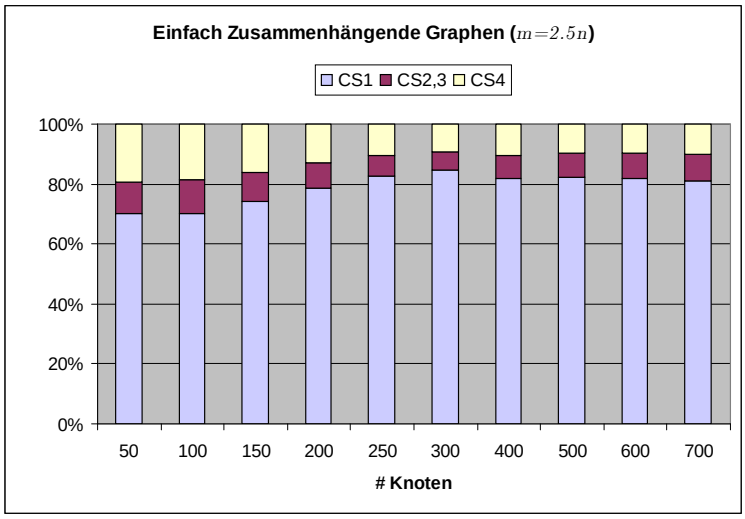

Abbildung 6.32: Verteilung der Gesamtlaufzeit des CS auf die einzelnen Schritte des Algorithmus in Bezug auf die Anzahl der Knoten im Graph.

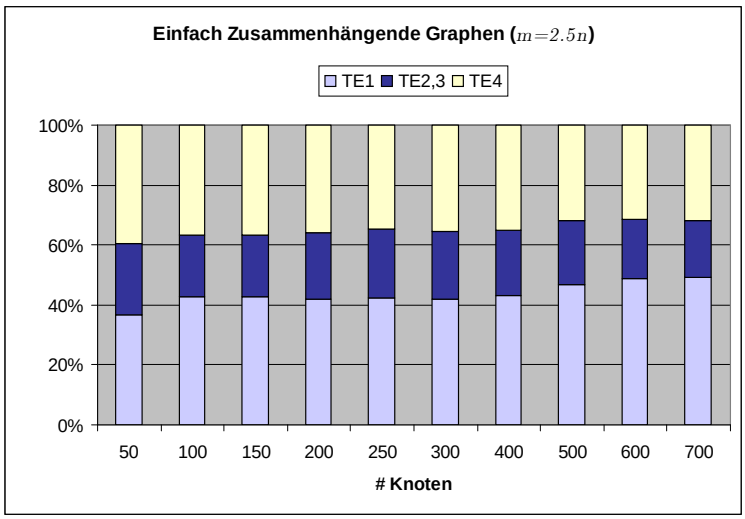

Abbildung 6.33: Verteilung der Gesamtlaufzeit des TE auf die einzelnen Schritte des Algorithmus in Bezug auf die Anzahl der Knoten im Graph.

tionen bei Bäumen mit $n = 700$) um ein vielfaches höher als die der anderen Verfahren. Die Laufzeiten liegen bei diesen relativ einfachen Instanzen für alle vier Verfahren unter einer Sekunde.

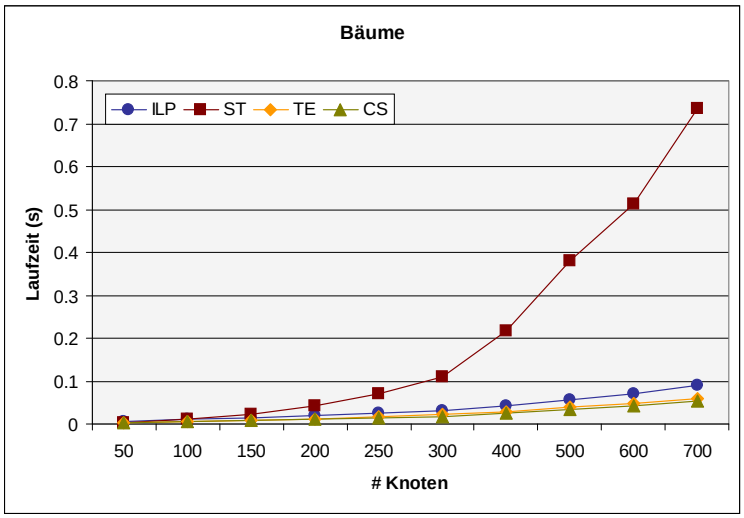

Abbildung 6.34: Laufzeiten der vier Verfahren bei den Bäumen in Bezug auf die Anzahl der Knoten im Graph.

Die Abb. 6.35 ermöglicht einen Vergleich zwischen den Laufzeiten des ILPs bei Graphen aus unterschiedlichen Klassen mit gleicher (oder ähnlicher) Dichte. Die Laufzeiten von CS und TE verhalten sich ähnlich.

Die größte Laufzeit benötigen alle Verfahren bei den vollständigen Graphen mit $n \geq 30$ (Abb. 6.36).

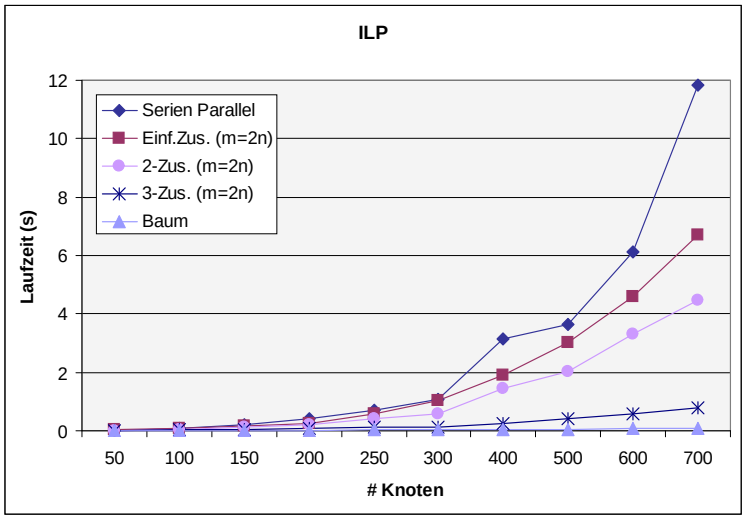

Abbildung 6.35: Laufzeit des ILPs bei den Graphen mit $m \simeq 2n$ und bei Bäumen in Bezug auf die Anzahl der Knoten im Graph.

