Kompaktierung orthogonaler Zeichnungen

Entwicklung und Analyse eines IP-basierten Algorithmus

Alexander M. Esser

Masterarbeit
zur Erlangung des akademischen Grades
Master of Science (M. Sc.)
im Studiengang Wirtschaftsmathematik

angefertigt am Institut für Informatik
der Universität zu Köln
unter Anleitung von
Prof. Dr. Michael Jünger

Köln, im Juni 2014

Kurzzusammenfassung
Diese Arbeit befasst sich mit dem zweidimensionalen Kompaktierungsproblem für orthogonale Zeichnungen. Ziel ist es, allen Knoten Koordinaten zuzuweisen, sodass die Gesamtkantenlänge minimiert wird, die Form der Zeichnung aber erhalten bleibt. Viele Kompaktierungsheuristiken, insbesondere eindimensionale, treffen lokale Entscheidungen, beispielsweise, wie Dissecting-Kanten angeordnet werden sollen oder in welche Richtung zuerst kompaktiert wird. Ungünstige lokale Entscheidungen können jedoch zu einer höheren Gesamtkantenlänge führen. In dieser Arbeit wird daher eine zweidimensionale Kompaktierungsheuristik für 2-zusammenhängende, 4-planare Graphen vorgestellt, die die Ausrichtung der Dissecting-Kanten offen lässt und keine Richtung bevorzugt. Das Optimierungsproblem wird als IP formuliert, um die parallele Kompaktierung in beide Dimensionen zu gewährleisten. Erst beim Lösen dieses IP wird die Ausrichtung der Dissecting-Kanten festgelegt. Unter bestimmten Voraussetzungen ist die Nebenbedingungsmatrix des IP total unimodular. Eine experimentelle Analyse zeigt abschließend, dass der vorgestellte Ansatz teilweise Zeichnungen mit geringerer Gesamtkantenlänge liefert als bekannte Heuristiken.

Abstract
This thesis considers the two-dimensional compaction problem for orthogonal grid drawings. The task is to determine the coordinates of all vertices while preserving the shape of the drawing so that the total edge length is minimized. Many compaction heuristics, especially one-dimensional ones, make local decisions on how to arrange dissection edges or on compacting in one dimension first. These local decisions can – globally – destroy optimality. Therefore, in this thesis a two-dimensional compaction algorithm for biconnected, 4-planar graphs is presented which keeps the arrangement of dissection edges open and does not prefer any direction. The problem of minimizing the total edge length is formulated as integer linear program to ensure that the drawing is compacted in both dimensions simultaneously. The arrangement of dissection edges is not set until solving this IP in the compaction step. Under certain conditions the inequalities of the IP form a totally unimodular matrix. Finally, computational experiments show that on some instances the introduced heuristics leads to better results with respect to the total edge length than well-known heuristics.

Inhaltsverzeichnis

1 Einleitung

Ein Graph ist ein mathematisches Konstrukt, das aus Knoten und Kanten besteht. Um mit dem sogenannten Topology-Shape-Metrics-Ansatz eine Zeichnung eines Graphen zu erzeugen, müssen verschiedene Schritte durchlaufen werden, unter anderem müssen die Längen der Kanten und die Koordinaten der Knoten festgelegt werden. Dies geschieht im Rahmen der Kompaktierung. Ziel dabei ist es, die Gesamtkantenlänge, die Länge der längsten Kante oder die Zeichenfläche zu minimieren. An orthogonale Zeichnungen wird zusätzlich die Bedingung gestellt, dass alle Kanten ausschließlich horizontal oder vertikal verlaufen.

Mithilfe orthogonaler Zeichnungen können Zusammenhänge übersichtlich und leicht verständlich dargestellt werden, sie kommen daher in verschiedenen Anwendungsbereichen zum Einsatz: Viele schematische Darstellungen basieren auf orthogonalen Zeichnungen, beispielsweise UML-Diagramme, Klassenhierarchien oder Flussdiagramme im Bereich der Software-Entwicklung. Ein weiteres wichtiges Anwendungsgebiet ist die Entwicklung von Computerchips. Dabei werden Schaltkreise durch orthogonale Zeichnungen dargestellt. Um sicherzustellen, dass bei der Übermittlung von Signalen keine Fehler oder Verzögerungen auftreten, werden verschiedene Bedingungen an die zugrunde liegende Zeichnung gestellt, insbesondere hinsichtlich der Kantenlängen.

Gebräuchliche Heuristiken zur Kompaktierung orthogonaler Zeichnungen unterteilen das Problem in zwei eindimensionale Subprobleme – ein horizontales und ein vertikales. Wegen der fehlenden Parallelität ist jedoch nicht gewährleistet, dass die entstehende Zeichnung minimale Kantenlänge besitzt. Außerdem werden bei diesen Heuristiken alle Entscheidungen lokal getroffen; eine unglückliche lokale Entscheidung kann jedoch – global gesehen – zu einer höheren Kantenlänge führen.

In dieser Arbeit wird ein Algorithmus vorgestellt, der diese Schwachpunkte üblicher Kompaktierungsheuristiken umgeht. Das Kompaktierungsproblem wird als ganzzahliges lineares Programm formuliert. Dadurch wird gewährleistet, dass das horizontale und das vertikale Subproblem parallel gelöst werden. Indem die Ausrichtung künstlicher Kanten zunächst offen gelassen wird, werden lokale Entscheidungen vermieden. Es wird gezeigt, dass das lineare Programm unter gewissen Voraussetzungen eine ganzzahlige Optimallösung besitzt und dass der vorgestellte Algorithmus teilweise bessere Resultate liefert als übliche Kompaktierungsheuristiken.

Diese Arbeit ist wie folgt aufgebaut: In Kapitel 2 werden zunächst einige Grundbegriffe der Graphentheorie und der linearen Programmierung definiert. In Kapitel 3 werden der Topology-Shape-Metrics-Ansatz und zwei Kompaktierungsheuristiken vorgestellt.

Es werden Schwächen dieser Heuristiken diskutiert und Lösungsansätze präsentiert. Im vierten Kapitel wird der eigene Ansatz zur Kompaktierung vorgestellt. Anschließend werden die Korrektheit und Laufzeit sowie die Existenz einer ganzzahligen Optimallösung untersucht. Der eigene Ansatz wird im Rahmen einer experimentellen Analyse in Kapitel 5 mit anderen Algorithmen verglichen, bevor in Kapitel 6 ein Fazit gezogen wird.

2 Grundlegende Definitionen

In diesem Kapitel werden zunächst einige Grundbegriffe der Graphentheorie definiert. Anschließend werden lineare Programme erklärt, insbesondere das Minimum-Kosten-Fluss-Problem, das später beim Lösen des Kompaktierungsproblems von Bedeutung ist.

2.1 Graphenzeichnen

Ein Graph ist eine abstrakte Struktur, mit deren Hilfe Relationen zwischen Objekten modelliert werden können. Diese Objekte werden durch Knoten dargestellt, Beziehungen zwischen Objekten durch Kanten. Beim Graphenzeichnen werden geometrische Repräsentationen dieser Graphen konstruiert.

Die Definitionen in diesem Abschnitt stammen, wenn nicht anders vermerkt, aus [Di Battista et al. 1999, Kap. 1] und [Jünger u. Mutzel 2003]. Die Darstellung ist angelehnt an [Spisla 2012, Kap. 2].

Ein *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 $e = (u, v) \in E$ zwei Knoten u und v miteinander verbindet; u und v heißen *Endknoten* von e. Man bezeichnet e als *inzident* zu u und v und die beiden Knoten u und v als *adjazent*. Der *Grad* $\deg(v)$ eines Knotens v ist die Anzahl seiner inzidenten Kanten.

Falls die Endknoten einer Kante $e \in E$ identisch sind, ist e eine *Schleife*. Besitzen zwei Kanten $e, f \in E$ dieselben Endknoten, so nennt man e und f *parallel*. Ein Graph heißt *einfach*, wenn er keine Schleifen oder parallele Kanten besitzt.

Man unterscheidet zwischen gerichteten und ungerichteten Graphen. In einem *gerichteten Graphen* stellen die Kanten geordnete Paare von Knoten dar, in einem *ungerichteten Graphen* ungeordnete Paare von Knoten. Eine gerichtete Kante wird auch als *Bogen* bezeichnet und in Zeichnungen üblicherweise als Pfeil dargestellt. Vernachlässigt man die Orientierung eines gerichteten Graphen, so erhält man einen ungerichteten Graphen.

Seien $G = (V, E)$ und $G' = (V', E')$ zwei ungerichtete Graphen oder zwei gerichtete Graphen. Die *Vereinigung* der beiden Graphen ist durch $G \cup G' := (V \cup V', E \cup E')$ definiert, der *Schnitt* durch $G \cap G' := (V \cap V', E \cap E')$. G' heißt *Subgraph* von G, falls $V' \subseteq V$ und $E' \subseteq E$ gilt (vgl. [Diestel 2012, Kap. 1]).

Ein *Weg* W in einem Graphen $G = (V, E)$ ist eine Folge von Knoten $v_0, v_1, \ldots, v_k \in V$ mit $(v_{i-1}, v_i) \in E$ für $i = 1, 2, \ldots, k$. W hat die *Länge* k. Man schreibt auch $W = v_0 \xrightarrow{*} v_k$. Ein *Pfad* ist ein Weg, bei dem alle Knoten voneinander verschieden sind. Ein *Kreis* ist ein Weg $v_0 \xrightarrow{*} v_k$ der Länge $k \geq 2$, in dem $v_0 = v_k$ gilt und alle anderen Knoten voneinander verschieden sind. Analog zur Orientierung von Kanten unterscheidet man zwischen gerichteten und ungerichteten Wegen.

Ein gerichteter Graph heißt *azyklisch*, wenn er weder Schleifen noch gerichtete Kreise enthält. Eine *topologische Nummerierung* eines gerichteten Graphen $G = (V, E)$ ist eine Funktion $topnum : V \to \mathbb{N}$, die jedem Knoten v eine natürliche Zahl $topnum(v)$ zuordnet, sodass für jede Kante $(u, v) \in E$ gilt:

$$topnum(v) > topnum(u)$$

Voraussetzung für die Existenz einer topologischen Nummerierung ist, dass G azyklisch ist. Abbildung 2.1 zeigt ein Beispiel einer topologischen Nummerierung.

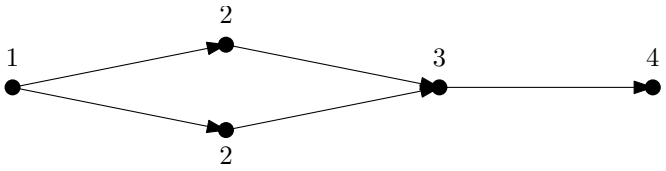


Abb. 2.1: Topologische Nummerierung

Ein Graph $G = (V, E)$ heißt *zusammenhängend*, wenn für je zwei beliebige Knoten $u, v \in V$ ein ungerichteter Weg $u \xrightarrow{*} v$ in G existiert; ansonsten heißt G unzusammenhängend. G ist *k-zusammenhängend*, wenn mindestens k Knoten aus V entfernt werden müssen, um G unzusammenhängend zu machen. Ein maximal zusammenhängender Subgraph von G wird als *Zusammenhangskomponente* bezeichnet.

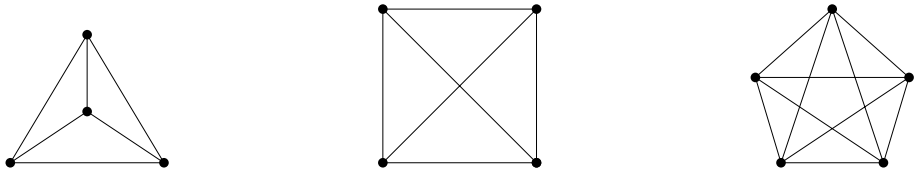
Eine *Zeichnung* oder ein *Layout* eines Graphen $G = (V, E)$ ist eine Funktion Γ, die alle Knoten $v \in V$ auf unterschiedliche Punkte $\Gamma(v)$ in der Ebene abbildet und alle Kanten $e = (u, v)$ auf eine offene Jordankurve $\Gamma(u, v)$ mit Endpunkten $\Gamma(u)$ und $\Gamma(v)$. Im Sprachgebrauch wird oft nicht zwischen einer Kante $e = (u, v)$ und deren Zeichnung $\Gamma(u, v)$ unterschieden, sondern beides mit e bezeichnet. Gleiches gilt für die Knoten.

Kanten in einer Zeichnung können sich im Allgemeinen schneiden. Auf spezielle Graphen, die ohne Kantenkreuzungen gezeichnet werden können, wird im folgenden Unterabschnitt eingegangen.

2.1.1 Planarität

Ein Graph $G = (V, E)$ heißt *planar*, wenn er in der Ebene gezeichnet werden kann, ohne dass sich Kanten schneiden. Eine solche Zeichnung bezeichnet man als *planare Zeichnung*. Beispiele von Planarität und Nicht-Planarität sind in Abbildung 2.2

dargestellt. Schneiden sich zwei Kanten in einer Zeichnung, so spricht man von einer *Kreuzung* oder *Kantenkreuzung*. G heißt *k-planar*, falls G planar ist und für jeden Knoten $v \in V$ $\deg(v) \leq k$ gilt. In den folgenden Kapiteln werden insbesondere 4-planare Graphen betrachtet.



(a) Planare Zeichnung eines planaren Graphen

(b) Nicht-planare Zeichnung eines planaren Graphen

(c) Nicht-planarer Graph

Abb. 2.2: Planarität

Eine planare Zeichnung unterteilt einen Graphen G in sogenannte *Flächen*. Genau eine der Flächen ist unbegrenzt, die *äußere Fläche*. Die anderen Flächen heißen *innere Flächen*.

Eine planare Zeichnung definiert für jeden Knoten v, in welcher Reihenfolge die inzidenten Kanten um v herum angeordnet sind. Eine *planare Einbettung* ist eine Äquivalenzklasse von planaren Zeichnungen mit identischen Anordnungen der inzidenten Kanten um jeden Knoten.

Sei $G = (V, E)$ ein planarer Graph mit planarer Einbettung $\Pi(G)$. F bezeichne die Menge der Flächen in G. Der *duale Graph* $G^* = (V^*, E^*)$ zu G wird wie folgt konstruiert: Jeder Knoten im primalen Graphen G entspricht einer Fläche in G^*, jede Fläche in G entspricht einem Knoten in G^*. Für jede primale Kante $e \in E$ existiert eine duale Kante $e^* \in E^*$. Offenbar trennt e zwei Flächen f', f'' voneinander. Die duale Kante e^* verbindet dann genau die beiden dualen Knoten, die zu den beiden Flächen f' und f'' korrespondieren. Der duale Graph ist also definiert durch:

$$V^* = F \quad \text{und} \quad E^* = \{(v_{f'}, v_{f''}) \in V^* \times V^* \mid \exists\, e \in E :\ e \text{ trennt } f' \text{ und } f''\}$$

Abbildung 2.3 zeigt einen Graphen (schwarz) und den zugehörigen dualen Graphen (rot).

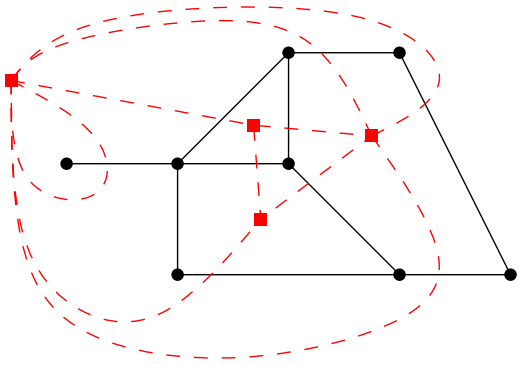


Abb. 2.3: Primaler und dualer Graph

2.1.2 Orthogonale Zeichnungen

In einer *orthogonalen Zeichnung*, auch *Gitterzeichnung* genannt, werden alle Knoten auf diskrete Gitterpunkte abgebildet. Jede Kante stellt eine Folge von horizontalen und vertikalen *Kantensegmenten* dar. Die Stelle, an der ein horizontales Kantensegment auf ein vertikales Kantensegment derselben Kante trifft, nennt man *Knick*. Ein Knick muss ebenfalls auf einem Gitterpunkt liegen. In Abbildung 2.4 ist eine orthogonale Zeichnung eines Graphen dargestellt.

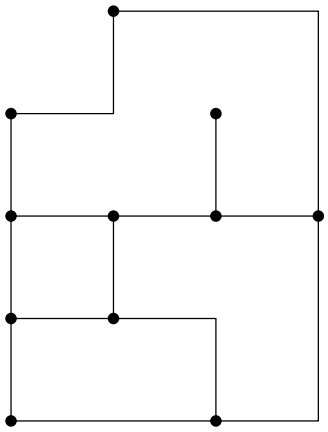


Abb. 2.4: Orthogonale Zeichnung

Eine orthogonale Zeichnung, die keine Knicke oder Kantenkreuzungen aufweist, heißt *einfach*. In einer einfachen orthogonalen Zeichnung lässt sich die Kantenmenge E in horizontale Kanten E_h und vertikale Kanten E_v unterteilen. Ein *horizontales Subsegment* ist ein zusammenhängender Subgraph in (V, E_h), ein *vertikales Subsegment* ein zusammenhängender Subgraph in (V, E_v). Ist ein Subsegment maximal zusammenhängend, so nennt man es *Segment*. Abbildung 2.5 zeigt die horizontalen und vertikalen Segmente einer einfachen orthogonalen Zeichnung (vgl. [Klau u. Mutzel 1999]).

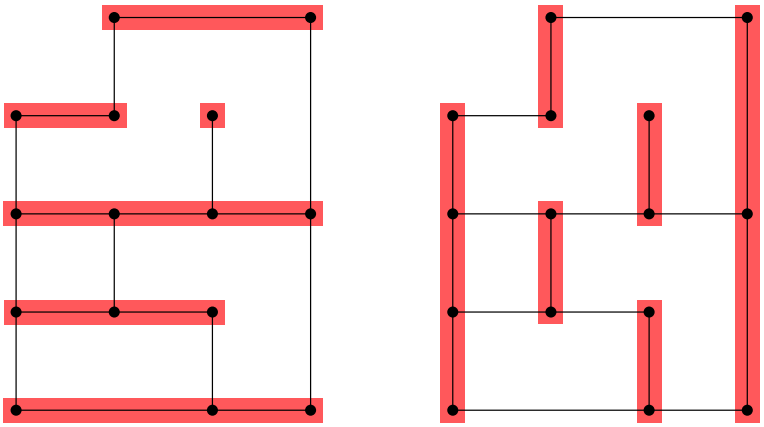


Abb. 2.5: Horizontale und vertikale Segmente

2.2 Lineare Programmierung

Mathematische Optimierung setzt sich mit Optimierungsproblemen, bestehend aus einer Zielfunktion und einer Menge von Nebenbedingungen, auseinander. Ziel ist es, eine optimale Lösung in Hinblick auf die gegebene Zielfunktion zu finden, wobei alle Nebenbedingungen eingehalten werden müssen. Lineare Programme stellen spezielle Optimierungsprobleme dar, bei denen sowohl die Zielfunktion als auch die Nebenbedingungen durch lineare Funktionen beschrieben werden können. Die Definitionen in diesem Abschnitt stammen, wenn nicht anders vermerkt, aus [Klau 2001, Kap. 2].