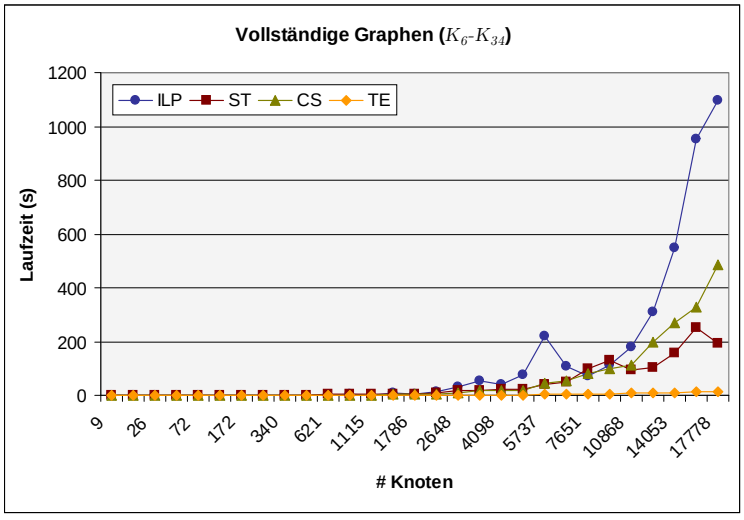

Abbildung 6.36: Die Laufzeit der vier Verfahren bei den vollständigen Graphen $K_6 - K_{34}$ in Bezug auf die Anzahl der Knoten im planarisierten Graph.

6.3.3 Zusammenfassung der experimentellen Ergebnisse

In Abschn. 5.5 hatten wir festgestellt, dass

$$z_{TE1} \leq z_{CS1} \leq z_{opt} \tag{6.1}$$

ist, also die Ausgangssituation vor der Korrektur bei CS *näher* am Optimum liegt, als die bei TE. Eine direkte Auswirkung dieses Umstandes haben wir bei unseren Tests bereits festgestellt. Die CS1 Lösung liegt für einen größeren Anteil der Test Graphen direkt am ganzzahligen Optimum: sie ist für 10783 Rome Graphen (ca. 93.5%) und 3803 RP Graphen (ca. 27.1%) zulässig, während die TE1 Lösung für 5325 Rome Graphen (ca. 46%) und 879 RP Graphen (ca. 6.3%) zulässig ist, also keine überfüllten Bündel enthält. Für weitere 736 Rome Graphen (ca. 6.4%) und 6695 RP Graphen (ca. 20.6%) liegt die CS1 Lösung so nahe am Optimum, dass für die Lösung nach der Korrektur (CS4) die Optimalität garantiert werden kann.

Die Tabellen 6.14 und 6.15 fassen diese Ergebnisse zusammen. Was den Anteil der optimal gelösten Instanzen betrifft, schneidet CS deutlich besser ab als die beiden anderen Ansätze. 76.4% aller RP Graphen werden von CS optimal gelöst, während ST 23.6% und TE 15.5% dieser Instanzen optimal löst. Ausserdem löst CS 100%, ST 88.5% und TE 74.1% aller Rome Graphen optimal.

Rome Graphen	CS	ST	TE
Triviale Instanzen	93.5%	46.2%	46.2%
Optimal gelöste Instanzen, mit Opt-Garantie	6.4%	–	–
Optimal gelöste Instanzen, ohne Opt-Garantie	0.1%	42.3%	27.9%
Nicht optimal gelöste Instanzen	–	11.5%	25.9%

Tabelle 6.14: Eine Zusammenfassung der Ergebnisse der drei Verfahren auf den Rome-Graphen

Was die durchschnittlichen absoluten und relativen Abweichungen der CS, ST und TE Lösungen vom ILP-Optimum bei den unterschiedlichen Graphklassen betrifft, sieht es nicht anders aus: CS hat jedes Mal eine deutlich ge-

RP Graphen	CS	ST	TE
Triviale Instanzen	27.1%	6.3%	6.3%
Optimal gelöste Instanzen, mit Opt-Garantie	20.6%	–	–
Optimal gelöste Instanzen, ohne Opt-Garantie	28.7%	17.3%	9.2%
Nicht optimal gelöste Instanzen	23.6%	76.4%	84.5%

Tabelle 6.15: Eine Zusammenfassung der Ergebnisse der drei Verfahren auf den Random-Planar Graphen zusammen mit den planarisierten vollständigen Graphen

ringere Abweichung. Über alle RP-Graphen betrachtet, liegt die obere Grenze für die durchschnittliche *absolute* Abweichung für TE bei 160.26, für ST bei 54.89 und für CS bei 4.88. Die obere Grenze für die durchschnittliche *relative* Abweichung liegt für TE bei 9.09%, für ST bei 4.45% und für CS bei 0.24%.

Wir haben festgestellt, dass der Maximalgrad des Eingabegraphen den größten Einfluss auf die Lösung aller drei Verfahren hat. Die Größe der Instanz hat auf die Lösungen von CS und ST keinen so großen Einfluss, während die TE Lösung deutlich stärker davon abängt. Die Ergebnisse auf den vollständigen Graphen sind ein sehr gutes Beispiel dafür.

Ein Anstieg in der Dichte der Instanz führt bei CS zu keiner Verschlächterung, wenn sich dabei der Maximalgrad nicht sehr verändert. Bei den Lösungen von ST und TE hingegen ist dabei eine deutliche Verschlächterung bemerkbar. Die Ergebnisse auf den 3-zusammenhängenden Graphen mit Dichte 1.5, 2 und 2.5 sind dafür ein gutes Beispiel.

6.4 Kandinsky Zeichnungen

Die folgenden vier Kandinsky Zeichnungen sind alle von CS berechnet und knickminimal. Sie sollen die in den vorigen Abschnitten erwähnten Unterschiede zwischen den Graphen aus den unterschiedlichen Klassen veranschaulichen.

Ein Vergleich der Abbildungen 6.37 und 6.38 zeigt den Unterschied im Maximalgrad zwischen zwei unterschiedlich generierten maximal planaren Graphen mit 50 Knoten. Der mit dem *Planar Triconnected* Generator erstellte Graph (Abb. 6.37) hat den Maximalgrad 9, während der mit dem

Random Planar Generator erstellte Graph (Abb. 6.38) den Maximalgrad 31 besitzt. Der serien-parallele Graph in Abb. 6.39 mit 50 Knoten weist sogar einen Maximalgrad von 41 auf. Die Abb. 6.40 zeigt einen Baum mit 50 Knoten und Maximalgrad 8.

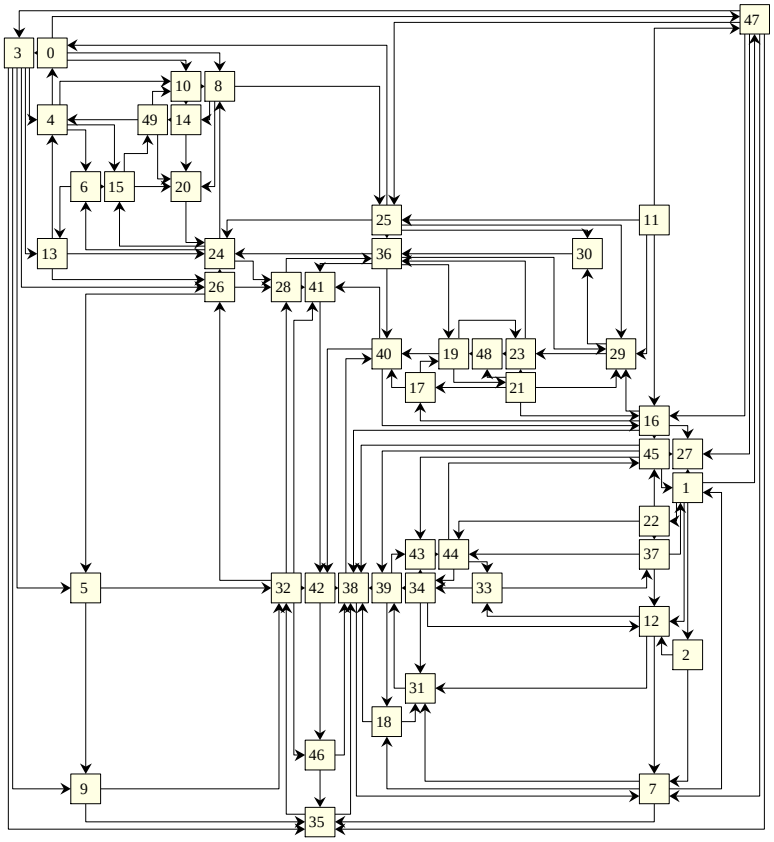

Abbildung 6.37: Die von CS berechnete Kandinsky Darstellung eines mit dem *Planar Triconnected* Generator erstellten maximal planaren Graphen mit 50 Knoten und Maximalgrad 9. $z_{opt} = z_{CS} = z_{ST} = 99$, $z_{TE} = 106$.

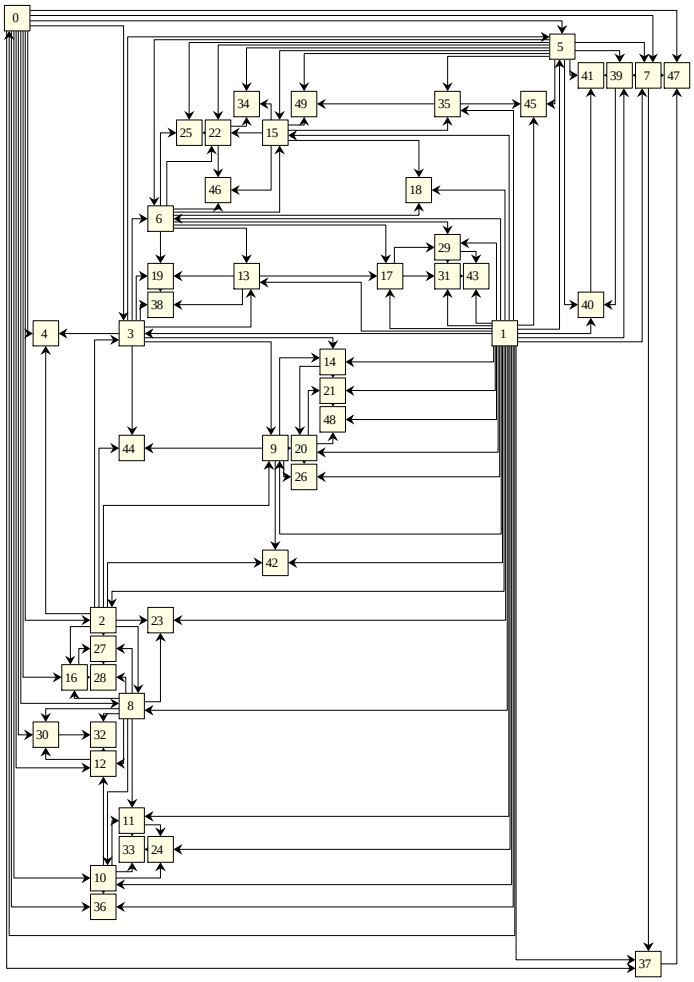

Abbildung 6.38: Die von CS berechnete Kandinsky Darstellung eines mit dem *Random Planar* Generator erstellten maximal planaren Graphen mit 50 Knoten und Maximalgrad 31. $z_{opt} = z_{CS} = 114$, $z_{ST} = 118$, $z_{TE} = 131$.

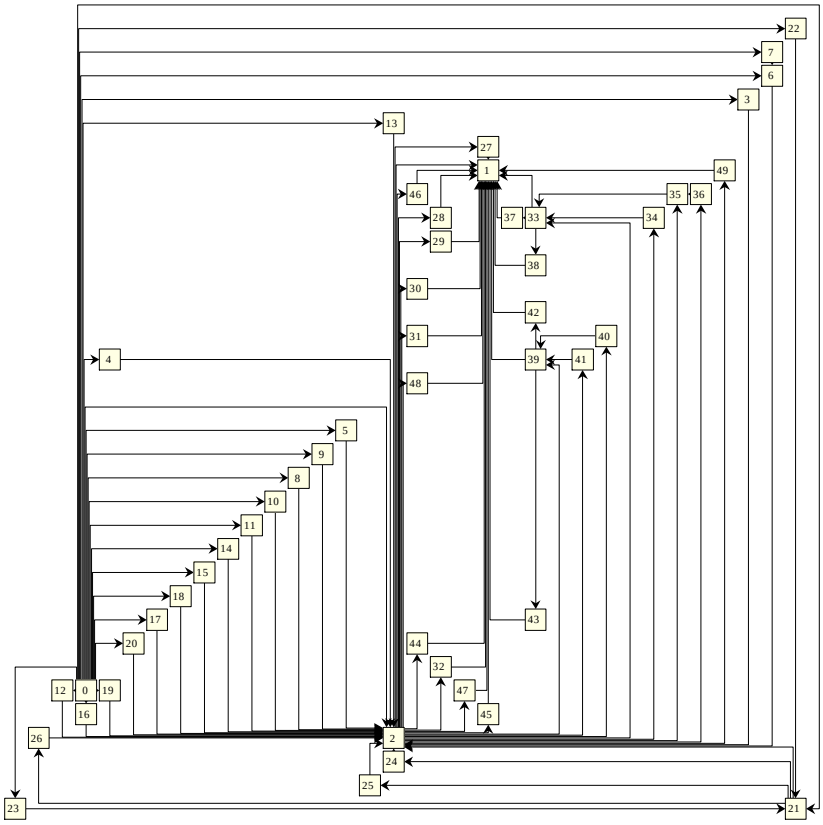

Abbildung 6.39: Die von CS berechnete Kandinsky Darstellung eines serien-parallelen Graphen mit 50 Knoten, 96 Kanten und Maximalgrad 41. $z_{opt} = z_{CS} = 83$, $z_{ST} = 84$, $z_{TE} = 87$.