Sei $A \in \mathbb{R}^{m \times n}$ eine Matrix und seien $b \in \mathbb{R}^m$ und $c \in \mathbb{R}^n$ zwei Vektoren. Ein *lineares Programm* oder kurz *LP* besteht aus einer linearen Zielfunktion $c^T x$ und einem System von linearen Nebenbedingungen $Ax \leq b$. Ein Vektor $\bar{x} \in \mathbb{R}^n$ ist eine *zulässige Lösung* des Problems, falls $A\bar{x} \leq b$ gilt. Ziel der linearen Programmierung ist es, eine zulässige Lösung zu finden, die maximal in Hinblick auf die gegebene Zielfunktion ist, d. h. es wird ein Vektor x^* gesucht mit

$$c^T x^* = \max\{c^T x \mid Ax \leq b\}.$$

Das zugehörige lineare Programm wird üblicherweise geschrieben als:

$$\begin{array}{rl} \max & c^T x \\ \text{sodass} & Ax \leq b \end{array} \tag{LP}$$

Dieses Maximierungsproblem kann leicht in ein Minimierungsproblem transformiert werden. Daher werden im Folgenden ohne Einschränkung der Allgemeinheit ausschließlich Maximierungsprobleme betrachtet.

Lineare Programme können in polynomieller Zeit gelöst werden, beispielsweise mit dem Innere-Punkte-Verfahren von Karmarkar (vgl. [Karmarkar 1984]). In der Praxis hat sich das Simplex-Verfahren, erstmals vorgestellt in [Dantzig 1963], als einer der schnellsten Algorithmen zum Lösen linearer Programme erwiesen, auch wenn es im schlechtesten Fall exponentielle Laufzeit besitzt.

Lineare Programme werden häufig verwendet, um Anwendungsprobleme zu lösen. Die Lösungen des lineares Programms entsprechen dann den Lösungen des Anwendungsproblems. Dadurch ergeben sich oft Ganzzahligkeitsforderungen an die LP-Lösung oder an einzelne Komponenten des Lösungsvektors. Man spricht dann von einem *gemischt-ganzzahligen linearen Programm* oder *MIP*:

$$\begin{array}{rll} \max & c^T x & \\ \text{sodass} & Ax \leq b & \\ & x_i \in \mathbb{Z} & \forall\, i \in I \end{array} \tag{MIP}$$

Dabei stellt $I \subseteq \{1, \ldots, n\}$ eine Teilmenge der Variablenindizes dar.

Falls alle Komponenten ganzzahlig sein müssen, spricht man von einem *ganzzahligen linearen Programm* oder *IP*:

$$\begin{array}{rl} \max & c^T x \\ \text{sodass} & Ax \leq b \\ & x \in \mathbb{Z} \end{array} \tag{IP}$$

Wie Garey und Johnson 1979 bewiesen haben, ist es im Allgemeinen *NP*-schwer, ein IP zu lösen (vgl. [Garey u. Johnson 1979]).

Dennoch existieren verschiedene Lösungsstrategien: Eine Möglichkeit ist es, die Ganzzahligkeitsbedingungen fallen zu lassen und die *LP-Relaxierung* des IP zu lösen. Die LP-Relaxierung kann in polynomieller Zeit gelöst werden, allerdings weicht deren Lösung im Allgemeinen von der Lösung des IP ab. Bei einem Maximierungsproblem ist der Zielfunktionswert der LP-Relaxierung gleich oder größer dem Zielfunktionswert der IP-Lösung, da die Nebenbedingungen der LP-Relaxierung weniger restriktiv sind.

Ein anderer Ansatz zum Lösen eines IP sind Branch-and-Bound-Verfahren: Dabei wird das Ursprungsproblem mittels einer Divide-and-Conquer-Strategie rekursiv in Subprobleme unterteilt, es entsteht ein sogenannter Branch-and-Bound-Baum. Die Wurzel des Baumes entspricht dem Ursprungsproblem, die anderen Baumknoten korrespondieren zu den Subproblemen. Für jedes Subproblem wird eine lokale obere Schranke berechnet, indem einer der Variablen ein ganzzahliger Wert zugewiesen wird und die entstehende LP-Relaxierung gelöst wird. Der Algorithmus versucht in jedem Schritt, eine zuvor initialisierte globale untere Schranke sukzessive zu erhöhen. Ist der Zielfunktionswert eines Subproblems kleiner als die globale untere Schranke, wird dieses Subproblem verworfen. Sofern das IP lösbar und beschränkt ist, liefern Branch-and-Bound-Algorithmen immer die Optimallösung des IP, unter Umständen allerdings erst nach exponentiell vielen Schritten.

2.2.1 Das Minimum-Kosten-Fluss-Problem

Für einige spezielle Optimierungsprobleme sind polynomielle Lösungsverfahren bekannt, selbst wenn bei diesen Problemen ganzzahlige Lösungen gefordert werden. Das Minimum-Kosten-Fluss-Problem, kurz MKFP, ist eines dieser Probleme.

Ein *Netzwerk* $N = (U, A, b, l, u, c)$ ist ein gerichteter Graph bestehend aus einer Knotenmenge U und einer Kantenmenge A. Jeder Knoten $u \in U$ hat einen *Bedarf* $b_u \in \mathbb{R}$, wobei $\sum_{u \in U} b_u = 0$ gelten muss. Ist $b_u > 0$, so spricht man von einem *Angebot*, bei $b_u < 0$ von einer *Nachfrage*. Falls $b_u = 0$ ist, so ist u ein *Durchflussknoten*. Jede Kante $a \in A$ besitzt eine untere Schranke $l_a \in \mathbb{R}^{\geq 0}$ und eine obere Schranke $u_a \in \mathbb{R}^{\geq 0}$. Außerdem hat jede Kante $a \in A$ *Kosten* $c_a \in \mathbb{R}$ (vgl. [Ahuja et al. 1993, Kap. 1]).

Ein *Fluss* x in N weist jeder Kante a einen nichtnegativen Wert x_a zu, sodass gilt:

$$f_u(x) := \sum_{a=(u,w)\in A} x_a - \sum_{a=(v,u)\in A} x_a = b_u \quad \forall\, u \in U \qquad \text{(Flusserhaltungsbedingung)}$$

$$l_a \le x_a \le u_a \quad \forall\, a \in A \qquad \text{(Kapazitätsbedingung)}$$

Die *Kosten* des Flusses x sind $c^T x = \sum_{a\in A} c_a x_a$.

Das *Minimum-Kosten-Fluss-Problem* bei einem gegebenen Netzwerk N lautet wie folgt:

Finde einen Fluss x in N mit minimalen Kosten. (MKFP)

Formal ausgedrückt:

$$\begin{aligned} \min \quad & c^T x \\ \text{sodass} \quad & f_u(x) = b_u \qquad \forall\, u \in U \\ & l_a \le x_a \le u_a \qquad \forall\, a \in A \end{aligned} \tag{MKFP$'$}$$

Das Minimum-Kosten-Fluss-Problem kann in polynomieller Zeit gelöst werden (vgl. [Ahuja et al. 1993, Kap. 10]). Falls die Ganzzahligkeit der Nebenbedingungen und die Existenz einer Optimallösung gewährleistet sind, lässt sich immer eine ganzzahlige Optimallösung finden:

Satz 2.1
Sei $b_u \in \mathbb{Z}\ \forall u \in U$ und seien $l_a, u_a \in \mathbb{Z}\ \forall a \in A$. Dann besitzt das Minimum-Kosten-Fluss-Problem (MKFP$'$) entweder keine Optimallösung oder eine ganzzahlige Optimallösung.

Der detaillierte Beweis kann beispielsweise [Grötschel 2013, Kap. 12] entnommen werden. Er nutzt aus, dass die Matrix, die die Nebenbedingungen des Minimum-Kosten-Fluss-Problems beschreibt, total unimodular ist.

Eine quadratische Matrix $A \in \mathbb{Z}^{n\times n}$ heißt *unimodular*, wenn $\det(A) \in \{-1, 1\}$ gilt. Eine Matrix $A \in \mathbb{Z}^{m\times n}$ heißt *total unimodular*, wenn jede quadratische, nicht-singuläre Teilmatrix A' von A unimodular ist. Für lineare Programme, deren Nebenbedingungen durch eine total unimodulare Matrix beschrieben werden, kann generell gezeigt werden, dass sie eine ganzzahlige Lösung besitzen – wiederum vorausgesetzt, dass $b \in \mathbb{Z}$ gilt und die Existenz einer Optimallösung gesichert ist (vgl. [Grötschel 2013, Kap. 12]).

Satz 2.2
Sei A total unimodular und $b \in \mathbb{Z}$. Dann besitzt das lineare Programm (LP) entweder keine Optimallösung oder eine ganzzahlige Optimallösung.

3 Kompaktierung orthogonaler Zeichnungen

In diesem Kapitel wird zunächst der Topology-Shape-Metrics-Ansatz zum Erstellen orthogonaler Zeichnungen vorgestellt. Dabei wird besonders auf die Kompaktierungsphase des Ansatzes eingegangen. Es werden zwei eindimensionale Kompaktierungsheuristiken vorgestellt und deren Schwächen diskutiert. Anschließend werden zwei Lösungsansätze – die Methode von Klau und Mutzel und der Turn-Regularity-Ansatz – präsentiert.

3.1 Der Topology-Shape-Metrics-Ansatz

Orthogonale Zeichnungen kennzeichnen sich durch ausschließlich horizontal oder vertikal verlaufende Kanten. Sie sind daher gut für verschiedene praktische Anwendungen geeignet, beispielsweise für schematische Zeichnungen oder Entwürfe von Schaltkreisen. Schematische Zeichnungen sollten leicht verständlich und gut lesbar sein. Bei der Entwicklung von Schaltkreisen, dem VLSI-Design, wird darauf geachtet, dass bei der Übermittlung von Signalen keine Fehler oder Verzögerungen auftreten. An orthogonale Zeichnungen werden daher verschiedene Ästhetikkriterien gestellt:

- Die Anzahl der *Kantenkreuzungen* sollte so gering wie möglich sein.
- Die Anzahl der *Knicke* sollte möglichst klein sein.
- Die *Gesamtkantenlänge* und die *Länge der längsten Kante* sollten möglichst klein gehalten werden.
- Die *Fläche* der Zeichnung sollte so klein wie möglich sein.
- *Symmetrien* und *Hierarchien* im Graphen sollten erhalten bleiben.

Bereits die isolierte Optimierung hinsichtlich nur eines dieser Kriterien ist in den meisten Fällen *NP*-schwer. Erschwerend kommt hinzu, dass diese Kriterien miteinander in Konflikt stehen; Optimalität hinsichtlich eines Kriteriums kann Optimalität hinsichtlich eines anderes Kriteriums ausschließen (vgl. [Klau 2001, Kap. 1.1]).

Für orthogonale Zeichnungen hat sich die folgende Rangfolge der Kriterien durchgesetzt: Primäres Ziel ist es, die Anzahl der Kantenkreuzungen zu minimieren. Zweitens

sollte die Anzahl der Knicke möglichst gering sein. Drittens sollte die Gesamtkantenlänge möglichst klein gehalten werden (vgl. [Klau 2001, Kap. 1.1]).

Der Topology-Shape-Metrics-Ansatz, entwickelt von [Batini et al. 1986] und [Tamassia et al. 1988], erzeugt orthogonale Zeichnungen gemäß dieser Rangfolge. In drei Phasen werden die Topologie, Form und Geometrie der Zeichnung festgelegt (vgl. [Batini et al. 1986], [Tamassia et al. 1988], [Klau 2001, Kap. 1.1], [Spisla 2012, S. 8]).

Setze im Folgenden voraus, dass für alle Knoten v des Graphen $\deg(v) \leq 4$ gilt.

3.1.1 Planarisierung

In der ersten Phase, der Planarisierungsphase, wird der Graph planar eingebettet. Ziel dabei ist es, eine Zeichnung mit minimaler Anzahl an Kreuzungen zu finden. Dieses Problem, das *Kreuzungsminimierungsproblem*, ist *NP*-schwer (vgl. [Garey u. Johnson 1983]). Sollte der gegebene Graph nicht planar sein, werden Kantenkreuzungen durch künstliche Knoten ersetzt, um eine planare Einbettung zu ermöglichen (vgl. [Tamassia et al. 1988], [Di Battista et al. 1999, S. 20], [Patrignani 2013], [Spisla 2012, Kap 3.1]). In den folgenden Phasen des Topology-Shape-Metrics-Ansatzes[1] kann also davon ausgegangen werden, dass der Graph 4-planar ist.

3.1.2 Orthogonalisierung

Ziel der zweiten Phase des TSM-Ansatzes ist es, die Winkel an jedem Knoten des Graphen festzulegen, sodass die Anzahl der Knicke minimal ist. Die Form einer orthogonalen Zeichnung wird durch die zugehörige *orthogonale Repräsentation* beschrieben:

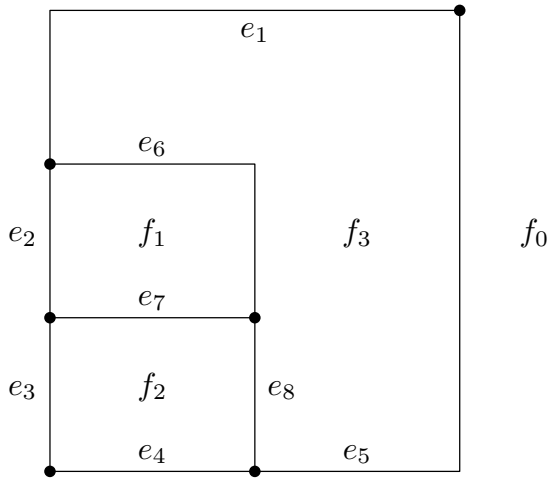
Definition 3.1 (Orthogonale Repräsentation)
Sei G ein 4-planarer Graph mit planarer Einbettung Π. Eine ***orthogonale Repräsentation*** *H von G ist eine Menge zirkulär geordneter Listen $H(f)$ für jede Fläche f von G. Jedes Element $r \in H(f)$ hat die Form (e_r, s_r, a_r), wobei*

- *e_r eine f begrenzende Kante ist,*
- *s_r ein Binärstring ist,*
- *$a_r \in \{90°, 180°, 270°, 360°\}$.*

Der Binärstring s_r beschreibt die Form der Kante e_r. Dabei steht das k-te Bit für den k-ten Knick, den e_r in der Fläche f macht. Für innere Flächen geht man im Uhrzeigersinn an der Kante entlang, für die Außenfläche entgegen des Uhrzeigersinns. Dabei steht eine Null für einen 90°-Knick und eine Eins für einen 270°-Knick. Der

[1] Im Folgenden meist kurz „TSM-Ansatz“.

leere String ϵ gibt an, dass die Kante keine Knicke macht. a_r ist der Winkel, den e_r mit der in der Flächenuhrzeigerliste von f nachfolgenden Kante bildet. Abbildung 3.1 zeigt eine orthogonale Zeichnung mit zugehöriger orthogonaler Repräsentation (vgl. [Tamassia 1987], [Duncan u. Goodrich 2013], [Spisla 2012, Kap. 3.2]).



f_0 : $\langle(e_1, 1, 180°), (e_2, \epsilon, 180°), (e_3, \epsilon, 270°), (e_4, \epsilon, 180°), (e_5, 1, 270°)\rangle$
f_1 : $\langle(e_2, \epsilon, 90°), (e_6, 0, 90°), (e_7, \epsilon, 90°)\rangle$
f_2 : $\langle(e_3, \epsilon, 90°), (e_7, \epsilon, 90°), (e_8, \epsilon, 90°), (e_4, \epsilon, 90°)\rangle$
f_3 : $\langle(e_1, 0, 90°), (e_5, 0, 90°), (e_8, \epsilon, 180°), (e_6, 1, 90°)\rangle$

Abb. 3.1: Orthogonale Zeichnung mit orthogonaler Repräsentation

Die Elemente $r \in H(f)$ können für 4-planare Graphen nicht beliebig gewählt werden. Die folgenden vier Bedingungen sind notwendig und hinreichend dafür, dass H eine orthogonale Repräsentation eines 4-planaren Graphen ist:

(B1) H ohne die Binärstrings und Winkel ist die planare Einbettung eines 4-planaren Graphen.

(B2) Seien r und r' zwei Elemente aus H, die mit derselben Kante korrespondieren. Dann erhält man $s_{r'}$ aus s_r, indem man den Binärstring bitweise negiert und die Reihenfolge der Bits umdreht.

(B3) Seien $|s|_0$ und $|s|_1$ die Anzahl der Nullen bzw. Einsen im String s. Dann gilt für jede Fläche f:

$$\sum_{r \in H(f)} |s_r|_0 - |s_r|_1 + (2 - \frac{a_r}{90°}) = \begin{cases} 4, & \text{falls } f \text{ eine innere Fläche ist} \\ -4, & \text{falls } f \text{ die äußere Fläche ist} \end{cases}$$

(B4) Für jeden Knoten v gilt:

$$\sum_{e_r=(u,v)} a_r = 360°$$

Bedingung (B3) stellt sicher, dass jede Fläche ein rektilineares Polygon ist; (B4) gewährleistet, dass sich die Winkel um jeden Knoten zu 360° aufaddieren (vgl. [Spisla 2012, Kap. 3.2]).

Das Problem, eine planare, orthogonale Zeichnung eines Graphen mit minimaler Anzahl von Knicken zu finden, ist *NP*-schwer (vgl. [Garg u. Tamassia 2002]). Wählt man allerdings eine feste planare Einbettung, so lässt sich mit Hilfe von Netzwerkflüssen eine orthogonale Zeichnung mit minimaler Anzahl Knicke berechnen. Diese Technik wurde erstmals in [Tamassia 1987] vorgestellt. Das Problem, die Anzahl der Knicke zu minimieren, wird dabei auf ein Minimum-Kosten-Fluss-Problem übertragen und kann in einer Laufzeit von $\mathcal{O}((n+b)^{3/2})$ gelöst werden, wobei n die Anzahl der Knoten im Ursprungsgraph und b die Anzahl der Knicke ist. Die Maximalzahl von Knicken pro Kante wird üblicherweise beschränkt. Daher kann $b \in \mathcal{O}(n)$ angenommen werden (vgl. [Cornelsen u. Karrenbauer 2012], [Tamassia u. Tollis 1989]).

Mit Abschluss der Orthogonalisierungsphase steht also die Form der Zeichnung fest, nicht jedoch die Länge der Kanten.

3.1.3 Kompaktierung

Die Länge der Kantensegmente wird in der dritten Phase des TSM-Ansatzes, dem Kompaktierungsschritt, festgelegt. Dabei werden allen Knoten Koordinaten zugewiesen, sodass Knoten nicht mit anderen Knoten oder Kanten kollidieren. Je nach Anwendungsproblem werden bei der Kompaktierung unterschiedliche Ziele verfolgt: So soll die Gesamtkantenlänge, die Länge der längsten Kante oder die Zeichenfläche minimiert werden. Alle drei Probleme sind *NP*-schwer (vgl. [Patrignani 2001]).

Das Ziel in dieser Arbeit ist es, die Gesamtkantenlänge zu minimieren. Dabei wird angenommen, dass die orthogonale Repräsentation fest vorgegeben ist. Die Winkel, die zwei Kanten an einem gemeinsamen Knoten miteinander bilden, können also nicht mehr verändert werden. Außerdem wird vorausgesetzt, dass die orthogonale Repräsentation zuvor *normalisiert* wurde, d. h. alle Knicke durch künstliche Knoten ersetzt wurden; im Anschluss an die Kompaktierungsphase wird dieser Schritt wieder rückgängig gemacht.

O. B. d. A. kann angenommen werden, dass der Graph nur aus einer Zusammenhangskomponente besteht. Sollte ein Graph aus mehreren Zusammenhangskomponenten bestehen, könnten diese einzeln kompaktiert werden, ohne dass sich die Gesamtkantenlänge dadurch ändert.

Der Graph kann unter diesen Voraussetzungen mithilfe verschiedener Algorithmen kompaktiert werden: Im folgenden Kapitel werden zunächst zwei eindimensionale Heuristiken vorgestellt, bevor in Abschnitt 3.4.1 eine zweidimensionale Methode erläutert wird.

In Kapitel 4 wird schließlich ein eigener Ansatz zur Kompaktierung präsentiert. Dabei wird zusätzlich vorausgesetzt, dass der Graph 2-zusammenhängend ist.

3.2 Eindimensionale Kompaktierungsheuristiken

Da das Kompaktierungsproblem *NP*-schwer ist und somit kein polynomieller Lösungsalgorithmus bekannt ist, werden häufig eindimensionale Heuristiken angewendet, die das Problem näherungsweise lösen. Das Kompaktierungsproblem wird dabei in zwei eindimensionale Subprobleme – ein horizontales und ein vertikales – zerlegt. Jedes dieser Subprobleme wird unabhängig vom anderen Subproblem gelöst, d. h. die orthogonale Zeichnung wird nur in eine Richtung verändert, während sie in die andere Richtung fixiert ist. Die Lösungen eindimensionaler Heuristiken sind im Allgemeinen nicht optimal, können jedoch in polynomieller Zeit bestimmt werden. Kompaktierungsheuristiken lassen sich in zwei Klassen einteilen: *Konstruktive Heuristiken* erzeugen ausgehend von der orthogonalen Repräsentation eine zulässige Zeichnung des Graphen. *Verbesserungsheuristiken* schließen an konstruktive Heuristiken an, indem sie bereits bestehende Zeichnungen weiter kompaktieren (vgl. [Eiglsperger et al. 2001, Kap. 6], [Spisla 2012, Kap. 3.4]).

3.2.1 Eine konstruktive Heuristik

Tamassia hat 1987 eine erste konstruktive Heuristik vorgestellt, die das eindimensionale Kompaktierungsproblem in polynomieller Laufzeit löst. Voraussetzung ist, dass alle Flächen rechteckig sind. Dies wird durch Anwendung der rechteckigen Dissecting-Methode erreicht. Anschließend werden im eigentlichen Kompaktierungsschritt die Längen der Kanten festgelegt (vgl. [Tamassia 1987], [Jünger u. Mutzel 2003, S. 40]).

Die rechteckige Dissecting-Methode

Sei H eine orthogonale Repräsentation des Graphen G. Um alle Flächen von H rechteckig zu machen, wird jeder inneren Fläche f ein zirkulärer Bitstring $S(f)$ zugewiesen:

$$S(f) = w(a_{r_0})w(a_{r_1}) \cdots w(a_{r_k}),$$

wobei

$$H(f) = r_0 r_1 \cdots r_k \qquad \text{und} \qquad w(a_{r_i}) = \begin{cases} 0, & \text{falls } a_{r_i} = 1, \\ \epsilon, & \text{falls } a_{r_i} = 2, \\ 1, & \text{falls } a_{r_i} = 3, \\ 11, & \text{falls } a_{r_i} = 4. \end{cases}$$

In $S(f)$ wird der Substring „100“ gesucht und durch „0“ ersetzt. An der entsprechenden Stelle in der Fläche wird ein neuer Knoten, ein sogenannter *Dissecting-Knoten*, eingefügt. Der Dissecting-Knoten wird durch eine neue Kante mit dem Knoten verbunden, dem vorher der Wert 1 zugeordnet war. Geometrisch bedeutet das, dass ein rechteckiges „Ohr“ ([Eiglsperger et al. 2001]) aus der Fläche herausgeschnitten wird, so wie in Abbildung 3.2 dargestellt. Dies wiederholt man solange, bis $S(f) = 0000$ gilt, d.h. bis f ein Rechteck ist (vgl. [Tamassia 1987], [Eiglsperger et al. 2001, Kap. 6], [Spisla 2012, Kap. 3.4]).

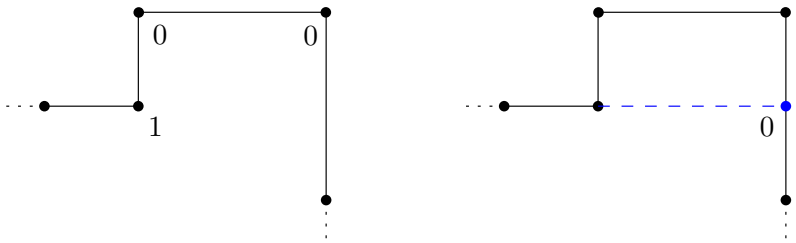


Abb. 3.2: Rechteckige Dissecting-Methode

Für die äußere Fläche wird genauso vorgegangen. Die äußere Fläche wird dadurch in eine Menge aus mehreren Rechtecken und das Komplement eines weiteren Rechtecks zerlegt (vgl. [Tamassia 1987], [Klau 2001, Lemma 4.5]).

Die rechteckige Dissecting-Methode hat eine Laufzeit von $\mathcal{O}(|V|)$ (vgl. [Tamassia 1987]).

Kompaktierung mithilfe von Flussnetzwerken

Die so veränderte orthogonale Repräsentation H' soll durch Lösen zweier Minimum-Kosten-Fluss-Probleme in eine orthogonale Zeichnung überführt werden. Dazu werden zwei Netzwerke N_h, N_v – ein horizontales und ein vertikales – erzeugt, diese sind in Abbildung 3.3 dargestellt. N_h und N_v sind Subgraphen des dualen Graphen von H': N_h enthält nur vertikale Kanten, d. h. für jede horizontale Kante e in H' existiert eine duale Kante a in N_h, die die beiden Knoten verbindet, die zu den durch e getrennten Flächen in H' gehören. Dabei sind alle Kanten in N_h von unten nach oben gerichtet. N_v enthält entsprechend nur horizontale Kanten, die dual zu vertikalen Kanten in H' sind und alle von links nach rechts gerichtet sind.

Die Zuweisung der Kantenlängen erfolgt durch Lösen zweier Minimum-Kosten-Fluss-Probleme auf N_h und N_v. Das MKFP wird im Folgenden an $N_h = (U_h, A_h, b, l, u, c)$ erläutert.

Der Fluss auf jeder Kante $a \in A_h$ gibt die Länge der zugehörigen primalen Kante e in G an. Um sicherzustellen, dass e mindestens Länge 1 hat, wird die untere Schranke l_a von a auf 1 gesetzt. Die obere Schranke u_a auf allen Kanten ist ∞, der Bedarf b_u aller

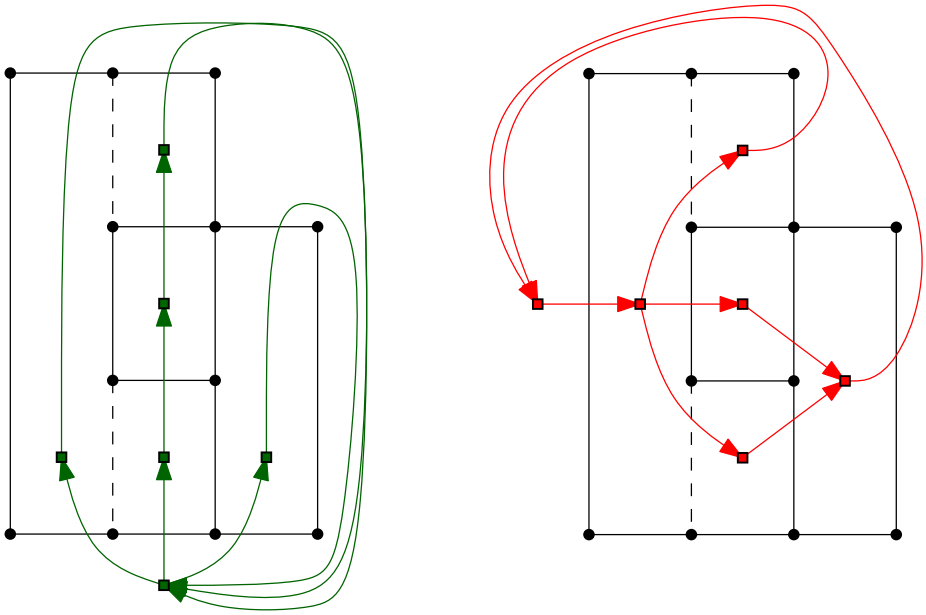


Abb. 3.3: Horizontales und vertikales Netzwerk

Knoten $u \in U_h$ ist 0. Durch die Flusserhaltungsbedingung ist gewährleistet, dass zwei gegenüberliegende Seiten desselben Rechtecks in G insgesamt dieselbe Länge haben.

Allen Kanten a, deren primale Kante e eine Dissecting-Kante ist, werden keine Kosten zugewiesen, da Dissecting-Kanten später wieder aus H' entfernt werden und bei der Berechnung der Gesamtkantenlänge nicht einfließen. Alle anderen Kanten a erhalten Einheitskosten. Die Kostenfunktion c lautet also:

$$c_a = \begin{cases} 0, & \text{falls die zu } a \text{ primale Kante eine Dissecting-Kante ist} \\ 1 & \text{sonst} \end{cases}$$

Durch Berechnung eines Minimum-Kosten-Flusses auf dem so definierten Netzwerk N_h wird allen horizontalen Kanten in H' eine zulässige Kantenlänge zugewiesen (vgl. [Tamassia 1987]).

In die andere Richtung geht man analog vor: Durch Lösen eines Minimum-Kosten-Fluss-Problems auf dem Netzwerk N_v wird allen vertikalen Kanten in H' eine zulässige Länge zugewiesen. Addiert man die Zielfunktionswerte der beiden Minimum-Kosten-Fluss-Probleme, erhält man die Gesamtkantenlänge. Diese konstruktive Methode hat eine Laufzeit von $\mathcal{O}(|V|^{3/2})$ (vgl. [Cornelsen u. Karrenbauer 2012]).

Die Lösung der konstruktiven Heuristik ist im Allgemeinen allerdings nicht optimal. Die Anordnung der Dissecting-Kanten hat großen Einfluss auf die Gesamtkantenlänge. Eine ungünstige Entscheidung beim Einfügen einer Dissecting-Kante kann zu einer höheren Gesamtkantenlänge führen. Auf dieses Problem wird in Abschnitt 3.3.2 näher

eingegangen. Die konstruktive Heuristik liefert nur dann eine optimale Lösung, wenn eine orthogonale Repräsentation ausschließlich rechteckige Flächen besitzt, also keine Dissecting-Kanten mehr eingefügt werden müssen (vgl. [Klau u. Mutzel 1999, S. 15f.], [Klau 2001, S. 54]).

3.2.2 Eine Verbesserungsheuristik

Verbesserungsheuristiken schließen an konstruktive Heuristiken an, indem sie versuchen, bestehende orthogonale Zeichnungen weiter zu kompaktieren. Sei Γ eine orthogonale Zeichnung von G. Dann bezeichne mit S_h bzw. S_v die Menge der horizontalen bzw. vertikalen Segmente in Γ. Die Idee der vorgestellten Verbesserungsheuristik besteht darin, Sichtbarkeitseigenschaften für alle Segmente festzulegen. Anschließend werden die Segmente so verschoben, dass sich die Gesamtkantenlänge verringert, die Sichtbarkeitseigenschaften aber erhalten bleiben. Das horizontale und das vertikale Subproblem werden dabei wiederum einzeln betrachtet.

Ein horizontales Segment *sieht* ein anderes horizontales Segment, wenn man die beiden Segmente durch eine vertikale Linie verbinden kann, ohne ein anderes horizontales Segment zu kreuzen. Sichtbarkeit zwischen vertikalen Segmenten ist analog definiert (vgl. [Eiglsperger et al. 2001, Kap. 6], [Spisla 2012, S. 18f.]).

Auf Basis dieser Sichtbarkeitseigenschaften lassen sich zwei sogenannte *Layout-Graphen* $D_x = (S_v, A_x)$, $D_y = (S_h, A_y)$ definieren, wobei die Kantenmengen gegeben sind durch:

$$\begin{aligned} A_x &:= \{(s_i, s_j) \mid s_i \text{ sieht } s_j \text{ rechts von sich}\} \\ A_y &:= \{(s_i, s_j) \mid s_i \text{ sieht } s_j \text{ oberhalb von sich}\} \end{aligned}$$

D_x und D_y sind per Konstruktion azyklisch. Abbildung 3.4 zeigt zwei Layout-Graphen D_x, D_y.

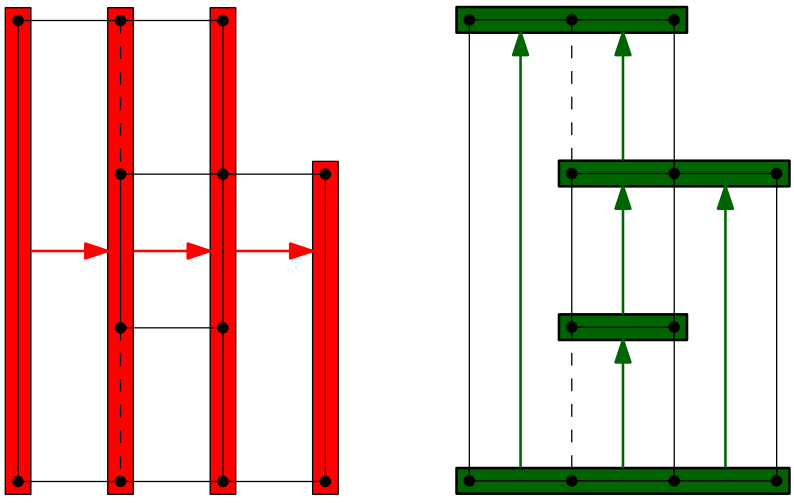


Abb. 3.4: Layout-Graphen

Jede Kante $a = (s_i, s_j)$ stellt eine Distanzbedingung an die beiden Segmente dar, die durch die Knoten s_i und s_j repräsentiert werden: Die Koordinaten von s_i und s_j müssen sich um mindestens eine Einheit unterscheiden. Das erreicht man, indem man eine topologische Nummerierung der Knoten von D_x berechnet, wobei die horizontale Koordinate der Knoten in vertikalen Segmenten gerade die topologische Nummer des Segments ist. Dies führt zu einer horizontalen Kompaktierung mit minimaler möglicher Breite. In vertikale Richtung geht man analog vor. Es wird abwechselnd in die eine und in die andere Richtung kompaktiert, bis keine Verbesserung mehr möglich ist (vgl. [Eiglsperger et al. 2001, Kap. 6], [Spisla 2012, S. 18f.]).

Dieses Vorgehen führt dazu, dass Knoten möglichst weit links bzw. unten angeordnet werden, da jedem Segment eine minimale topologische Nummerierung zugewiesen wird. Wichtig ist die Reihenfolge der Kompaktierungsschritte: Jede Längenzuweisung stellt eine eindimensionale Entscheidung dar, durch die die Zeichnung in eine Richtung verändert wird. In die andere Richtung ist die Zeichnung währenddessen fixiert. Ein Kompaktierungsschritt in eine Richtung verhindert daher unter Umständen eine anschließende Kompaktierung in die andere Richtung. Die Lösung ist im Allgemeinen nicht optimal (vgl. [Eiglsperger et al. 2001, Kap. 6], [Spisla 2012, S. 18f.]).

3.3 Schwächen eindimensionaler Heuristiken

Besteht eine orthogonale Repräsentation ausschließlich aus rechteckigen Flächen, so liefert die konstruktive Heuristik von Tamassia eine Zeichnung mit minimaler Gesamtkantenlänge. Eine Anwendung der Verbesserungsheuristik ist dann gar nicht mehr notwendig. (vgl. [Klau u. Mutzel 1999, S. 15f.], [Klau 2001, S. 54]). Für das allgemeine zweidimensionale Kompaktierungsproblem liefern die beiden Heuristiken hingegen keine Optimallösung. Ursachen dafür sind die fehlende Parallelität und lokale Entscheidungen, die beim Einfügen der Dissecting-Kanten getroffen werden. Dies wird im Folgenden an mehreren Beispielen erläutert.

3.3.1 Künstliche Kanten

Durch Dissecting-Kanten wird insbesondere festgelegt, welche relative Position zwei Segmente zueinander haben und welche Segmente sich gegenseitig sehen. Die Festlegung dieser Sichtbarkeitseigenschaften ist notwendig, damit die konstruktive Heuristik und die Verbesserungsheuristik zulässige Zeichnungen erzeugen. Die Layout-Graphen der Verbesserungsheuristik basieren gerade auf diesen Sichtbarkeitseigenschaften.

Sichtbarkeit ist jedoch eine Eigenschaft einer konkreten Zeichnung, nicht der Instanz im Allgemeinen. Indem man Dissecting-Kanten einfügt und somit die Sichtbarkeitseigenschaften festlegt, trifft man – lokal – bereits eine Entscheidung zugunsten einer bestimmten Zeichnung. Dies ist eine starke Einschränkung der Geometrie (vgl. [Klau 2001, Kap. 4.2.6]).

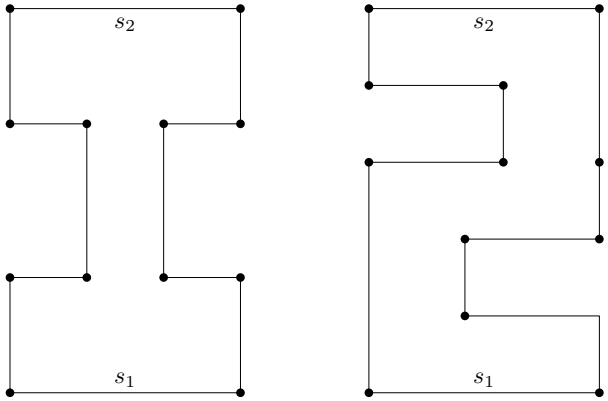


Abb. 3.5: Unterschiedliche Sichtbarkeitseigenschaften

Abbildung 3.5 zeigt zwei Zeichnungen, denen dieselbe orthogonale Repräsentation zugrunde liegt. In der linken Zeichnung sehen sich die Segmente s_1 und s_2, in der rechten nicht. Indem man die rechteckige Dissecting-Methode auf diese orthogonale Repräsentation anwendet, schließt man bereits eine der beiden möglichen Zeichnungen aus.