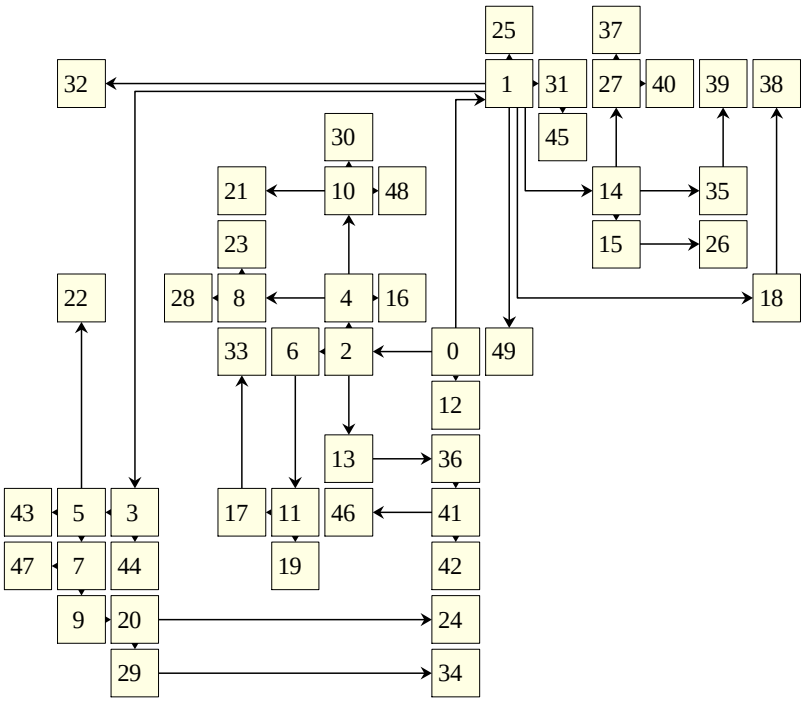

Abbildung 6.40: Die von CS berechnete Kandinsky Darstellung eines Baumes mit 50 Knoten und Maximalgrad 8. $z_{opt} = z_{CS} = z_{ST} = z_{TE} = 4$.

Die Abbildungen 6.41 und 6.42 zeigen jeweils die von CS und TE berechneten Kandinsky Zeichnungen desselben Graphen mit 50 Knoten und 75 Kanten. Die von CS berechnete Zeichnung hat 37 Knicke und ist somit knickminimal, während die von TE berechnete Zeichnung 42 Knicke enthält.

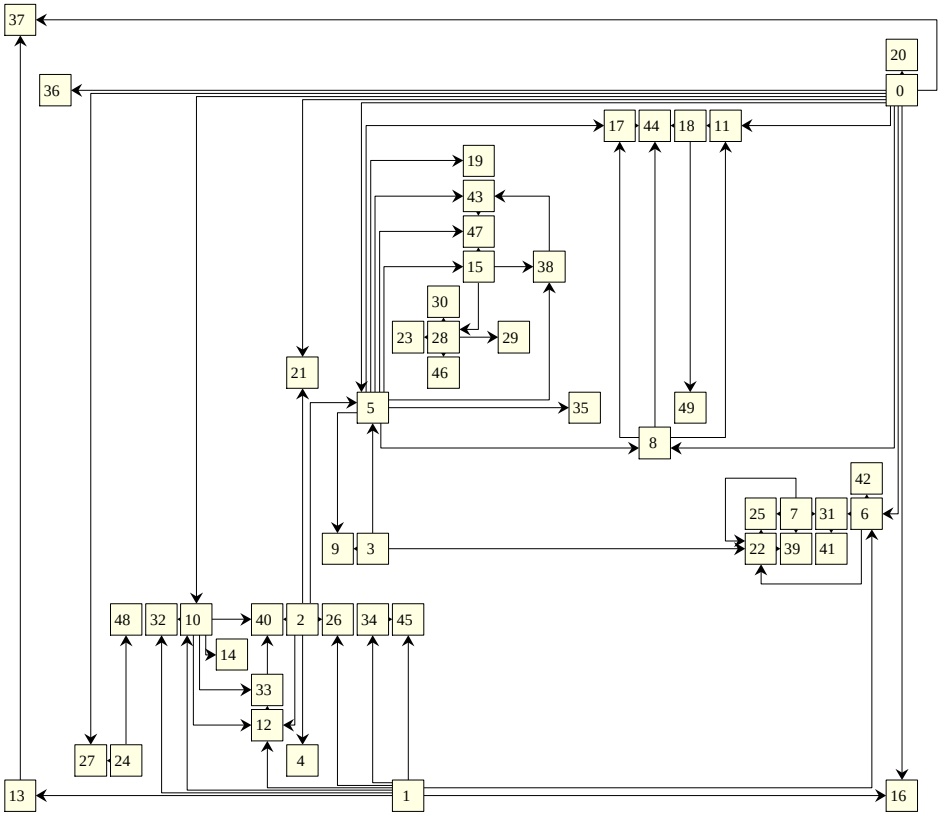

Abbildung 6.41: Die von CS berechnete Kandinsky Darstellung eines einfach zusammenhängenden planaren Graphen mit 50 Knoten und Dichte 1.5. $z_{opt} = z_{CS} = 37$, $z_{TE} = 42$.

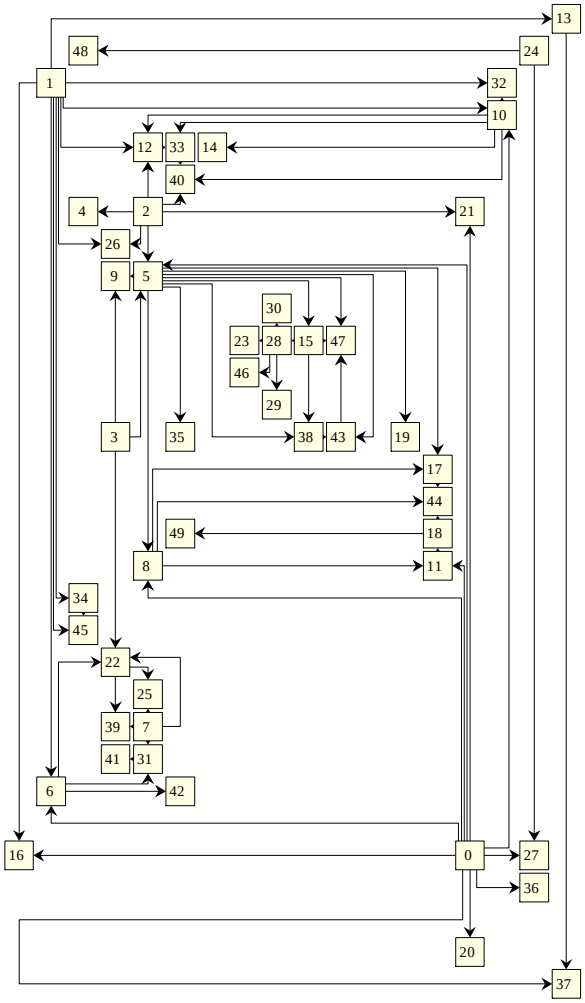

Abbildung 6.42: Die von TE berechnete Kandinsky Darstellung eines einfach zusammenhängenden planaren Graphen mit 50 Knoten und Dichte 1.5. $z_{opt} = z_{CS} = 37$, $z_{TE} = 42$.