Selbst in „relativ einfachen“ Flächen, wie der Fläche in Abbildung 3.6, existieren bereits verschiedene Möglichkeiten, Dissecting-Kanten einzufügen.

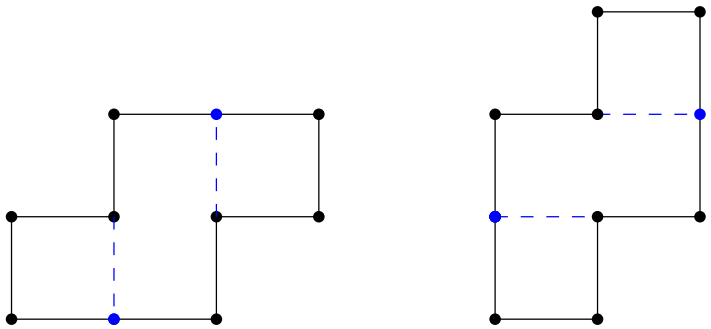


Abb. 3.6: Unterschiedliche Dissecting-Kanten

Abbildung 3.7 macht deutlich, dass sich solche Entscheidungen stark auf die Gesamtkantenlänge auswirken können. Die beiden dargestellten Zeichnungen basieren auf demselben Graphen und derselben orthogonalen Repräsentation. Beide wurden mit der konstruktiven Heuristik mit anschließender Verbesserungsheuristik erzeugt, lediglich die Dissecting-Kanten wurden an anderen Stellen eingefügt bzw. anders ausgerichtet. Während die Gesamtkantenlänge in der linken Zeichnung 128 beträgt, liegt sie in der rechten Zeichnung bei 108.

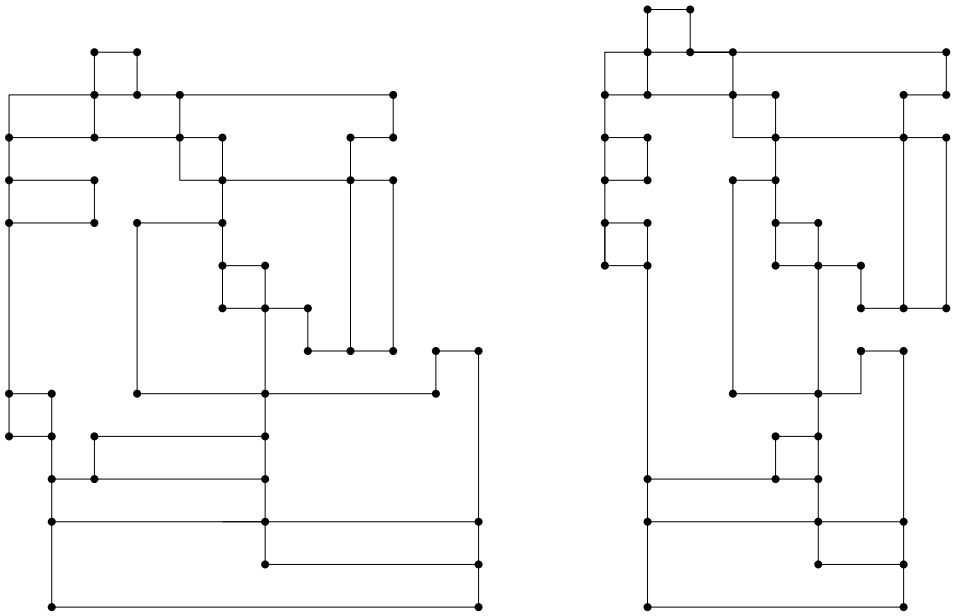


Abb. 3.7: Auswirkungen auf die Gesamtkantenlänge

3.3.2 Fehlende Parallelität

Eine weitere Schwäche eindimensionaler Heuristiken ist die fehlende Parallelität. Wendet man die Verbesserungsheuristik auf das horizontale oder vertikale Subproblem an, ist die Zeichnung währenddessen in die andere Richtung fixiert. Ein Kompaktierungsschritt in eine Richtung verhindert daher unter Umständen eine spätere Kompaktierung in die andere Richtung.

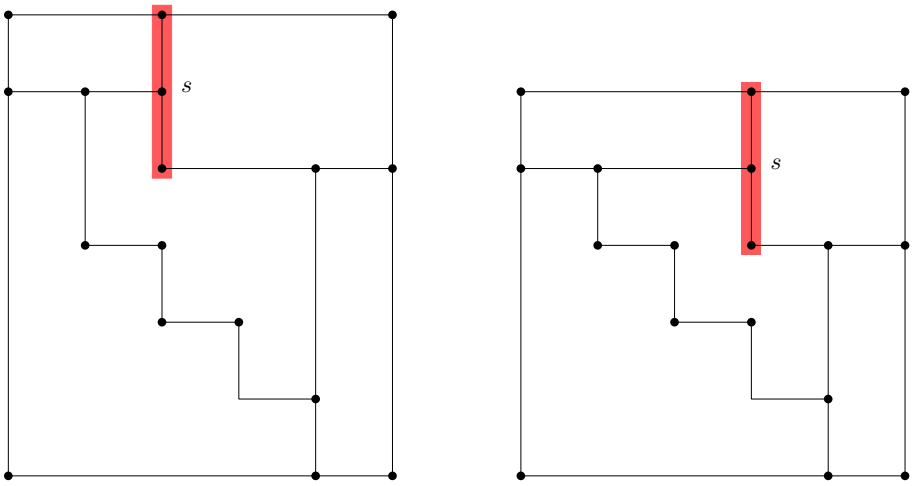


Abb. 3.8: Folgen fehlender Parallelität

Abbildung 3.8 zeigt, dass die entstehende Zeichnung – je nach Reihenfolge der Kompaktierungsschritte – nicht optimal ist. In der linken Zeichnung wurde zuerst in horizontale Richtung kompaktiert, wodurch dem vertikalen Segment s eine minimale topologische Nummer zugewiesen wurde. Dies verhindert jedoch eine weitere Kompaktierung in vertikale Richtung. In diesem Beispiel wäre es günstiger, zuerst vertikal zu kompaktieren, dann entsteht die rechte Zeichnung.

3.4 Lösungsansätze

Das Kompaktierungsproblem wurde in den letzten Jahren ausgiebig erforscht, da es eine wichtige Rolle beim VLSI-Design spielt. Es existieren verschiedene Ansätze, um die Probleme, die eindimensionale Methoden mit sich bringen, zu umgehen und bessere Ergebnisse zu erzielen. In diesem Abschnitt werden zwei solcher Ansätze vorgestellt, die zweidimensionale Methode von Klau und Mutzel sowie der Turn-Regularity-Ansatz.

Klau und Mutzel haben eine neue kombinatorische Formulierung des Kompaktierungsproblems entwickelt. Dies ermöglicht es, das Problem als ganzzahliges lineares Programm zu formulieren. Dieses IP kann mithilfe von Branch-and-Cut-Verfahren optimal gelöst werden. Für spezielle Graphenklassen kann in polynomieller Zeit eine Optimallösung gefunden werden (vgl. [Klau u. Mutzel 1999], [Klau 2001, Satz 4.5], [Jünger u. Mutzel 2003, S. 40f.], [Spisla 2012, S. 19ff.]).

Der Turn-Regularity-Ansatz setzt nicht bei der Kompaktierung, sondern beim Dissecting-Schritt an. Da die Ausrichtung der Dissecting-Kanten starken Einfluss auf die Gesamtkantenlänge hat, wurde in [Bridgeman et al. 2000] mit dem Turn-Regularity-Ansatz eine neue Dissecting-Methode entwickelt, bei der weniger künstliche Kanten eingefügt werden müssen als bei der rechteckigen Dissecting-Methode (vgl. [Klau 2001]).

Bei der Entwicklung einer eigenen Kompaktierungsheuristik, die in Kapitel 4 vorgestellt wird, sind wesentliche Ideen sowohl der Methode von Klau und Mutzel als auch des Turn-Regularity-Ansatzes eingeflossen. Die beiden Methoden werden daher in den folgenden Unterabschnitten näher erläutert.

3.4.1 Die zweidimensionale Methode von Klau und Mutzel

Bei der Methode von Klau und Mutzel handelt es sich – im Gegensatz zu den bisher vorgestellten Heuristiken – um ein zweidimensionales Kompaktierungsverfahren. Das Verfahren wurde erstmals in [Klau u. Mutzel 1999] vorgestellt. Die Definitionen in diesem Unterabschnitt stammen, wenn nicht anders vermerkt, aus dieser Quelle. Die Darstellung orientiert sich an [Spisla 2012, Kap. 3.4.3].

Der Ansatz von Klau und Mutzel besteht darin, das Kompaktierungsproblem in ein kombinatorisches Problem zu überführen. Dies geschieht mittels sogenannter *Formbeschreibungen*, die jeweils aus zwei *Constraint-Graphen* bestehen. Diese Constraint-Graphen haben Ähnlichkeit mit den oben beschriebenen Layout-Graphen: Für jedes Segment in G existiert wiederum ein Knoten in dem horizontalen Constraint-Graphen D_h oder dem vertikalen Constraint-Graphen D_v. Die Kanten zwischen diesen Knoten werden jedoch nicht auf Basis von Sichtbarkeitseigenschaften eingefügt; stattdessen wird immer dann eine Kante in D_h bzw. D_v eingefügt, wenn zwei gleich ausgerichtete Segmente durch eine Kante in G verbunden sind.

Definition 3.2 (Formbeschreibung)
*Eine **Formbeschreibung** einer einfachen orthogonalen Repräsentation H ist ein Tupel $\sigma = \langle D_h, D_v \rangle$ von zwei gerichteten Constraint-Graphen $D_h = (S_v, A_h)$ und $D_v = (S_h, A_v)$ mit*

$$\begin{aligned} A_h &:= \{(l(e), r(e)) \mid e \text{ ist horizontale Kante}\} \quad \text{und} \\ A_v &:= \{(b(e), t(e)) \mid e \text{ ist vertikale Kante}\}. \end{aligned}$$

Dabei bezeichnet $l(e)$ das vertikale Segment, in dem der linke Knoten der horizontalen Kante e liegt. $r(e)$, $b(e)$ und $t(e)$ bezeichnen analog das Segment, in dem der rechte, untere bzw. obere Knoten von e liegt.

Es wird wiederum vorausgesetzt, dass die orthogonale Repräsentation zuvor normalisiert wurde. Anders als bei den eindimensionalen Heuristiken ist es jedoch nicht mehr notwendig, dass alle Flächen rechteckig sind.

Die Kanten in $A_h \cup A_v$ bestimmen die relative Position zweier Segmente. Diese Informationen reichen jedoch noch nicht aus, um eine zulässige orthogonale Zeichnung zu erzeugen. Für je zwei Knoten u und v muss ein gerichteter Weg $u \xrightarrow{*} v$ existieren. Dazu muss die Formbeschreibung gegebenenfalls erweitert werden.

Definition 3.3 (Vollständige Erweiterung)
*Eine **vollständige Erweiterung** einer Formbeschreibung $\sigma = \langle (S_v, A_h), (S_h, A_v) \rangle$ ist ein Tupel $\tau = \langle (S_v, B_h), (S_h, B_v) \rangle$ mit folgenden Eigenschaften:*

(E1) $A_h \subseteq B_h$ *und* $A_v \subseteq B_v$.

(E2) B_h *und* B_v *sind azyklisch.*

(E3) Für jedes Paar von Segmenten $(s_i, s_j) \in S \times S$ mit $S := S_h \cup S_v$ gilt eine der folgenden Bedingungen:

1. $r(s_i) \xrightarrow{*} l(s_j)$
2. $r(s_j) \xrightarrow{*} l(s_i)$
3. $t(s_j) \xrightarrow{*} b(s_i)$
4. $t(s_i) \xrightarrow{*} b(s_j)$

Dabei ist $l(s_i)$, ähnlich wie $l(e)$, das vertikale Segment, zu dem der linkeste Knoten von s_i gehört; $r(s_i)$, $b(s_i)$ und $t(s_i)$ sind analog definiert.

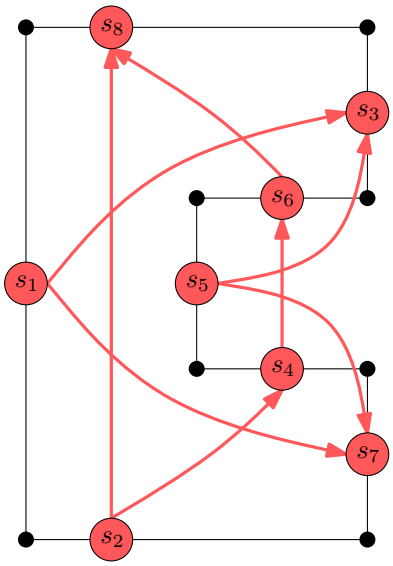


Abb. 3.9: Formbeschreibung

Abbildung 3.9 zeigt eine Formbeschreibung (rot). Die Formbeschreibung ist nicht vollständig, da kein gerichteter Weg von Segmentknoten s_1 zu Segmentknoten s_5 existiert. Durch die Kante (s_1, s_5) könnte die Formbeschreibung vollständig erweitert werden.

Aus jeder vollständig erweiterten Formbeschreibung kann eine orthogonale Zeichnung konstruiert werden. Die Aufgabe bei der Kompaktierung besteht also darin, eine vollständige Erweiterung mit minimaler Gesamtkantenlänge zu finden. Dieses Problem lässt sich als ganzzahliges lineares Programm formulieren:

$$\begin{aligned} &\min \quad \sum_{e \in E_h} c(r(e)) - c(l(e)) + \sum_{e \in E_v} c(t(e)) - c(b(e)) \\ &\text{sodass} \end{aligned} \tag{3.1}$$

$$\begin{aligned} x_{ij} &= 1 && \forall (s_i, s_j) \in A_h \cup A_v && (1) \\ x_{r(i)l(j)} + x_{r(j)l(i)} + x_{t(i)b(j)} + x_{t(j)b(i)} &\geq 1 && \forall (s_i, s_j) \in S \times S && (2) \\ c(s_i) - c(s_j) + (M+1)x_{ij} &\leq M && \forall (s_i, s_j) \in C && (3) \\ x_{ij} &\in \{0, 1\} && \forall (s_i, s_j) \in C && (4) \end{aligned}$$

C ist definiert durch $C := (S_h \times S_h) \cup (S_v \times S_v)$. Für eine vollständige Erweiterung $\tau = \langle (S_v, B_h), (S_h, B_v) \rangle$ setze

$$x_{ij} = \begin{cases} 1, & \text{falls } (s_i, s_j) \in B_h \cup B_v, \\ 0 & \text{sonst.} \end{cases}$$

Die Bestimmung der Kantenlängen erfolgt durch *Längenzuordnungen* $c : S_h \cup S_v \to \mathbb{N}$ mit der Eigenschaft $c(s_i) < c(s_j)$ für alle $(s_i, s_j) \in B_h \cup B_v$. Diese Bedingung sorgt dafür, dass die Längen der Segmente konsistent sind und für jeden Knoten u aus H

$$x_u = c(\text{vert}(u)) \quad \text{und} \quad y_u = c(\text{hor}(u))$$

gilt, wobei $\text{vert}(u)$ bzw. $\text{hor}(u)$ das vertikale bzw. horizontale Segment ist, zu dem u gehört. M sei eine hinreichend große Zahl.[2]

Die Zielfunktion des IP (3.1) gibt die Gesamtkantenlänge einer Zeichnung von G an. Durch Bedingung (1) wird gewährleistet, dass Eigenschaft ($E1$) erfüllt ist. Nebenbedingung (2) repräsentiert Eigenschaft ($E3$). Durch Bedingung (3) wird garantiert, dass c eine Längenzuordnung ist.

M kann als Maximum von Höhe und Breite der Zeichnung gewählt werden. Falls $x_{ij} = 1$ ist, entspricht Gleichung (3) der Bedingung $c(s_j) - c(s_i) \geq 1$, d. h. das Segment s_j wird rechts bzw. oberhalb des Segmentes s_i gezeichnet. Für $x_{ij} = 0$ lautet die Bedingung $c(s_i) - c(s_j) \leq M$ und stellt keine Einschränkung der zulässigen Lösungen dar, da dies für alle $(s_i, s_j) \in C$ erfüllt ist.

Dieses IP kann durch Branch-and-Cut-Verfahren gelöst werden und liefert eine Optimallösung des zweidimensionalen Kompaktierungsproblems. Ist die Formbeschreibung bereits eine vollständige Erweiterung oder gibt es nur eine eindeutige Möglichkeit zur Erweiterung, so kann das Problem in polynomieller Zeit optimal gelöst werden (vgl. [Klau 2001, Satz 4.5]).

Künstliche Kanten stellen also auch bei der Methode von Klau und Mutzel ein Problem dar. Sobald unterschiedliche Möglichkeiten bestehen, künstliche Kanten – hier in Form von Erweiterungskanten, nicht von Dissecting-Kanten – einzufügen, ist die polynomielle Laufzeit nicht mehr gewährleistet.

Das Problem der fehlenden Parallelität hingegen wurde gelöst. Durch die Formulierung als IP kann es nicht mehr passieren, dass ein Kompaktierungsschritt in eine Richtung eine günstigere Kompaktierung in die andere Richtung verhindert. Diese Idee – das Kompaktierungsproblem durch ein zweidimensionales, IP-basiertes Verfahren zu lösen – wurde bei der Entwicklung eines eigenen Ansatzes übernommen.

3.4.2 Der Turn-Regularity-Ansatz

Der Turn-Regularity-Ansatz ist eine Dissecting-Methode, die im Jahr 2000 von Bridgeman et al. entwickelt wurde. Die Definitionen, Lemmata und Sätze in diesem Unterabschnitt stammen, wenn nicht anders vermerkt, aus [Bridgeman et al. 2000].

Die Idee des Turn-Regularity-Ansatzes besteht darin, Paare von Knoten zu bestimmen, die bei der Zuweisung der Kantenlängen kollidieren können. Nur zwischen solchen

[2] Im Operations Research ist dieser Ansatz als „Big-M-Ansatz“ bekannt.

Knoten wird eine Dissecting-Kante eingefügt. Es ist nicht mehr nötig, alle Flächen des Graphen rechteckig zu machen. Dadurch lässt sich die Anzahl der Dissecting-Kanten reduzieren.