Kapitel 7

Parametrisierte Komplexität

Nach einer kurzen Einführung in die Theorie der parametrisierten Komplexität (Abschn. 7.1) wird in Abschn. 7.2 eine Variation des Kandinsky Modells vorgestellt. Es wird gezeigt, dass das Knickminimierungsproblem für Zeichnungen in diesem Modell Fest-Parameter behandelbar ist.

7.1 Fest-Parameter Behandelbarkeit

Anfangs eine einleitende Definition:

Definition 7.1 *Eine formale Sprache L ist eine Menge von Wörtern über einem Alphabet $\sum$. Die Menge aller endlichen Wörter über einem Alphabet $\sum$ ist durch $\sum^*$ gegeben. Welche Wörter über einem Alphabet zur Sprache gehören und welche nicht, wird meist durch die Regeln einer formalen Grammatik bestimmt.*

Die folgenden Definitionen sind aus [Fel02] entnommen:

Definition 7.2 *Eine parametrisierte Sprache L ist eine Teilmenge $L \subseteq \sum^* \times \sum^*$. Ist L eine parametrisierte Sprache und $(x, k) \in L$ ist, dann bezeichnen wir x als den **Hauptteil**, und k als den **Parameter**.*

Der Parameter kann nicht-numerisch sein, es kann sich auch aus verschiedenen Teilen oder strukturellen Eigenschaften der Eingabe zusammensetzen.

Oftmals sind Probleme durch eine Eingabe definiert, die sich aus bestimmten Informationen und einer positiven ganzen Zahl zusammensetzt. Als Beispiel seien hier Graphentheoretische Probleme genannt, bei denen der Input meist aus einem Graphen $G = (V, E)$ und einer positiven ganzen Zahl k besteht. Der Graph G kann in diesem Fall als das Hauptteil, die Zahl k als der Parameter betrachtet werden.

Definition 7.3 *Eine parametrisierte Sprache L ist Fest-Parameter behandelbar, wenn es einen Algorithmus gibt, der in Zeit $f(k)q(|x|)$ entscheiden kann, ob $(x, k) \in L$ ist, wobei f eine beliebige Funktion und q ein Polynom in $|x|$ ist. Die Menge aller Fest-Parameter behandelbaren Probleme/Sprachen wird mit $\mathcal{FPT}$ bezeichnet.*

Das VERTEX COVER ist eines der bekanntesten Fest-Parameter behandelbaren Probleme in der Graphentheorie:

Definition 7.4 ***VERTEX COVER*** *Sei $\mathcal{G}$ die Menge aller Graphen und sei $L_{VC} \subseteq \mathcal{G} \times \mathbb{N}$ definiert durch*

$$L_{VC} = \{(G(V, E), k) : \exists V' \subseteq V, |V'| \leq k, \forall (u, v) \in E : u \in V' \vee v \in V'\}$$

Das Vertex Cover Problem kann nun folgendermassen definiert werden:
Eingang: Ein Graph $G = (V, E)$ und eine positive ganze zahl k.
Problemstellung: Ist $(G, k) \in L_{VC}$?

Dieses Problem ist bekanntlich NP-schwer. In [BG89] führt Buss den ersten Fest-Parameter Algorithmus mit der Laufzeit $O(kn+2^k k^{2k+2})$ für das Vertex Cover Problem ein. Downey und Fellows verbessern diese obere Schranke in [DF95] zu $O(kn+2^k k^2)$. In [BFR98] stellen die Autoren einen Algorithmus mit Laufzeit $O(kn + 1.324718^k k^2)$ vor, welches in [DF99] mit einer geringfügigen Verbesserung auf $O(kn + 1.31951^k k^2)$ gesenkt wird. In [NR99] können Niedermeier und Rosmanith diese Laufzeit weiter auf $O(kn + 1.29175^k k^2)$ senken. Der zur Zeit beste Algorithmus zur Lösung dieses Problems hat die Laufzeit $O(1.2852^k + kn)$ [CKJ01]. Dieser ist implementiert und anwendbar für ein unbegrenztes n und für ein k bis zu 400.

Es erweist sich also als sinnvoll, bei schweren Problemen die Eingabe in ein Hauptteil x und einen Parameter k zu zerlegen, sodass die Laufzeit polynomial in $|x|$ ist und nur in $|k|$ exponentiell werden kann. Somit können für praxisnahe Probleminstanzen mit relativ kleinem (beschränktem) Parameter effiziente Algorithmen entwickelt werden.

7.2 Eine Variation des Kandinsky Models

Wir betrachten eine Variante des Kandinsky Models, in der 0°-Winkel nur an Knoten mit Grad ≥ 5 erlaubt sind und das wir als das High Degree Kandinsky Modell bezeichnen. Dieses Modell kann als eine Erweiterung des Tamassia Modells betrachtet werden, nur so weit modifiziert, dass es auf Graphen mit höherem Knotengrad angewendet werden kann, ohne dass die Knoten mit der Anzahl der inzidenten Kanten wachsen.

Das High Degree Kandinsky Modell hat weniger Einschränkungen als das in Abschn. 5.3 vorgestellte Simple Kandinsky Modell, da es sowohl Knotenknicke in beide Richtungen als auch 0°-Winkel an Knoten mit höherem Grad erlaubt.

7.2.1 Eine Parametrisierte Analyse des High Degree Kandinsky

Wie bereits erwähnt, ist bisher kein effizienter Algorithmus zur Lösung des KMCF Problems bekannt. Mit Hilfe von modifizierten MCF Algorithmen können Lösungen berechnet werden, die von jedem Bündel höchstens eine Kante benutzen und somit gültig sind, aber nicht immer optimal. Da von jedem Bündel mindestens eine Kante mit Sicherheit nicht in der optimalen Lösung enthalten ist, müsste für jedes Bündel entschieden werden, welche dieser beiden Kanten das ist. Diese Bündelkanten könnten dann aus dem Netzwerk entfernt werden, wodurch das MCF Problem im restlichen Netzwerk mit den klassischen Algorithmen in polynomialer Zeit gelöst werden könnte. Es konnten bisher aber keine Optimalitätskriterien formuliert werden, mit deren Hilfe auf die Optimalität oder nicht Optimalität einer solchen

Auswahl von Bündelkanten geschlossen werden kann. Es bleibt nichts anderes übrig, als das MCF Problem für alle möglichen $2^{|\bar{E}|}$ Kombinationen zu lösen. Die $|\bar{E}|$ Bündel bilden folglich den Teil des KMCF Problems, der für die exponentielle Laufzeit ($O(2^{|\bar{E}|})$) des exakten Algorithmus verantwortlich ist.