G sei im Folgenden ein 4-planarer Graph, H sei eine normalisierte orthogonale Repräsentation von G und Γ sei eine planare Zeichnung von H. Für einen Knoten v aus G bezeichne mit $x(v)$ bzw. $y(v)$ die x- bzw. y-Koordinate von v in Γ.

Sei (u, v) ein beliebiges Knotenpaar aus G. Da beim Turn-Regularity-Ansatz Paare kollidierender Knoten bestimmt werden sollen, spielen die relativen Positionen zweier Knoten zueinander eine wichtige Rolle. Definiere dazu die folgenden vier *binären Relationen*:

$$u <_x v, \quad \text{falls } x(u) < x(v)$$
$$u =_x v, \quad \text{falls } x(u) = x(v)$$
$$u <_y v, \quad \text{falls } y(u) < y(v)$$
$$u =_y v, \quad \text{falls } y(u) = y(v)$$

Kombiniert man eine Relation in x-Richtung mit einer Relation in y-Richtung, ergeben sich drei weitere binäre Relationen:

$$=_x \wedge <_y$$
$$<_x \wedge =_y$$
$$<_x \wedge <_y$$

Gemeinsam bilden diese sieben binären Relationen die *orthogonalen Relationen*. Sie stellen die Grundlage des Turn-Regularity-Konzeptes dar. Es besteht genau dann eine orthogonale Relation für jedes Knotenpaar aus G, wenn die orthogonale Repräsentation H turn-regulär ist. Aufgabe im Dissecting-Schritt ist es also, alle nicht-turn-regulären Flächen zu bestimmen und diese durch Dissecting-Kanten turn-regulär zu machen.

Dazu werden allen Winkeln in einer Fläche Werte zugeordnet, ähnlich wie bei der rechteckigen Dissecting-Methode. Die Werte unterscheiden sich allerdings von den Werten der rechteckigen Methode:

Sei f eine Fläche von G. Für jeden Winkel c in f setze

$$\text{turn}(c) := \begin{cases} 1, & \text{falls } c \text{ ein } 90°\text{-Winkel ist,} \\ 0, & \text{falls } c \text{ ein } 180°\text{-Winkel ist,} \\ -1, & \text{falls } c \text{ ein } 270°\text{-Winkel ist.} \end{cases}$$

360°-Winkel können nicht auftreten, da für den Turn-Regularity-Ansatz vorausgesetzt wird, dass der Graph 2-zusammenhängend ist. Eine orthogonale Repräsentation eines 2-zusammenhängenden Graphen kann keine 360°-Winkel enthalten.

Darauf aufbauend lässt sich die Rotation zweier Winkel definieren:

Definition 3.4 (Rotation)
Sei (v_i, v_j) ein geordnetes Paar von Knoten auf dem Rand einer Fläche f, seien c_i, c_j die zu v_i, v_j gehörigen Winkel in f. Dann ist die ***Rotation*** *des geordneten Paares (c_i, c_j) definiert durch*

$$rot(c_i, c_j) := \sum_{c \in P} turn(c),$$

wobei P ein Pfad von c_i (eingeschlossen) zu c_j (nicht eingeschlossen) entlang der Kanten von f ist.

In [Bridgeman et al. 2000] wurden folgende Eigenschaften der Rotation gezeigt:

Lemma 3.5 (Eigenschaften der Rotation)
Sei f eine Fläche von G. Dann gilt:

(i)

$$rot(c_i, c_i) = \begin{cases} 4, & \text{falls } f \text{ eine innere Fläche ist,} \\ -4, & \text{falls } f \text{ die äußere Fläche ist.} \end{cases}$$

(ii)

$$rot(c_i, c_j) = 2 \quad \Leftrightarrow \quad rot(c_j, c_i) = \begin{cases} 2, & \text{falls } f \text{ eine innere Fläche ist,} \\ -6, & \text{falls } f \text{ die äußere Fläche ist.} \end{cases}$$

(iii) Für jedes geordnete Tripel von Winkeln (c_i, c_j, c_k) auf dem Rand von f gilt:

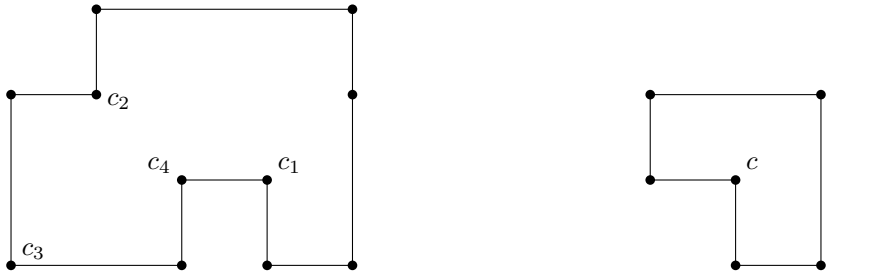
$$rot(c_i, c_k) = rot(c_i, c_j) + rot(c_j, c_k)$$

In dem Beispiel in Abbildung 3.10(a) gilt $\mathrm{rot}(c_1, c_2) = 1$, $\mathrm{rot}(c_2, c_3) = 2$ sowie $\mathrm{rot}(c_2, c_4) = 2$.

Definition 3.6 (Kitty Corners)
Zwei 270°-Winkel c_i, c_j auf dem Rand derselben Fläche f heißen ***Kitty Corners****, wenn*

$$rot(c_i, c_j) = 2 \quad oder \quad rot(c_j, c_i) = 2.$$

In Abbildung 3.10(a) stellt (c_2, c_4) ein Paar von Kitty Corners dar.



(a) Rotation verschiedener Winkelpaare **(b)** Turn-reguläre Fläche

Abb. 3.10: Turn-Regularity-Ansatz

Definition 3.7 (Turn-reguläre Fläche)
*Eine Fläche f heißt **turn-regulär**, wenn sie keine Kitty Corners besitzt.*

Abbildung 3.10(b) zeigt eine turn-reguläre Fläche. Der Winkel c ist der einzige 270°-Winkel und damit der einzige Kandidat für einen Kitty Corner. Es existiert jedoch kein anderer Winkel, der mit c ein Paar von Kitty Corners bilden könnte. Während bei der rechteckigen Methode eine Dissecting-Kante eingefügt werden würde, ist dies beim Turn-Regularity-Ansatz nicht notwendig.

Definition 3.8 (Turn-reguläre orthogonale Repräsentation)
*Eine orthogonale Repräsentation heißt **turn-regulär**, wenn all ihre Flächen turn-regulär sind.*

Die 270°-Winkel in einer Fläche f lassen sich in vier Klassen unterteilen: Nordost-Winkel, Nordwest-Winkel, Südost-Winkel und Südwest-Winkel. Man spricht von einem Nordost-Winkel, wenn der zugehörige Knoten nur in nördlicher und östlicher Richtung eine inzidente Kante besitzt, d. h. der Knoten ist linker und unterer Endknoten zweier Kanten auf dem Rand von f. Nordwest-, Südost- und Südwest-Winkel sind analog definiert.

Lemma 3.9
Sei (c_1, c_2) ein Paar von Kitty Corners. Dann handelt es sich bei c_1 und c_2 entweder um einen Südwest- und einen Nordost-Winkel oder um einen Südost- und einen Nordwest-Winkel.

Mithilfe der obigen Lemmata kann der folgende Satz hergeleitet werden, der den Zusammenhang zwischen den eingangs erwähnten orthogonalen Relationen und der Turn-Regularität herstellt:

Satz 3.10
Eine orthogonale Repräsentation H eines eingebetteten, 4-planaren Graphen ist turn-regulär genau dann, wenn für jedes Knotenpaar aus G eine orthogonale Relation besteht.

Der detaillierte Beweis von Satz 3.10 findet sich in [Bridgeman et al. 2000, S. 82f.].

Mithilfe des Turn-Regularity-Ansatzes können also, zusammenfassend, Paare von Kitty Corners bestimmt werden, die bei der Zuweisung der Kantenlängen kollidieren könnten. Zwischen solchen Kitty Corners wird eine Dissecting-Kante eingefügt. Diese Idee wurde bei der Entwicklung des eigenen Ansatzes übernommen.

In [Bridgeman et al. 2000] wird die Richtung der Dissecting-Kante schon beim Einfügen fest gewählt – nach dem Zufallsprinzip horizontal oder vertikal. Beim eigenen Ansatz wird die Richtung hingegen offengelassen, um lokale Entscheidungen, die sich ungünstig auf die Gesamtkantenlänge auswirken können, zu vermeiden.

4 Eigener Ansatz zur Kompaktierung

In diesem Kapitel wird ein eigener Ansatz zur Kompaktierung vorgestellt. Die Schwachpunkte eindimensionaler Heuristiken sollen dabei umgangen werden: Um die Parallelität des horizontalen und des vertikalen Kompaktierungsschrittes zu gewährleisten, wird das Problem als IP formuliert, ähnlich wie im zweidimensionalen Ansatz von Klau und Mutzel. Um die Anzahl der Dissecting-Kanten gering zu halten, wird statt der rechteckigen Dissecting-Methode der Turn-Regularity-Ansatz verwendet. Um ungünstige lokale Entscheidungen zu vermeiden, die unter Umständen zu höherer Gesamtkantenlänge führen können, wird die Richtung der Dissecting-Kanten zunächst nicht festgelegt. Ob eine Dissecting-Kante horizontal oder vertikal verläuft, wird erst beim Lösen des IP entschieden.

In Abschnitt 4.1 wird dieser eigene Ansatz zur Kompaktierung vorgestellt. Anschließend wird auf die Korrektheit und Laufzeit der einzelnen Schritte eingegangen. Die Lösung des eigenen Ansatzes weicht aufgrund einiger Vereinfachungen von der Optimallösung des allgemeinen Kompaktierungsproblems ab. Diese Einschränkungen werden in Abschnitt 4.3 erläutert. Die Nebenbedingungsmatrix des entstehenden IP ist unter gewissen Voraussetzungen total unimodular, wie in Abschnitt 4.4 gezeigt wird.

4.1 Vorgehensweise

Sei $G = (V, E)$ ein 4-planarer, 2-zusammenhängender Graph und sei H eine normalisierte orthogonale Repräsentation von G. Dann erzeuge dieselben beiden Netzwerke $N_h = (U_h, A_h, \cdot, \cdot, \cdot, \cdot)$ und $N_v = (U_v, A_v, \cdot, \cdot, \cdot, \cdot)$ wie bei der konstruktiven Heuristik, allerdings ohne zuvor die rechteckige Dissecting-Methode anzuwenden. Für jede Kante $a \in A_h \cup A_v$ bezeichne mit $\text{prim}(a)$ die zu a primale Kante aus E. Für alle Nicht-Dissecting-Kanten $e \in E$ sei $\text{dual}(e)$ die zugehörige duale Kante in $A \setminus A_D$.

Die beiden Minimum-Kosten-Fluss-Probleme, die bei der konstruktiven Heuristik auf N_h und N_v gelöst werden, lassen sich als gemeinsames IP formulieren:

$$\begin{aligned} \min \quad & \mathbf{1}^T x \\ \text{sodass} \quad & f_u(x) = 0 && \forall\ u \in U \\ & x_a \geq 1 && \forall\ a \in A \\ & x_a \in \mathbb{Z} && \forall\ a \in A \end{aligned} \tag{4.1}$$

wobei $U := U_h \cup U_v$ und $A := A_h \cup A_v$ ist.

In dieser Form besteht das IP noch aus zwei voneinander unabhängigen, eindimensionalen Minimum-Kosten-Fluss-Problemen. Erst durch zusätzliche Nebenbedingungen, die beim Dissecting eingefügt werden, entsteht eine Verbindung zwischen den beiden eindimensionalen Problemen.

4.1.1 Dissecting

IP (4.1) liefert im Allgemeinen noch keine zulässigen Zeichnungen, da Knoten miteinander kollidieren oder Kanten sich schneiden könnten. Um dies zu verhindern, werden im Folgenden zusätzliche Dissecting-Kanten und Nebenbedingungen eingefügt.

Kollidierende Knoten

Bestimme mithilfe des Turn-Regularity-Ansatzes alle Kitty Corners in H. Sei f eine beliebige Fläche in G, in der mindestens ein Paar von Kitty Corners existiert. Dann füge eine Dissecting-Kante zwischen dem ersten Paar von Kitty Corners in f ein. f wird dadurch in zwei Flächen f' und f'' unterteilt. Gehe für diese beiden Teilflächen rekursiv vor, d. h. suche in f' und f'' nach einem weiteren Paar von Kitty Corners und füge zwischen diesem Paar eine Dissecting-Kante ein.

Insoweit entspricht die Vorgehensweise dem Turn-Regularity-Ansatz. Im Unterschied zum Turn-Regularity-Ansatz soll aber die Ausrichtung von Dissecting-Kanten noch nicht festgelegt werden, sondern bis zur Kompaktierung offen gelassen werden.

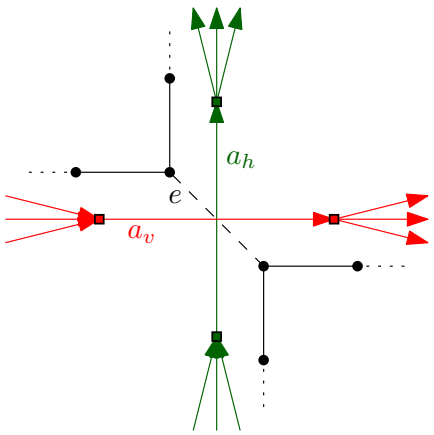


Abb. 4.1: Dissecting-Kante ohne vorgegebene Richtung

Dadurch besitzt eine Dissecting-Kante sowohl im horizontalen als auch im vertikalen Netzwerk eine duale Kante. In Abbildung 4.1 stellt e eine Dissecting-Kante dar, a_h ist die duale Kante in N_h, a_v die duale Kante in N_v. Bezeichne solche zusammengehörigen Kanten a_h, a_v als *Partnerkanten*.

Definition 4.1 (Partnerkanten)
*Zwei Kanten $(a_h, a_v) \in A_h \times A_v$ heißen **Partnerkanten**, wenn $prim(a_h) = prim(a_v)$.*

Aus $\text{prim}(a_h) = \text{prim}(a_v)$ folgt zwangsläufig, dass $\text{prim}(a_h)$ eine Dissecting-Kante ist, denn für alle Nicht-Dissecting-Kanten ist die duale Kante eindeutig bestimmt.

Um zu verhindern, dass die Endknoten von e kollidieren, muss auf einer der beiden Partnerkanten a_h, a_v ein Fluss von mindestens 1 existieren. Der Fluss auf der anderen Kante kann beliebig gewählt werden – auch negativ. Auf Kanten, die dual zu Dissecting-Kanten sind, existiert also keine untere Schranke. Formal ausgedrückt:

$$\max(x_{a_h}, 0) + \max(x_{a_v}, 0) \geq 1$$

Diese Ungleichung kann in dieser Form allerdings nicht in das IP übernommen werden, da die Lösungsmenge dieser Ungleichung keine konvexe Menge in der a_h-a_v-Ebene darstellt. Das entstehende Optimierungsproblem wäre nicht mehr linear.

Füge stattdessen die vereinfachte Ungleichung

$$x_{a_h} + x_{a_v} \geq 1$$

zu den Nebenbedingungen hinzu. Auf diese Vereinfachung wird in Abschnitt 4.3 näher eingegangen.

Wähle die Kostenfunktion $c : A \to \mathbb{N}$ wie folgt:

$$c_a = \begin{cases} 0, & \text{falls die zu } a \text{ primale Kante eine Dissecting-Kante ist,} \\ 1 & \text{sonst.} \end{cases}$$

Es lässt sich zeigen, dass Kitty Corners nur in Flächen mit mindestens acht Kanten auftreten:

Lemma 4.2
Sei (c_1, c_2) ein Paar von Kitty Corners in einer Fläche f. Dann wird f von mindestens acht Kanten begrenzt.

Beweis:
In jeder inneren Fläche f gilt:

$$\begin{aligned}
& \sum_{c \text{ in } f} \text{turn}(c) && = 4 \\
\Rightarrow \quad & \sum_{\substack{c \text{ in } f,\\ c_1 \neq c \neq c_2}} \text{turn}(c) + \text{turn}(c_1) + \text{turn}(c_2) && = 4 \\
\overset{\text{turn}(c_1)=\text{turn}(c_2)=-1}{\Rightarrow} \quad & \underbrace{\sum_{\substack{c \text{ in } f,\\ c_1 \neq c \neq c_2}} \text{turn}(c)}_{=6} - 2 && = 4 \\
\overset{\text{turn}(c)\in\{-1,0,1\}}{\Rightarrow} \quad & f \text{ wird von mindestens acht Kanten begrenzt.}
\end{aligned}$$

Der Beweis für die externe Fläche funktioniert analog. ■

Der Algorithmus zum Einfügen von Dissecting-Kanten kann wie folgt zusammengefasst werden:

```
Dissecting()
    for all faces f do
        if |E_f| ≥ 8 then
            List kc = DetermineKittyCorners(f);
            InsertDissectingEdges(kc);
        end
    end
end

InsertDissectingEdges(List kc)
    if !kc.empty() then
        Tuple (c1, c2) = kc.pop();
        InsertPartnerEdges(c1, c2);
        List remainingKC;
        while !kc.empty() do
            (c1, c2) = kc.pop();
            if StillKittyCorners(c1, c2) then
                remainingKC.push((c1, c2));
            end
        end
        InsertDissectingEdges(remainingKC);
    end
end
```

Kollidierende Kanten

Durch die eingefügten Dissecting-Kanten ist gewährleistet, dass Knoten nicht kollidieren. Es kann jedoch passieren, dass Kanten miteinander kollidieren. Abbildung 4.2 zeigt ein solches Beispiel.

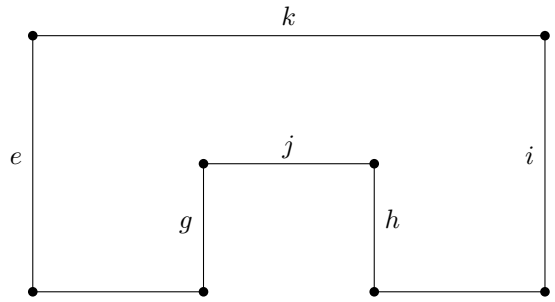


Abb. 4.2: Kollidierende Kanten

Die horizontale Flusserhaltungsbedingung für die Fläche in Abbildung 4.2 lautet:

$$x_{\text{dual}(e)} - x_{\text{dual}(g)} + x_{\text{dual}(h)} - x_{\text{dual}(i)} = 0$$

Es wäre also zulässig, $x_{\text{dual}(e)}$, $x_{\text{dual}(g)}$, $x_{\text{dual}(h)}$ und $x_{\text{dual}(i)}$ gleich 1 zu wählen. Dann würden aber die Kanten j und k kollidieren. Dieses Problem kann immer dann auftreten, wenn eine Fläche einen 270°-Winkel enthält.