Die Hauptmotivation für die Einführung des Kandinsky Modells war, die Erstellung orthogonaler Zeichenungen mit fixer Knotengröße für Graphen mit einem höheren Knotengrad zu ermöglichen. Dazu wurden bekanntlich die 0°-Winkel eingeführt, die es mehreren Kanten erlauben, inzident zu derselben Seite eines Knotens zu sein, was bei einem Knotengrad ≥ 5 unumgänglich ist. Verzichten wir an Knoten mit Grad ≤ 4 auf die 0°-Winkel, die nicht unbedingt notwendig sind, können wir jene Bündel, die für Kanten $(u, v) \in \bar{E}$ mit $\deg(v) \leq 4$ eingeführt wurden, aus dem Kandinsky Netzwerk entfernen. Denn gibt es keine 0°-Winkel an einem Knoten v, müssen keine zugehörigen Knotenknicke erzwungen werden, und genau dazu hatten die Bündel gedient. In dem neuen Netzwerk, das wir als das High Degree Kandinsky Netzwerk bezeichnen, ist die Menge A_{FH} der Bündelkanten nun durch

$$A_{FH} = \bigcup_{(u,v)\in\bar{E}:\deg(v)\geq 5} B_{u,v}$$

gegeben, wobei $B_{u,v}$ wie in Def. 4.3 definiert ist. Der Rest des Netzwerkes bleibt unverändert. Ist $V_{HD} = \{v \in V : \deg(v) \geq 5\}$, haben wir die Anzahl der Bündel von $|\bar{E}|$ auf

$$\sum_{v\in V_{HD}} \deg(v).$$

reduziert.

Definition 7.5 *Jeder gültige Fluss auf einem High Degree Kandinsky Netzwerk entspricht einer* ***High Degree Kandinsky Darstellung****. Eine Kandinsky Zeichnung, die keine* 0°*-Winkel an Knoten* v *mit* $\deg(v) \leq 4$ *besitzt, bezeichnen wir als High Degree Kandinsky Zeichnung.*

Sei k nun die Anzahl der Knicke in einer High Degree Kandinsky Zeichnung eines Graphen $G = (E, V, F)$ mit planarer Einbettung P. Wir können

an dieser Stelle folgendes schließen:

Folgerung 7.1 *Für alle Knoten $v \in V_{HD}$ gilt: Mindestens eines der zu v inzidenten Kanten besitzt einen Knotenknick der eindeutig einem $0°$-Winkel an dem Knoten v zuordenbar ist. Daraus ergibt sich:*

$$|V_{HD}| \leq k \tag{7.1}$$

Folgerung 7.2 *Sei $v \in V_{HD}$. Von allen zu v inzidenten Kanten verlaufen höchstens 4 geradeaus. Alle anderen $\deg(v)-4$ Kanten besitzen mit Sicherheit einen Knotenknick, die eindeutig den $\deg(v) - 4$ $0°$-Winkeln an dem Knoten v zuordenbar sind. Daraus ergibt sich:*

$$\deg(v) - 4 \leq k \implies \deg(v) \leq k + 4 \tag{7.2}$$

Somit gilt für die Anzahl der Bündel:

$$\sum_{v \in V_{HD}} \deg(v) \leq \sum_{v \in V_{HD}} \Delta = |V_{HD}| \cdot \Delta \leq k(k+4) \tag{7.3}$$

wobei $\Delta = \max\{\deg(v) : v \in V_{HD}\}$.

Ein exakter Algorithmus, der aus jedem dieser Bündel eine Kante auswählt, diese aus dem Netzwerk löscht und im restlichen Netzwerk das MCF Problem löst; und das für jede mögliche Auswahl von Bündelkanten macht, hätte eine Laufzeit von

$$O\left(2^{k(k+4)} \cdot p(|N|, |A|)\right) \tag{7.4}$$

wobei p eine polynomiale Funktion ist. Diese Laufzeit kann, unter Berücksichtigung folgender Tatschen verbessert werden.

Sei $v \in V_{HD}$ und $n_v \in N_{HD}$ der dazugehörige Knoten im Netzwerk, wobei $N_{HD} = \{n_v \in N : v \in V_{HD}\}$ ist.

- Suchen wir uns für jeden der vier 90°-Winkel, die der Knoten v sozusagen zu vergeben hat, eine Kante (u, v) aus der Menge der zu v inzidenten Kanten aus und platzieren diesen Winkel in die Fläche links

von (u, v). Da an allen Knoten auch Winkel $> 90°$ erlaubt sind, kann eine Kante mehrmals gewählt werden. Somit gibt es $\deg(v)^4$ Möglichkeiten, diese Auswahl zu treffen.

  - In die Netzwerk-Sprache übersetzt bedeutet das folgendes: Für jeden der vier Einheiten, den der Knoten n_v zu vergeben hat, haben wir eine Kante (u, v) ausgewählt und diese Einheit an den Knoten $h_{u,v}$, und von dort aus in die Fläche $\text{face}(u, v)$ geschickt (Abb. 7.1,(a)). Für die ausgewählten (u, v) ist die untere Schranke der Kante $(h_{u,v}, n_{\text{face}(u,v)})$ erfüllt und die $f - h$ Kanten (Bündelkanten), die in $h_{u,v}$ münden, können aus dem Netzwerk entfernt werden. Wie gesagt: nur für diese feste Auswahl.

- Für jede feste Einteilung der Winkel um v steht auch fest, welche der zu v inzidenten Kanten von derselben Seite des Knotens ausgehen (Abb. 7.1,(b)). Es ist klar, dass von den zu derselben Seite inzidenten Kanten höchstens eine geradeaus verlaufen kann. Für die restlichen Kanten gilt: die Kanten rechts von der geraden Kante müssen einen Rechts-Knotenknick, die Kanten links davon einen Links-Knotenknick machen. Setzen wir für jede Seite von v eine Kante als die gerade Kante fest (in Abb. 7.1,(c) durch Pünktchen markiert), ist dadurch das Knoten-Knick-Verhalten der anderen Kanten auf dieser Seite eindeutig festgelegt. Hat v auf einer Seite nur eine inzidente Kante, wird diese als die gerade Kante festgelegt. Hat sie keine inzidenten Kanten auf einer Seite, wird nichts gemacht. Wieviele Möglichkeiten stehen uns für die Auswahl der geraden Kanten zur Verfügung? Im schlimmsten Fall sind die Kanten gleichmäßig auf die vier Seiten des Knotens verteilt, womit wir für jede Seite $\deg(v)/4$ Möglichkeiten hätten. Insgesamt also höchstens $(\deg(v)/4)^4$ Möglichkeiten.