Definition 4.3 (Reflexive Kante)
*Eine Kante $e = (u, v) \in E$ heißt **reflexive Kante**, wenn sie durch einen 270°-Winkel begrenzt wird, d. h. falls*

$$\mathit{turn}(u) = -1 \quad \vee \quad \mathit{turn}(v) = -1.$$

Definition 4.4 (Gegenüberliegende Kante)
*Sei e eine reflexive Kante, die durch einen 270°-Winkel c in einer Fläche f begrenzt wird. Sei $P_{e,c}$ ein Pfad entlang des Randes von f, beginnend bei e in Richtung c. Dann ist die **gegenüberliegende Kante** $\mathit{opp}(e)$ die erste Nicht-Dissecting-Kante auf dem Pfad $P_{e,c}$ mit $\mathit{rot}(c, c') = 2$, wobei c' der Winkel von $\mathit{opp}(e)$ mit ihrer Nachfolgerkante ist.*

Dabei ist „in Richtung c“ so zu verstehen, dass die Kante, die mit e den Winkel c bildet, die Nachfolgerkante von e ist.

Die Kanten in einer Fläche f lassen sich in Nord-, Ost-, Süd- und West-Kanten einteilen, je nachdem, in welche Himmelsrichtung sie f begrenzen. Wegen $\text{rot}(c, c') = 2$ ist $\text{opp}(e)$ immer die erste Nicht-Dissecting-Kante auf dem Pfad $P_{e,c}$, die die entgegengesetzte Himmelsrichtung zu e besitzt.

Definition 4.5 (Innen- aus Außenkanten)
*Sei $P_{e,c}$ ein Pfad von e nach $\text{opp}(e)$ in einer Fläche f. Sei g die letzte Nicht-Dissecting-Kante vor $\text{opp}(e)$. Eine Kante h auf dem Pfad $P_{e,c}$ heißt **Innenkante**, wenn sie die entgegengesetzte Himmelsrichtung besitzt wie g. h heißt **Außenkante**, wenn sie dieselbe Himmelsrichtung besitzt wie g.*

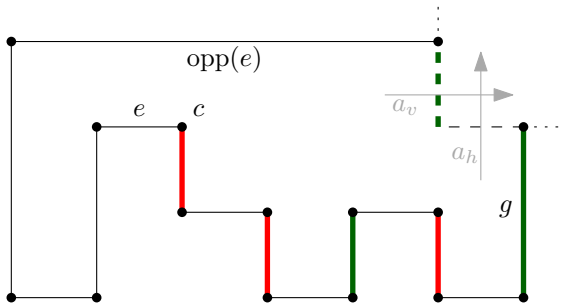


Abb. 4.3: Innen- und Außenkanten

Abbildung 4.3 zeigt die Innen- und Außenkanten auf einem Pfad $P_{e,c}$. Die Kante g ist eine Ost-Kante. Alle West-Kanten auf dem Pfad $P_{e,c}$ sind also Innenkanten (rot), alle Ost-Kanten auf $P_{e,c}$ sind Außenkanten (grün). Die Dissecting-Kante zwischen g und $\text{opp}(e)$ wurde in Abbildung 4.3 mit Knick gezeichnet, um ihren horizontalen „Anteil“ a_h und ihren vertikalen „Anteil“ a_v zu verdeutlichen. Bei einem positiven Fluss auf a_v ist die Dissecting-Kante eine Ost-Kante und wird daher ebenfalls als Außenkante aufgefasst. Gehe in solchen Zweifelsfällen, immer davon aus, dass eine Dissecting-Kante, deren Richtung noch nicht feststeht, ebenfalls eine Innen- bzw. Außenkante ist.

In dem Beispiel in Abbildung 4.2 muss sichergestellt werden, dass

$$x_{\text{dual}(i)} \geq x_{\text{dual}(h)} + 1$$

gilt. Allgemein lässt sich die Bedingung formulieren als

$$-\sum_{\substack{\text{prim}(a) \in P_{e,c},\\ \text{prim}(a) \text{ Innenkante}}} x_a \quad + \sum_{\substack{\text{prim}(a) \in P_{e,c},\\ \text{prim}(a) \text{ Außenkante}}} x_a \quad \geq 1$$

wobei $P_{e,c}$ ein Pfad von e nach $\text{opp}(e)$ ist. Füge dem IP für alle reflexiven Kanten und alle 270°-Winkel diese Nebenbedingungen hinzu. Ein Sonderfall tritt auf, wenn die Vorgängerkante von $\text{opp}(e)$ eine Dissecting-Kante ist. Füge dem IP dann zusätzlich zu der obigen Nebenbedingung die Bedingung

$$-\sum_{\substack{\text{prim}(a) \in P'_{e,c},\\ \text{prim}(a) \text{ Innenkante}}} x_a \quad + \sum_{\substack{\text{prim}(a) \in P'_{e,c},\\ \text{prim}(a) \text{ Außenkante}}} x_a \quad \geq 1$$

hinzu. Dabei ist P' ein Teilpfad von P, der nur bis zur letzten Nicht-Dissecting-Kante vor $\text{opp}(e)$ führt.

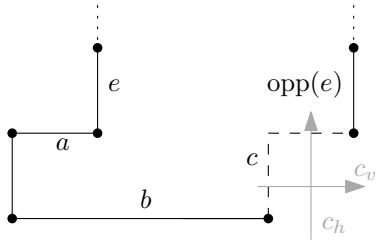


Abb. 4.4: Sonderfall: Dissecting-Kante als Vorgängerkante von opp(e)

In dem Beispiel in Abbildung 4.4 werden dem IP die beiden Nebenbedingungen

$$\begin{aligned} -x_{\text{dual}(a)} + x_{\text{dual}(b)} + x_{c_v} &\geq 1 \\ -x_{\text{dual}(a)} + x_{\text{dual}(b)} &\geq 1 \end{aligned}$$

hinzugefügt. Die Bedingung $-x_{\text{dual}(a)} + x_{\text{dual}(b)} + x_{c_v} \geq 1$ ist notwendig, wenn x_{c_v} negativ gewählt wird; im Falle von $x_{\text{dual}(b)} \leq x_{\text{dual}(a)} + |x_{c_v}|$ würden ansonsten e und opp(e) miteinander kollidieren. Die Bedingung $-x_{\text{dual}(a)} + x_{\text{dual}(b)} \geq 1$ ist nötig, wenn $x_{c_v} = 0$ ist.

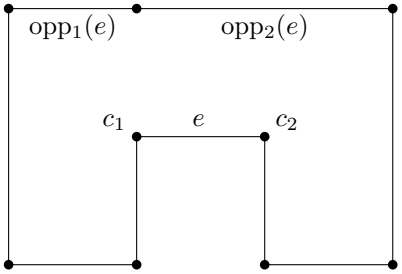


Abb. 4.5: Zwei verschiedene gegenüberliegende Kanten

Beim Einfügen der Nebenbedingungen ist außerdem ist Folgendes zu beachten:

1. Für eine reflexive Kante e können zwei Nebenbedingungen existieren, falls e von zwei 270°-Winkeln c_1, c_2 begrenzt wird. e kann dann zwei unterschiedliche gegenüberliegende Kanten $\text{opp}_1(e)$, $\text{opp}_2(e)$ besitzen, wie in Abbildung 4.5 dargestellt. Zwischen diesen beiden gegenüberliegenden Kanten können aber nur 180°-Winkel auftreten. Denn würde zwischen $\text{opp}_1(e)$ und $\text{opp}_2(e)$ ein 90°- oder 270°-Winkel auftreten, dann würde dieser Winkel mit c_1 oder c_2 ein Paar von Kitty Corners bilden und es wäre zuvor eine Dissecting-Kante eingefügt worden. Zwischen $\text{opp}_1(e)$ und $\text{opp}_2(e)$ kann auch keine Dissecting-Kante auftreten. $\text{opp}_1(e)$ und $\text{opp}_2(e)$ gehören also immer zu demselben Segment.

2. In einer Fläche können mehrere Nebenbedingungen für verschiedene reflexive Kanten auftreten. Es ist möglich, dass ein Pfad $P_{e',c'}$ ein Teilpfad eines anderen Pfades $P_{e,c}$ ist. In Abbildung 4.6 tritt dieser Fall auf. Für solche Kanten e und e' gilt immer $\text{opp}(e) = \text{opp}(e')$. Sobald nämlich auf P die erste Kante mit Rotation 2 auftritt, muss diese auch auf P' Rotation 2 haben, da $\text{rot}(c, c') = 0$ gilt.

3. Es ist möglich, dass eine reflexive Kante von einer Dissecting-Kante begrenzt wird. Füge auch für solche reflexiven Kanten Nebenbedingungen ein, selbst wenn noch nicht feststeht, wie die Dissecting-Kante angeordnet wird und welchen Winkel die reflexive Kante mit der Dissecting-Kante bildet. Gehe im Zweifelsfall immer davon aus, dass eine reflexive Kante einen 270°-Winkel mit einer Dissecting-Kante bildet. In Abbildung 4.4 tritt dieser Fall auf: Wird x_{c_h} negativ gewählt, dann bildet b einen 270°-Winkel mit c. b ist daher als reflexive Kante zu behandeln. Wird x_{c_h} positiv gewählt, so bilden b und c einen 90°-Winkel. Die Nebenbedingung, die für den Weg von b nach $\text{opp}(b)$ eingefügt wurde, ist dann redundant, stellt also keine Einschränkung dar.

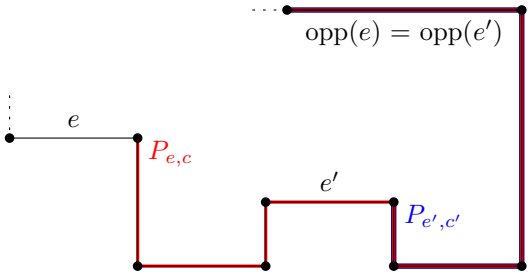


Abb. 4.6: $P_{e',c'}$ als Teilpfad eines anderen Pfades $P_{e,c}$

4.1.2 Kompaktierung

Durch das Einfügen der Dissecting-Kanten und der Nebenbedingungen ist abschließend folgendes IP entstanden:

$$\min \quad c^T x \tag{$\star$}$$

sodass

$$\begin{array}{rrcll}
 & f_u(x) & = & 0 & \forall\, u \in U \\
- \sum_{\substack{\text{prim}(a) \in P_{e,c},\\ \text{prim}(a)\ \text{Innenkante}}} x_a & + \sum_{\substack{\text{prim}(a) \in P_{e,c},\\ \text{prim}(a)\ \text{Außenkante}}} x_a & \geq & 1 & \forall\ \text{Pfade } P_{e,c} \\
 & x_{a_h} + x_{a_v} & \geq & 1 & \forall\ \text{Partnerkanten } (a_h, a_v) \in A_h \times A_v \\
 & x_a & \geq & 1 & \forall\, a \in A \setminus A_D \\
 & x_a & \in & \mathbb{Z} & \forall\, a \in A
\end{array}$$

wobei $A_D := \{a \in A \mid \text{prim}(a) \text{ ist Dissecting-Kante}\} \subset A$.

Aufgabe im Kompaktierungsschritt ist es nun, dieses IP zu lösen. Eine Lösung x gibt, wie bei der konstruktiven Heuristik, wiederum die Kantenlängen der primalen Kanten an. Unter einer Voraussetzung, die in Abschnitt 4.4 formuliert wird, ist die Nebenbedingungsmatrix total unimodular. Löse in diesem Fall die LP-Relaxierung von ($\star$), ansonsten wende ein Branch-and-Bound-Verfahren auf ($\star$) an.

4.1.3 Postprozessor-Schritt

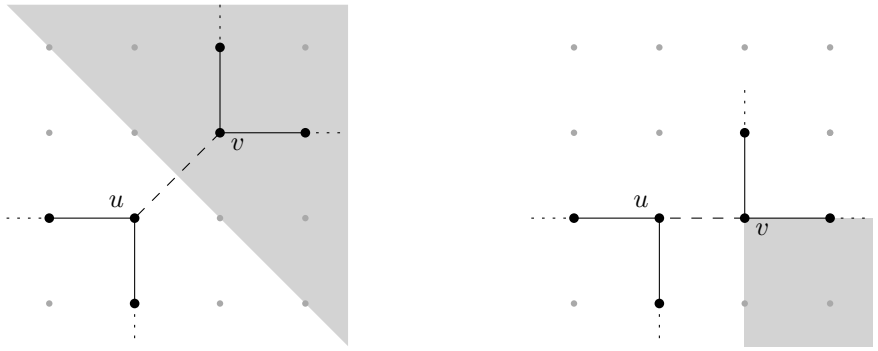
Im Rahmen des Dissecting-Schrittes wurden dem IP die Nebenbedingungen

$$x_{a_h} + x_{a_v} \geq 1$$

statt der Bedingungen

$$\max(x_{a_h}, 0) + \max(x_{a_v}, 0) \geq 1$$

hinzugefügt. Dies stellt eine Einschränkung dar. In Abbildung 4.7(a) kann der Knoten v dadurch nur in der grau markierten Fläche gewählt werden. (x_{a_h}, x_{a_v}) kann beispielsweise nicht $(1, -1)$ gewählt werden, d. h. v kann nicht auf dem Gitterpunkt unten rechts angeordnet werden.



(a) Wahl von x_{a_h} und x_{a_v} im Kompaktierungsschritt

(b) Wahl von x_{a_h} und x_{a_v} im Postprozessor-Schritt

Abb. 4.7: Einschränkungen bei der Wahl von x_{a_h} und x_{a_v}

Wegen dieser Einschränkung im Kompaktierungsschritt wird durch einen anschließenden Postprozessor-Schritt versucht, die Zeichnung weiter zu verbessern. Wird (x_{a_h}, x_{a_v}) im Rahmen der Kompaktierung beispielsweise $(1, 0)$ gewählt, ist es offenbar günstig, die Dissecting-Kante (u, v) horizontal anzuordnen. Da x_{a_h} positiv ist, kann die Bedingung $x_{a_h} + x_{a_v} \geq 1$ fallen gelassen werden, x_{a_v} kann dann beliebig klein gewählt werden.

Im Postprozessor-Schritt wird also wie folgt vorgegangen: Setze die untere Schranke auf der positiven Partnerkante auf 1, setze die obere Schranke auf der anderen Partnerkante auf deren aktuellen Fluss, entferne die zugehörige Bedingung $x_{a_h} + x_{a_v} \geq 1$ und löse das IP erneut.

```
Postprocessing()
    for all edges a ∈ A do
        if PrimalEdgeIsDissectingEdge(a) then
            edge b = PartnerEdge(a);
            if x_a < x_b && x_b ≥ 1 then
                SetBounds(x_a, −∞, x_a);
                SetBounds(x_b, 1, ∞);
                RemoveConstraint(x_a, x_b);
            end
        end
    end
    ResolveLP();
end
```

Angenommen, in dem Beispiel aus Abbildung 4.7(a) wurde $(x_{a_h}, x_{a_v}) = (1, 0)$ gewählt. Dann kann Knoten v nun beliebig in der grau markierten Fläche in Abbildung 4.7(b) platziert werden. Im Postprozessor-Schritt ist es also möglich, $(x_{a_h}, x_{a_v}) = (1, -1)$ zu wählen.

Der Postprozessor-Schritt kann zu unzulässigen Zeichnungen führen. Verwerfe in diesem Fall die Postprozessor-Lösung und stelle die zulässige IP-Lösung wieder her. Die experimentelle Analyse in Kapitel 5 wird zeigen, dass dieser Fall jedoch nur selten auftritt; auf 99,81% der Testinstanzen lieferte der Postprozessor-Schritt zulässige Zeichnungen.

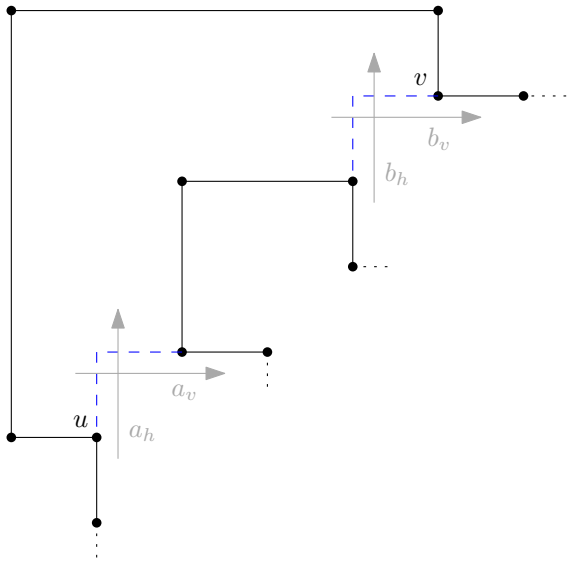


Abb. 4.8: Postprozessor-Schritt kann zu unzulässiger Zeichnung führen

Eine Instanz, die unzulässig gezeichnet werden könnte, ist in Abbildung 4.8 dargestellt. Die Nebenbedingungen

$$\begin{aligned} x_{a_h} + x_{a_v} &\geq 1 \\ x_{b_h} + x_{b_v} &\geq 1 \end{aligned}$$

im IP gewährleisten, dass u und v nicht kollidieren. Denn dazu müssten $x_{a_v} < 0$ und $x_{b_h} < 0$ genügend klein gewählt werden, die Knoten u und v würden sich dann in die obere linke Ecke schieben. Wenn aber $x_{a_v} < 0$ und $x_{b_h} < 0$ sehr klein sind, müssen im Gegenzug x_{a_h} und x_{b_v} ausreichend groß gewählt werden, um die obigen Nebenbedingungen zu erfüllen. Dadurch ist ein Mindestabstand zwischen u und v gewährleistet.

Im Postprozessor-Schritt werden die obigen Nebenbedingungen fallen gelassen. Dann können $x_{a_v} < 0$ und $x_{b_h} < 0$ beliebig klein gewählt werden, ohne dass im Gegenzug x_{a_h} und x_{b_v} ausreichend groß gewählt werden müssen. u und v können daher kollidieren.

4.2 Korrektheit und Laufzeit

In diesem Abschnitt wird kurz auf die Korrektheit und Laufzeit des Dissecting-, Kompaktierungs- und Postprozessor-Schrittes eingegangen.

Das Einfügen der Dissecting-Kanten verläuft im eigenen Ansatz im Wesentlichen wie im Turn-Regularity-Ansatz. Zur Korrektheit und Laufzeit des Turn-Regularity-Ansatzes wurden in [Bridgeman et al. 2000] folgende Lemmata bewiesen:

Lemma 4.6
Sei H eine turn-reguläre orthogonale Repräsentation eines 4-planaren, 2-zusammenhängenden Graphen und sei Γ eine orthogonale Zeichnung von H derart, dass für jedes Knotenpaar (u, v) in H eine orthogonale Relation zwischen u und v besteht. Dann ist Γ planar.

Lemma 4.7
Sei G ein 4-planarer, 2-zusammenhängender Graph mit n Knoten und Knicken und sei H eine orthogonale Repräsentation von G. Die Kitty Corners in H können in einer Laufzeit von $\mathcal{O}(n)$ bestimmt werden.

Der Unterschied zum eigenen Ansatz liegt nur in der Ausrichtung der Dissecting-Kanten. Da diese im eigenen Ansatz zunächst offen gelassen wird, gilt die Aussage von Lemma 4.6 nur unter der zusätzlichen Voraussetzung, dass die Längen aller Kanten bereits festgelegt wurden. Unmittelbar nach dem Dissecting-Schritt besitzen Knoten noch keine eindeutigen relativen Positionen zueinander. In Abbildung 4.9 kann beispielsweise sowohl $u <_x v$ als auch $v <_x u$ gelten, je nachdem, ob der horizontale

Anteil der Dissecting-Kante e positiv oder negativ gewählt wird. Die relativen Positionen stehen im eigenen Ansatz daher erst nach Abschluss des Kompaktierungsschrittes fest. Berücksichtigt man diesen Unterschied in den Voraussetzungen von Lemma 4.6, dann können für den eigenen Ansatz dieselben Aussagen getroffen werden wie für den Turn-Regularity-Ansatz.

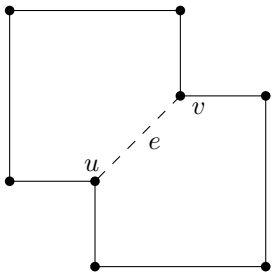


Abb. 4.9: Uneindeutige relative Positionen

Auch die Nebenbedingungen gegen kollidierende Kanten basieren auf einer Idee aus [Bridgeman et al. 2000]: In dieser Arbeit wurden zwei Hilfsgraphen H_x, H_y mit sogenannten saturierenden Kanten erzeugt. Eine saturierende Kante verbindet dabei jeweils den 270°-Knoten einer reflexiven Kante mit dem 90°-Knoten der gegenüberliegenden Kante, wie in Abbildung 4.10 dargestellt.

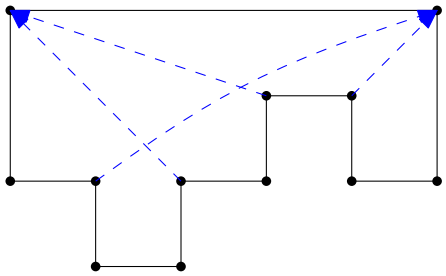


Abb. 4.10: Saturierende Kanten

Es wurde gezeigt, dass die saturierenden Kanten genau dann eindeutig bestimmt sind, wenn die orthogonale Repräsentation turn-regulär ist.

Lemma 4.8
Sei H eine turn-reguläre orthogonale Repräsentation eines 4-planaren, 2-zusammenhängenden Graphen. H_x und H_y sind genau dann eindeutig bestimmt, wenn H turn-regulär ist.

Jede saturierende Kante in H_x oder H_y entspricht einer Nebenbedingung im eigenen Ansatz. Ein Unterschied ergibt sich jedoch daraus, dass im eigenen Ansatz die Ausrichtung der Dissecting-Kanten offen gelassen wird und auf den zugehörigen dualen Kanten ein negativer Fluss existieren kann. Dadurch sind im eigenen Ansatz noch weitere Nebenbedingungen notwendig, wie Abbildung 4.11 zeigt.

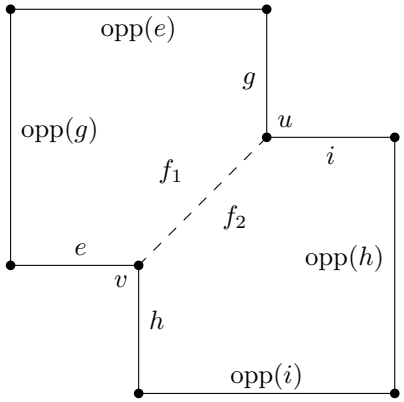


Abb. 4.11: Zusätzlich benötigte Nebenbedingungen

Wendet man den Turn-Regularity-Ansatz auf die abgebildete orthogonale Repräsentation an, werden u und v durch eine Dissecting-Kante verbunden. In f_1 und f_2 existiert anschließend keine reflexive Kante mehr. Es muss also keine saturierende Kante eingefügt werden. Wendet man hingegen den eigenen Ansatz an so entstehen in Fläche f_1 zwei Bedingungen für die Pfade von e bzw. g nach opp(e) bzw. opp(g). Die Kanten e und g werden als reflexive Kanten aufgefasst, da die Dissecting-Kante (u, v) einen 270°-Winkel mit e bzw. g bilden könnte. In f_2 entstehen zwei Nebenbedingungen für die Pfade von h bzw. i nach opp(h) bzw. opp(i). In der Optimallösung sind später maximal zwei dieser vier Nebenbedingungen bindend, abhängig davon, ob die Dissecting-Kante (u, v) einen positiven oder negativen horizontalen bzw. vertikalen Anteil besitzt.

Die Korrektheit der Nebenbedingungen gegen kollidierende Kanten folgt aus Lemma 4.8. Da jedoch mehr Nebenbedingungen eingefügt werden müssen, muss gezeigt werden, dass die Laufzeitschranke von $\mathcal{O}(n)$ trotz der zusätzlichen Bedingungen weiterhin gilt.

Lemma 4.9
Sei H eine normalisierte, turn-reguläre, orthogonale Repräsentation eines 4-planaren, 2-zusammenhängenden Graphen. Dann können die reflexiven Kanten in H sowie die jeweils gegenüberliegende Kante in einer Laufzeit von $\mathcal{O}(n)$ bestimmt werden.

Beweis:
Sei f eine beliebige Fläche. Durchlaufe die Kanten auf dem Rand von f zweimal – einmal vorwärts und einmal rückwärts. Prüfe jeweils den Winkel am Ende einer Kante. Wird dabei eine reflexive Kante e gefunden, können zwei Fälle auftreten. Nehme o. B. d. A. an, dass e horizontal verläuft.

Fall 1: Es existiert keine weitere horizontale Kante e', für die $\text{opp}(e')$ noch nicht bestimmt wurde.

Der Pfad P_{e,c_e} ist also nicht Teilpfad eines anderen Pfades $P_{e',c_{e'}}$. Entweder ist noch keine andere reflexive Kante e' aufgetreten oder für eine solche Kante e' wurde $\text{opp}(e')$ bereits bestimmt. Iteriere weiter entlang der Kanten von f und aktualisiere dabei in jeder Iteration

- die Rotation von e bis zur aktuell betrachteten Kante und
- die zuletzt gefundene Nicht-Dissecting-Kante auf dem Weg von e nach $\text{opp}(e)$.

Sobald die Rotation gleich 2 ist und die aktuell betrachtete Kante keine Dissecting-Kante ist, speichere $\text{opp}(e)$.

Fall 2: Es existiert eine weitere horizontale Kante e', für die $\text{opp}(e')$ noch nicht bestimmt wurde.

Der Pfad P_{e,c_e} ist dann Teilpfad eines anderen Pfades $P_{e',c_{e'}}$. Nutze aus, dass $\text{opp}(e) = \text{opp}(e')$ gilt und setze einen Zeiger von $\text{opp}(e)$ auf $\text{opp}(e')$. Somit genügt es, die Rotation für e' in jeder Iteration zu aktualisieren, für e ist dies nicht notwendig.

Die reflexiven Kanten in f und die gegenüberliegende Kanten können also während eines Vorwärts- und eines Rückwärts-Durchlaufes entlang der Kanten von f bestimmt werden. Da in planaren Graphen $2\sum_f n_f = \mathcal{O}(n)$ gilt, folgt die Behauptung. ■

Im Postprozessor-Schritt müssen zunächst die Nebenbedingungen für alle Paare von Partnerkanten überprüft und gegebenenfalls abgeändert werden, dies funktioniert in $\mathcal{O}(|A_D|) = \mathcal{O}(n)$. Anschließend wird das IP neu gelöst.

Sei $r(n)$ die Laufzeit, die benötigt wird, um das IP $(\star)$ bzw. die LP-Relaxierung von $(\star)$ zu lösen. Dann hat der gesamte Algorithmus eine Laufzeit von

$$\mathcal{O}\Big(\underbrace{n}_{\substack{\text{Dissecting-}\\\text{Kanten}\\\text{einfügen}}} + \underbrace{n}_{\substack{\text{Nebenbe-}\\\text{dingungen}\\\text{einfügen}}} + \underbrace{r(n)}_{\text{Kompaktierung}} + \underbrace{n + r(n)}_{\substack{\text{Postprozessor-}\\\text{Schritt}}} \Big) = \mathcal{O}(n + r(n)) \ .$$

4.3 Einschränkungen der Optimalität

Die Lösungen des eigenen Ansatzes sind im Allgemeinen nicht optimal unter allen möglichen Kompaktierungen. Dies hat mehrere Ursachen:

(U1) Im Abschnitt über den Postprozessor-Schritt wurde bereits auf die Einschränkung eingegangen, die aus der Vereinfachung der Nebenbedingungen

$$\max(x_{a_h}, 0) + \max(x_{a_v}, 0) \geq 1$$

zu

$$x_{a_h} + x_{a_v} \geq 1$$

resultiert.

Dadurch kann ein Paar von Kitty Corners (u, v) nicht alle relativen Positionen zueinander einnehmen.

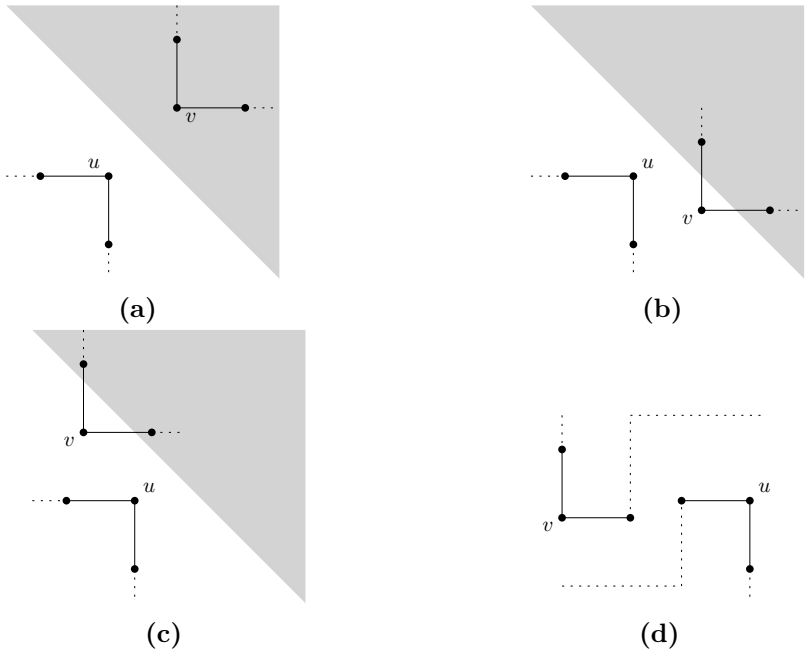


Abb. 4.12: Relative Positionen eines Kitty Corner-Paares

In Abbildung 4.12(a) kann v – bei fest gewähltem u – die Positionen in dem grau markierten Bereich einnehmen. Die Positionen in Abbildung 4.12(b) und 4.12(c) können hingegen nur durch den Postprozessor-Schritt erreicht werden. Die Position in Abbildung 4.12(d) kann gar nicht eingenommen werden.

(U2) Treten in einer Fläche mehrere Paare von Kitty Corners auf, so ist die Reihenfolge von Bedeutung, in der die Dissecting-Kanten eingefügt werden.

Im eigenen Ansatz wird stets das erste Kitty Corner-Paar in einer Fläche ausgewählt, anschließend wird eine Dissecting-Kante eingefügt und die entstandenen Teilflächen werden rekursiv auf Turn-Regularität geprüft. Dieses Vorgehen ist nicht optimal, da mögliche Anordnungen ausgeschlossen werden können. Insofern lassen sich lokale Entscheidungen im eigenen Ansatz nicht vollkommen vermeiden.

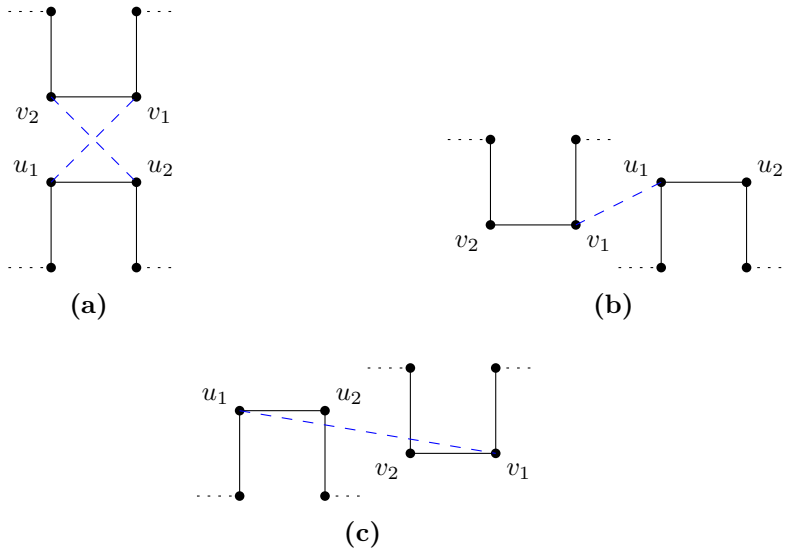


Abb. 4.13: Sich gegenseitig ausschließende Dissecting-Kanten

In der Fläche in Abbildung 4.13(a) treten die beiden Kitty Corner-Paare (u_1, v_1) und (u_2, v_2) auf, die sich gegenseitig ausschließen. Fügt man zuerst eine Dissecting-Kante zwischen u_1 und v_1 ein, so schränkt dies, wie unter (U1) beschrieben, die relativen Positionen von u_1 und v_1 zueinander ein. Abbildung 4.13(b) zeigt eine mögliche Anordnung von u_1 und v_1. Eine Anordnung wie in Abbildung 4.13(c) ist dann jedoch nicht mehr möglich. Diese kann nur erreicht werden, wenn zuerst das Kitty Corner-Paar (u_2, v_2) bearbeitet wird.

4.4 Existenz einer ganzzahligen Optimallösung

Minimum-Kosten-Fluss-Probleme besitzen immer eine ganzzahlige Optimallösung, sofern die Nebenbedingungen ganzzahlig sind und die Existenz einer Optimallösung gewährleistet ist. Dies besagt Satz 2.1.

Im Rahmen des eigenen Ansatzes wurden dem ursprünglichen Minimum-Kosten-Fluss-Problem die Nebenbedingungen

$$\begin{aligned} -\sum_{\substack{\text{prim}(a) \in P_{e,c},\\ \text{prim}(a) \text{ Innenkante}}} x_a + \sum_{\substack{\text{prim}(a) \in P_{e,c},\\ \text{prim}(a) \text{ Außenkante}}} x_a &\geq 1 \quad \forall \text{ Pfade } P_{e,c} \\ x_{a_h} + x_{a_v} &\geq 1 \quad \forall \text{ Partnerkanten } (a_h, a_v) \in A_h \times A_v \end{aligned}$$

hinzugefügt.

Auch das so abgewandelte Problem besitzt unter folgender Voraussetzung immer eine ganzzahlige Optimallösung:

Voraussetzung 4.10
Sei f eine beliebige Fläche, seien $a^{(1)}, a^{(2)} \in E$ zwei beliebige Dissecting-Kanten in f und seien $(a_h^{(1)}, a_v^{(1)}), (a_h^{(2)}, a_v^{(2)}) \in A_h \times A_v$ die zugehörigen Paare von Partnerkanten. Dann gelte entweder

(i) $a_h^{(1)}$ tritt in der vertikalen Flusserhaltungsbedingung mit demselben Vorzeichen auf wie $a_h^{(2)}$ und $a_v^{(1)}$ tritt in der horizontalen Flusserhaltungsbedingung mit demselben Vorzeichen auf wie $a_v^{(2)}$

oder

(ii) $a_h^{(1)}$ tritt in der vertikalen Flusserhaltungsbedingung mit anderem Vorzeichen auf als $a_h^{(2)}$ und $a_v^{(1)}$ tritt in der horizontalen Flusserhaltungsbedingung mit anderem Vorzeichen auf als $a_v^{(2)}$.

Es muss also ausgeschlossen werden, dass zwei Paare von Partnerkanten in einer Flusserhaltungsbedingung mit demselben Vorzeichen auftreten, in der anderen Flusserhaltungsbedingung aber mit unterschiedlichen Vorzeichen. Sofern jede Fläche maximal eine Dissecting-Kante enthält, ist Voraussetzung (4.10) immer erfüllt.

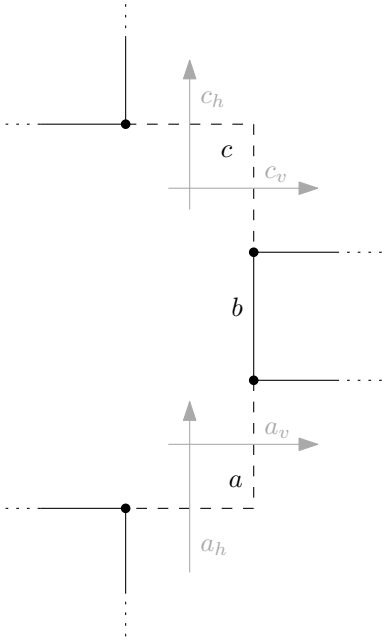


Abb. 4.14: Voraussetzung (4.10) nicht erfüllt

In Abbildung (4.14) beispielsweise ist Voraussetzung (4.10) verletzt, da a_v und c_v in der horizontalen Flusserhaltungsbedingung mit demselben Vorzeichen auftreten, a_h und c_h in der vertikalen Flusserhaltungsbedingung mit unterschiedlichen Vorzeichen:

$$\begin{array}{lccccccc} & & a_h & a_v & c_h & c_v & & \\ \text{hor. FEB} & (\ \cdots & 0 & -1 & 0 & -1 & \cdots &) \\ \text{vert. FEB} & (\ \cdots & 1 & 0 & -1 & 0 & \cdots &) \end{array}$$

Satz 4.11
Unter Voraussetzung (4.10) ist die Nebenbedingungsmatrix des ganzzahligen linearen Programms ($\star$) total unimodular.

Für den Beweis werden folgende Lemmata benötigt:

Lemma 4.12
Folgende Operationen erhalten totale Unimodularität:

1. *Vertauschen von Zeilen oder Spalten*
2. *Transponieren*
3. *Multiplikation einer Zeile oder Spalte mit* -1
4. *Hinzufügen einer Zeile oder Spalte mit genau einem Nichtnullelement mit Wert* ± 1
5. *Wiederholen einer Zeile oder einer Spalte*

Lemma 4.13
Sei A eine Matrix mit Einträgen -1, 0 oder 1. Dann gilt:

A ist total unimodular

$\Leftrightarrow$ *Jede Teilmenge $\mathcal{M}$ der Spalten von A kann in zwei disjunkte Teilmengen $\mathcal{M}^+$ und $\mathcal{M}^-$ geteilt werden, sodass die Summe der Spalten aus $\mathcal{M}^+$ abzüglich der Summe der Spalten aus $\mathcal{M}^-$ einen Vektor ergibt, dessen Komponenten -1, 0 oder 1 sind.*

Die Beweise der beiden Lemmata können beispielsweise in [Grötschel 2013, Kap. 12] gefunden werden.