  - Im Netzwerk entspricht das folgendem Fall: Die Bündelkanten, die Knotenknicken auf einer geraden Kante entsprechen, können aus dem Netzwerk entfernt werden, da diese Kante keine Knotenknicke aufweisen muss. Für eine gerade Kante (u, v) sind das

die $f - h$ Kanten $a^R_{u,v}$ und $a^L_{u,v}$. Da alle Kanten rechts von (u, v) nach rechts knicken müssen, können die $f - h$ Kanten, die Links-Knotenknicken auf diesen Kanten entsprechen, ebenfalls aus dem Netzwerk entfernt werden. Analog können für die Kanten links von (u, v) die $f - h$ Kanten für Links-Knotenknicke entfernt werden.

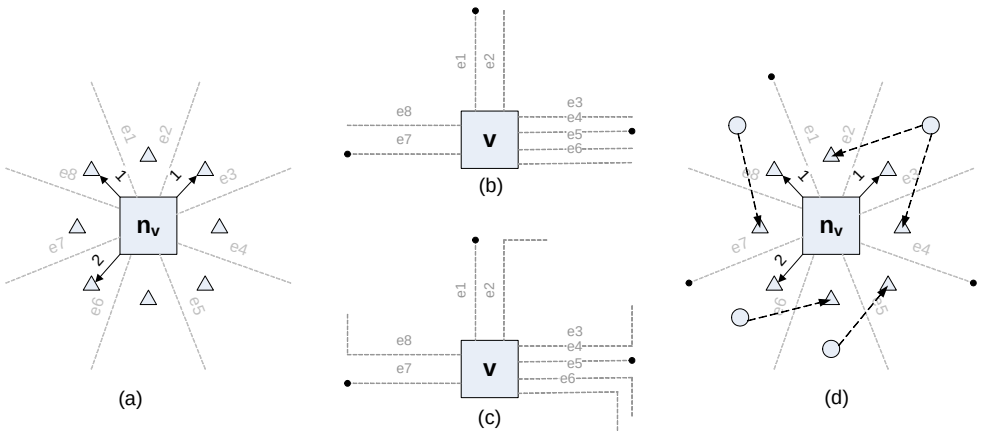

Abbildung 7.1: Einteilung der positiven Winkel und Festlegung der geraden Kanten

Damit ist mindestens eine Kante jedes Bündels um den Knoten n_v aus dem Netzwerk entfernt worden (Abb. 7.1,(d)). Macht man das für alle Knoten in V_{HD}, gibt es im Netzwerk keine zwei Kanten mehr, die zu demselben Bündel gehören, sodass das MCF Problem im restlichen Netzwerk in polynomialer Zeit gelöst werden kann.

Fassen wir zusammen: Für jeden Knoten $v \in V_{HD}$ gibt es $\deg(v)^4$ mögliche Einteilungen der Winkel, für jede dieser Einteilungen $(\deg(v)/4)^4$ mögliche Auswahlen der geraden Kanten, also insgesamt höchstens

$$\deg(v)^4 \cdot \left(\frac{\deg(v)}{4}\right)^4 \leq \deg(v)^8 \tag{7.5}$$

Möglichkeiten sich festzulegen. Bildet man das Produkt über alle Knoten in

V_{HD}:

$$\prod_{v \in V_{HD}} \deg(v)^8 \leq \prod_{v \in V_{HD}} \left(\Delta^8\right) = \Delta^{8|V_{HD}|} \quad \underset{\downarrow \; (7.1), (7.2)}{\leq} \quad (k+4)^{4k} \tag{7.6}$$

Ein exakter Algorithmus, der für alle $(k+4)^{4k}$ Möglichkeiten das MCF Problem löst und das beste Ergebnis zurückliefert, hätte somit eine Laufzeit von

$$O\left((k+4)^{4k} \cdot p(|N|, |A|)\right) \tag{7.7}$$

wobei p eine polynomiale Funktion und k die Anzahl der Knicke ist.

Kapitel 8

Zusammenfassung und Ausblick

In dieser Arbeit haben wir uns mit dem Knickminimierungsproblem bei Zeichnungen im Kandinsky Modell befasst. Das Problem wird dabei auf ein spezielles Min-Cost Flow Problem, das Kandinsky MCF Problem, zurückgeführt. Die Komplexität dieses Problems ist bisher unbekannt.

Die bisherigen Lösungsansätze für dieses Problem sind die von Eiglsperger in [Eig03] eingeführten zwei Verfahren: ein Approximationsalgorithmus mit Gütegarantie 2 und eine verbessernde Heuristik.

Wir stellen den von uns entwickelten Cyclic-Shift Algorithmus vor, der eine Näherungslösung für dieses Problem berechnet, die im schlimmsten Fall höchstens zwei Mal so schlecht wie die optimale Lösung ist. Dazu wird das Kandinsky MCF Problem zunächst als Ganzzahliges Lineares Programm formuliert. Ausgehend von der nicht ganzzahligen Lösung der LP-Relaxierung wird eine ganzzahlige Näherungslösung konstruiert.

Wir testen alle drei Verfahren und vergleichen die Ergebnisse mit den exakten Lösungen. Die Testergebnisse zeigen, dass unser Algorithmus in der Praxis viel bessere Ergebnisse als die bisherigen Ansätze liefert.

Wir definieren ausserdem eine Variation des Kandinsky Modells, in der $0°$-Winkel nur an Knoten mit Grad > 4 erlaubt sind und zeigen, dass das Knickminimierungsporblem in diesem Modell Fest Parameter behandelbar ist.

Einige offene Fragen, die für zukünftige Arbeiten in dieser Richtung in-

teressant sein könnten:

1- Die verbessernde Heuristik von Eiglsperger ist im Grunde ein wiederholtes Anwenden des Transformationsalgorithmus. Auf ähnliche Weise könnte auch durch wiederholtes Anwenden des Cyclic-Shift Algorithmus eine Verbesserung erzielt werden.

2- Da es bei der Lösung des KMCF Problems im Grunde auf die Auswahl der richtigen Bündelkanten ankommt, stellt sich die Frage, ob nicht Kriterien für eine optimale Auswahl der Bündelkanten formuliert werden können.

3- In dieser Arbeit haben wir gezeigt, dass die Laufzeit des Knickminimierungsproblems im High Degree Kandinsky Modell durch $O\left((k+4)^{4k} \cdot p(|N|, |A|)\right)$ beschränkt ist. Es wäre interessant ob diese obere Schranke weiter verbessert werden kann.

4- Letztendlich stellt die Komplexität des KMCF Problems weiterhin eine offene Forschungsfrage dar.

Literaturverzeichnis

[AMO93] Ahuja, R.K., Magnanti, T.L. und Orlin, J.B. *Network Flows.* Prentice-Hall, 1993.

[AGD] AGD – *Library of Algorithms for Graph Drawing.* Online-Manual. http://www.ads.tuwien.ac.at/AGD, v1.3, 2004.

[BDD00] Bertolazzi, P., Di Battista, G., Didimo W. *Computing Orthogonal Drawings with the Minimum Number of Bends.* IEEE Transactions on Computers, Band 49, No.8, 2000.

[BFR98] Balasubramanian, R., Fellows, M. R. und Raman, V. *An Improved Fixed Parameter Algorithm for Vertex Cover.* Info Process Lett 65, Seiten 163–168. 1998.

[BG89] Buss, J.F., und Goldsmith, J. *Nondeterminism Within P.* SIAM J. Comput., Band 22, Seiten 560-572, 1993.

[CKJ01] Chen, J., Kanj, I.A., und Jia, W. *Vertex Cover: Further Observations and Further Improvements.* In Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'99), Band 1665 der Reihe *Lecture Notes in Computer Science*, Seiten 313-324. Springer, 1999.

[CNAO85] Chiba, N., Nishizeki, T., Abe, S. und Ozawa T. A Linear Algorithm for Embedding Planar Graphs Using PQ-Trees. J. Comput. Syst. Sci., Band 30(1), Seiten 54-76, 1985.

[DDPP99] Di Battista, G., Didimo, W., Patrignani, M. und Pizzonia, M. *Orthogonal and Quasi-Upward Drawings with Vertices*

of Prescribed Size. In J.Kratochvil (Herausgeber): Proceedings of the 7th International Symposium on Graph Drawing (GD'99), Band 1731 der Reihe *Lecture Notes in Computer Science*, Seiten 297-310. Springer, 1999.

[DETT99] Di Battista, G., Eades, P., Tamassia, R. und Tollis, I.G. *Graph Drawing, Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999.

[DF95] Downey, R.G., und Fellows, M.R. *Parameterized Computational Feasibility.* In Clote, P. und Remmel., J. (Heruasgeber): Feasible mathematics II, Seiten 219-244. Birkhauser, Boston, 1995.

[DF99] Downey, R.G., Fellows, M.R. and Stege, U. *Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability.* In "Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future", Roberts, F., Kratochvil, J. und Nesetril, J. (Herausgeber): AMS-DIMACS Proceedings Series, Band 49, Seiten 49–99. AMS, 1999.

[DGLTTV95] Di Battista, G., Garg, A., Liotta, G., Tamassia, R., Tassinari, E., and Vargiu, F. *An Experimental Comparison of Three Graph Drawing Algorithms.* Proceedings of the 11th Annu. ACM Sympos. Comput. Geom., 1995

[Die00] Diestel, R. *Graphentheorie.* Elektronische Ausgabe: http://www.math.uni-hamburg.de/-home/diestel/books/graphentheorie/GraphentheorieII.pdf. Springer-Verlag Heidelberg 1996, 2000.

[DT81] Dolev, D. und Trickey, H. *On Linear Embedding of Planar Graphs.* Technical Report CS-81-876, Stanford Univ., 1981.

[EFK00] Eiglsperger, M., Fößmeier, U. und Kaufmann, M. *Orthogonal Graph Drawing with Constraints.* In Proceedings of the

eleventh annual ACM-SIAM Symposium on Discrete Algorithms, Seiten 3-11. 2000.

[Eig03] Eiglsperger, M. *Automatic Layout of UML Class Diagrams: A Topology-Shape-Metrics Approach.* Dissertation. Eberhard-Karls-Universität zu Tübingen, 2003.

[Fel02] Fellows, M.R. *Parametrized Complexity: The Main Ideas and Connections to Practical Computing.* In R. Fleischer (Herausgeber): Experimental Algorithms, Band 2547 der Reihe *Lecture Notes in Computer Science*, Seiten 51-77. Springer, 2002.

[FK96] Fößmeier, U. und Kaufmann, M. *Drawing High Degree Graphs with Low Bend Numbers.* In F.J. Brandenburg (Herausgeber): Proceedings of the 3rd International Symposium on Graph Drawing (GD'95), Band 1027 der Reihe *Lecture Notes in Computer Science*, Seiten 254-266. Springer, 1996.

[FK97] Fößmeier, U. und Kaufmann, M. *Algorithms and Area Bounds for Non-Planar Orthogonal Drawings.* In G. Di Battista (Herausgeber): Proceedings of the 5th International Symposium on Graph Drawing (GD'97), Band 1353 der Reihe *Lecture Notes in Computer Science*, Seiten 134-145. Springer, 1997.

[Foe97] Fößmeier, U. *Orthogonale Visualisierungstechniken für Graphen.* Dissertation. Eberhard-Karls-Universität zu Tübingen, 1997.

[GJ79] Garey, M.R. und Johnson, D.S. *Computers and Intractability A guide to the Theory of NP-completeness.* W. H. Freeman, San Francisco, 1979.

[GJ83] Garey, M.R. und Johnson, D.S. *Crossing Number is NP-Complete.* SIAM J. Algebraic Discrete Methods, Band 4(3), Seiten 312-316, 1983.

[GT95] Garg, A. und Tamassia, R. *On the Computational Complexity of Upward and Rectilinear Planarity Testing.* In R. Tamassia und I.G. Tollis (Herausgeber): Proceedings of the 2nd International Symposium on Graph Drawing (GD'94), Band 894 der Reihe *Lecture Notes in Computer Science*, Seiten 286-297. Springer, 1995.

[Har69] Harary, F. *Graph Theory.* Addison-Wesley, Philippines, 1969.

[LEDA] LEDA - *Library of Efficient Data Types and Algorithms.* Online Manual: http://www.algorithmic-solutions.com, Version 5.0, 2005.

[NR99] Niedermeier, R. und Rossmanith, P. *Upper Bounds for Vertex Cover Further Improved.* In Meinel., C. und Tison, S. (Herausgeber): Symposium on Theoretical Aspects of Computer Science (STACS'99), Band 1563 der Reihe *Lecture Notes in Computer Science,* Seiten 561–570. Springer, Berlin, 1999.

[NW88] Nemhauser, G.L. und Wolsey, L.A. *Integer and Combinatorial Optimization.* Wiley Interscience, 1988.

[NR04] Nishizeki, T. und Rahman, Md.S. *Planar Graph Drawing.* World Scientific Publishing, 2004.

[PS82] Papadimitriou, C. H. und Steiglitz, K. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[Tam87] Tamassia, R. *On Embedding a Graph in the Grid with the Minimum Number of Bends.* SIAM J. Comput., Band 16(3), Seiten 421-444, 1987.