Lemma 4.14
Sei G 2-zusammenhängend und seien f' und f'' zwei Flächen in G, die durch eine Dissecting-Kante getrennt werden. Dann besitzen f' und f'' außer der Dissecting-Kante keine weiteren gemeinsamen Kanten.

Beweis:
Die beiden Flächen f' und f'' werden durch eine Dissecting-Kante getrennt, haben also zuvor eine gemeinsame Fläche in G gebildet. Angenommen, es existieren eine oder mehrere Nicht-Dissecting-Kanten $e_1, \ldots, e_k$, die f' und f'' trennen, so wie die Kanten e_1 und e_2 in Abbildung 4.15. Dann könnte man den Graphen unzusammenhängend machen, indem man den Endknoten v einer dieser Kanten entfernt. Da der Graph aber nach Voraussetzung 2-zusammenhängend ist, kann solch ein Knoten v nicht existieren. Die beiden Flächen f' und f'' können also außer der Dissecting-Kante keine weiteren gemeinsamen Kanten besitzen. ■

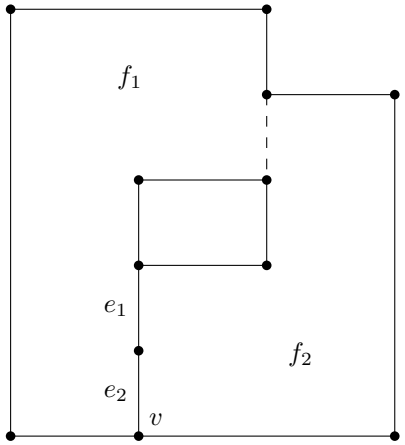


Abb. 4.15: Nicht-2-zusammenhängender Graph

Beweis von Satz 4.11:
Die Nebenbedingungen der LP-Relaxierung von $(\star)$ lassen sich umformen zu:

$$
\begin{array}{rrrcll}
 & & f_u(x) & \geq & 0 & \forall\ u \in U \\
 & & -f_u(x) & \geq & 0 & \forall\ u \in U \\
- & \sum\limits_{\substack{\text{prim}(a) \in P_{e,c},\\ \text{prim}(a)\ \text{Innenkante}}} x_a & + \sum\limits_{\substack{\text{prim}(a) \in P_{e,c},\\ \text{prim}(a)\ \text{Außenkante}}} x_a & \geq & 1 & \forall\ \text{Pfade } P_{e,c} \\
 & & x_{a_h} + x_{a_v} & \geq & 1 & \forall\ \text{Partnerkanten } (a_h, a_v) \in A_h \times A_v \\
 & & x_a & \geq & 1 & \forall\ a \in A \setminus A_D
\end{array}
$$

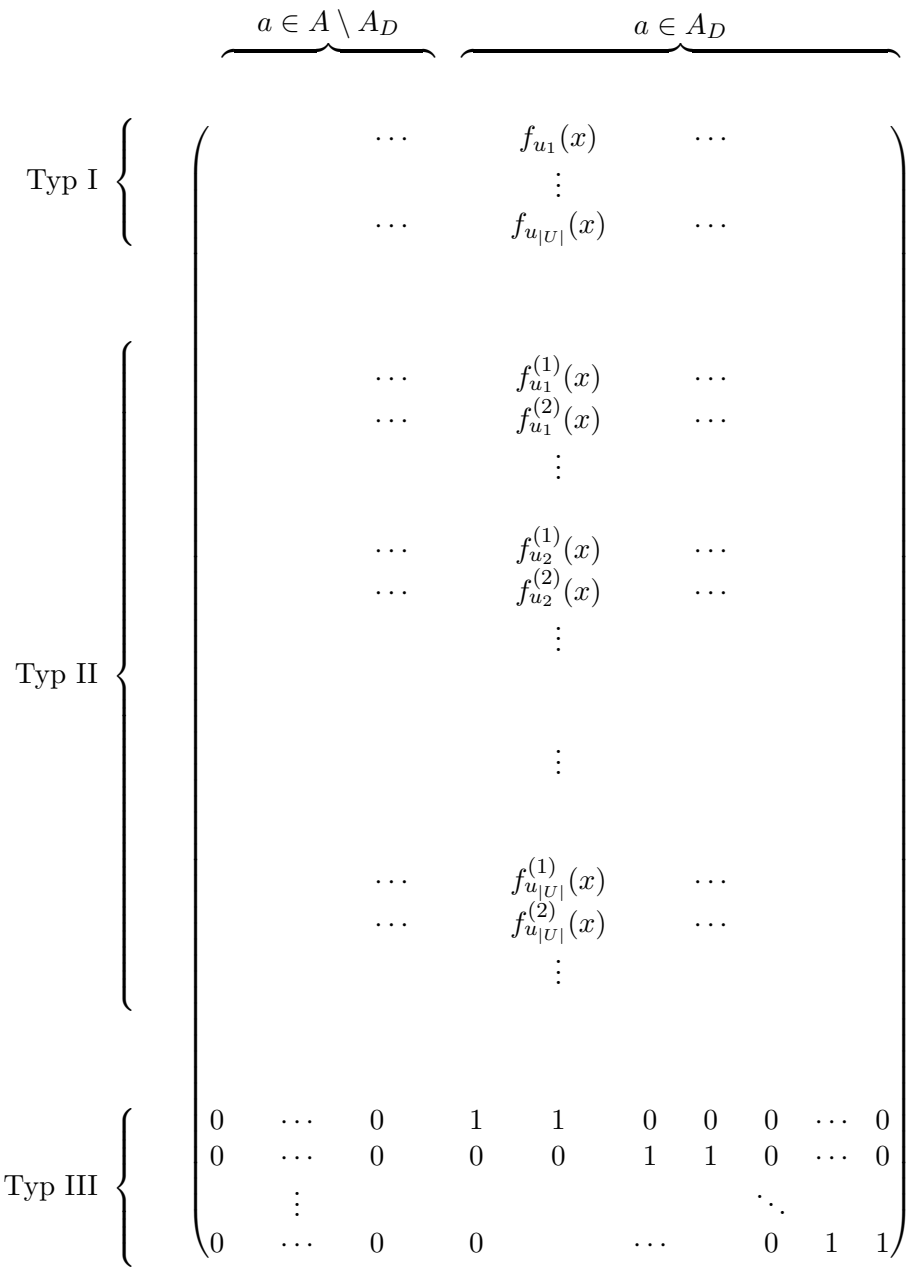
Sei A die Matrix, die die linke Seite dieser Nebenbedingungen beschreibt. Streiche alle Zeilen, die zu den Bedingungen

$$
\begin{array}{rcll}
-f_u(x) & \geq & 0 & \forall\ u \in U \\
x_a & \geq & 1 & \forall\ a \in A \setminus A_D
\end{array}
$$

gehören aus A, bezeichne die entstehende Matrix mit A'.

Nach Lemma 4.12 bleibt Unimodularität beim Einfügen einer vorhandenen Zeile und bei Multiplikation einer Zeile mit -1 erhalten. Ebenso bleibt Unimodularität erhalten, wenn man eine Matrix um einen Einheitsvektor erweitert. Daher genügt es zu zeigen, dass A' total unimodular ist.

Die Matrix A' hat folgende Form:



Da Unimodularität beim Vertauschen von Zeilen oder Spalten erhalten bleibt, kann angenommen werden, dass die Zeilen und Spalten in A' in dieser Reihenfolge angeordnet sind. Dabei ist $|U|$ die Anzahl der Flusserhaltungsbedingungen.

Typ I-Zeilen entsprechen Flusserhaltungsbedingungen im LP.

Typ II-Zeilen sind aus den Bedingungen

$$-\sum_{\substack{\text{prim}(a) \in P_{e,c},\\ \text{prim}(a)\ \text{Innenkante}}} x_a \quad + \sum_{\substack{\text{prim}(a) \in P_{e,c},\\ \text{prim}(a)\ \text{Außenkante}}} x_a \geq 1 \qquad \forall\ \text{Pfade}\ P_{e,c}$$

entstanden. Jede Typ II-Zeile stellt einen „Teil“ einer Typ I-Zeile dar, d. h. in allen Komponenten hat die Typ II-Zeile entweder denselben Eintrag wie die zugehörige Typ I-Zeile oder den Eintrag 0. Formal ausgedrückt:

$$a'_{jk} = a'_{ik} \quad \vee \quad a'_{jk} = 0 \quad \forall\ k,$$

wobei $a'_{j,\cdot}$ eine beliebige Typ II-Zeile ist und $a'_{i,\cdot}$ die zugehörige Typ I-Zeile in A'.

Mehrere Typ II-Zeilen können zu derselben Typ I-Zeile korrespondieren. Die Typ II-Zeilen, die zu der Flusserhaltungsbedingung $f_u(x)$ korrespondieren, wurden daher in der Matrix mit $f_u^{(1)}(x), f_u^{(2)}(x), \ldots$ bezeichnet. Abbildung 4.16 zeigt ein Beispiel für zusammengehörige Typ I- und Typ II-Zeilen:

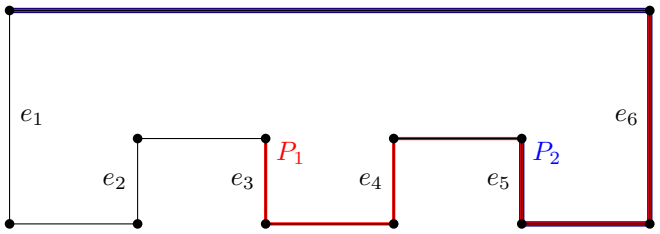


Abb. 4.16: Zusammengehörige Typ I- und Typ II-Zeilen

Sei $a_i := \text{dual}(e_i)$. Dann wird die horizontale Flusserhaltungsbedingung in der abgebildeten Fläche durch den Zeilenvektor

$$\begin{array}{cccccccccc} & & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & & \\ (& \cdots & +1 & -1 & +1 & -1 & +1 & -1 & \cdots &) \end{array}$$

beschrieben. Aus den Bedingungen für die beiden Pfade P_1 und P_2 ergeben sich die beiden Typ II-Zeilen

$$\begin{array}{cccccccccc} & & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & & \\ (& \cdots & 0 & 0 & +1 & -1 & +1 & -1 & \cdots &) \\ (& \cdots & 0 & 0 & 0 & 0 & +1 & -1 & \cdots &) \end{array}$$

die jeweils einen „Teil“ der obigen Typ I-Zeile darstellen. Sollte eine Typ II-Zeile $a'_{j,\cdot}$ das umgekehrte Vorzeichen haben wie die zugehörige Typ I-Zeile $a'_{k,\cdot}$, so kann sie o. B. d. A. mit -1 multipliziert werden.

Typ III-Zeilen sind aus den Bedingungen $x_{a_h} + x_{a_v} \geq 1$ entstanden.

Die vorderen Spalten der Matrix A' gehören zu Kanten, deren primale Kante keine Dissecting-Kante ist. Die hinteren Spalten gehören zu Kanten in A_D.

Nach Lemma 4.12 ist A' genau dann total unimodular, wenn $(A')^T$ total unimodular ist. Nach Lemma 4.13 genügt es also zu zeigen, dass man jede Teilmenge der Zeilen von A' so in zwei disjunkte Gruppen $\mathcal{M}^+$, $\mathcal{M}^-$ einteilen kann, dass der Differenzvektor der Gruppensummen nur die Komponenten -1, 0 oder 1 enthält.

Sei $\mathcal{M}$ also eine beliebige Teilmenge der Zeilen von A'. Sei M die durch $\mathcal{M}$ induzierte Matrix, d. h.

$$M = \left(a'_{i,\cdot}\right)_{i \in \mathcal{M}} .$$

Nehme wiederum an, dass die Zeilen von M so geordnet sind wie in A' und teile die Zeilen von M wiederum in Typ I-, Typ II- und Typ III-Zeilen ein.

Dann führe folgende Operationen auf M durch:

1. Bilde die Differenz zusammengehöriger Typ II-Zeilen.
2. Bilde die Summe bzw. Differenz von Typ I-Zeilen mit dem zugehörigen Differenzvektor aus Schritt 1.
3. Bilde die Summe bzw. Differenz von Typ III-Zeilen mit den zugehörigen Summen- oder Differenzvektoren aus Schritt 2.

Die einzelnen Schritte werden im Folgenden näher erläutert:

Schritt 1:
Seien $m_{1,\cdot}, \ldots, m_{k,\cdot}$ Typ II-Zeilen in M, die zu derselben Typ I-Zeile korrespondieren. Dann bilde die Differenz dieser Typ II-Zeilen, sodass Einträge in dem Differenzvektor verschwinden. Für $k > 2$ bilde nach dem Distributivgesetz jeweils die Differenz mit dem vorherigem Differenzvektor, d. h. $m_{k,\cdot} - (m_{k-1,\cdot} - (\ldots (m_{3,\cdot} - (m_{2,\cdot} - m_{1,\cdot})) \ldots))$. Ersetze die voneinander subtrahierten Zeilen in M durch ihren Differenzvektor. Dadurch werden zusammengehörige Typ II-Zeilen jeweils zu einer Zeile mit Einträgen -1, 0 oder 1 zusammengefasst.

In dem obigen Beispiel werden die beiden Typ II-Zeilen

$$\begin{array}{cccccccccc} (& \cdots & 0 & 0 & +1 & -1 & +1 & -1 & \cdots &) \\ (& \cdots & 0 & 0 & 0 & 0 & +1 & -1 & \cdots &) \end{array}$$

durch ihren Differenzvektor

$$\begin{array}{cccccccccc} (& \cdots & 0 & 0 & +1 & -1 & 0 & 0 & \cdots &) \end{array}$$

ersetzt.

Schritt 2:
M besteht aus einer beliebigen Teilmenge der Zeilen in A'. Falls M die Typ I-Zeile enthält, die zu dem Differenzvektor aus Schritt 1 korrespondiert, addiere oder subtrahiere diese beiden Zeilen, sodass Einträge verschwinden. Ersetze die beiden Zeilen in M durch den entstehenden Summen- oder Differenzvektor.

In dem obigen Beispiel wird folgende Subtraktion durchgeführt:

$$\begin{array}{cccccccccc} & (& \cdots & +1 & -1 & +1 & -1 & +1 & -1 & \cdots &) \\ - & (& \cdots & 0 & 0 & +1 & -1 & 0 & 0 & \cdots &) \\ \hline & (& \cdots & +1 & -1 & 0 & 0 & +1 & -1 & \cdots &) \end{array}$$

Durch Schritt 1 und 2 werden alle Bedingungen, die zu derselben Fläche im horizontalen bzw. vertikalen Netzwerk korrespondieren, zu einem Zeilenvektor zusammengefasst. Nach Schritt 1 und 2 existieren in M für jede Fläche maximal zwei Zeilenvektoren – ein Zeilenvektor, der aus den horizontalen Bedingungen entstanden ist, und ein Zeilenvektor, der aus den vertikalen Bedingungen entstanden ist.

Schritt 3:
Betrachte zwei Spalten in M, die zu einem beliebigen Paar von Partnerkanten (a_h, a_v) gehören. In jeder dieser Spalten existieren maximal drei Einträge ungleich 0:

$$\begin{array}{c} \begin{matrix} a_h & a_v & \end{matrix} \\ \begin{pmatrix} \cdots & 1 & 0 & \cdots \\ \cdots & 0 & 1 & \cdots \\ \cdots & -1 & 0 & \cdots \\ \cdots & 0 & -1 & \cdots \\ \cdots & 1 & 1 & \cdots \end{pmatrix} \end{array}$$

Die ersten vier Zeilen sind Summen- bzw. Differenzvektoren, die in Schritt 1 und 2 entstanden sind. a_h tritt in diesen Zeilen maximal einmal mit positivem und einmal mit negativem Vorzeichen auf, als eingehende oder ausgehende Kante in einer Flusserhaltungsbedingung bzw. in einer zusammengefassten Typ II-Bedingung. Dasselbe gilt für a_v. Die letzte Zeile ergibt sich aus der Bedingung $x_{a_h} + x_{a_v} \geq 1$.

Diese fünf Zeilen können problemlos zusammengefasst werden, ohne dass in einer der restlichen Komponenten ein Eintrag ungleich -1, 0 oder 1 entsteht: Zwei Zeilen der Form

$$\begin{pmatrix} \cdots & \pm 1 & 0 & \cdots \\ \cdots & 0 & \pm 1 & \cdots \end{pmatrix}$$

können addiert oder subtrahiert werden, da die erste Zeile zu horizontalen Bedingungen gehört, die zweite Zeile zu vertikalen. Besitzt eine dieser Zeilen an einer Stelle einen Eintrag ungleich 0, so ist die andere Zeile an dieser Stelle zwangsläufig gleich 0.

Zwei Zeilen der Form

$$\begin{pmatrix} \cdots & \pm 1 & 0 & \cdots \\ \cdots & \mp 1 & 0 & \cdots \end{pmatrix}$$

besitzen im restlichen Teil der Matrix ebenfalls keine gemeinsamen Spalteneinträge. Die primale Kante zu a_h ist eine Dissecting-Kante, trennt also zwei Flächen f' und f'', die zuvor eine gemeinsame Fläche gebildet haben. Nach Lemma 4.14 können f' und f'' außer der Dissecting-Kante keine weiteren gemeinsamen Kanten besitzen. Wenn eine der beiden obigen Zeilen also einen Eintrag ungleich 0 besitzt, ist die andere Zeile an dieser Stelle gleich 0. Die fünfte Zeile kann zu allen anderen addiert werden, da sie in den übrigen Spalten ohnehin nur Nullen enthält.

Addiere oder subtrahiere diese maximal fünf Zeilen in Schritt 3 so, dass ein Zeilenvektor mit Einträgen -1, 0 oder 1 entsteht. Auch wenn M nur eine Teilmenge dieser fünf Zeilen enthält, lassen sich diese Zeilen unter Voraussetzung (4.10) immer so zusammenfassen, dass eine der Zeilen $(\cdots 11 \cdots)$, $(\cdots 10 \cdots)$, $(\cdots 01 \cdots)$ oder $(\cdots 00 \cdots)$ entsteht. Ersetze die zusammengefassten Zeilen in M durch diesen Summen- oder Differenzvektor. Im Anschluss an Schritt 3 existiert in jeder Spalte von M maximal ein Nichtnullelement mit Wert ± 1.

Während in Schritt 1 und 2 Zeilen zusammengefasst wurden, die ausschließlich zu horizontalen oder ausschließlich zu vertikalen Bedingungen korrespondieren, ist durch Schritt 3 erstmals eine Verbindung zwischen horizontalen und vertikalen Bedingungen entstanden. Die erste und dritte der obigen fünf Zeilen beziehen sich auf eine horizontale Bedingung, die zweite und vierte Zeile auf eine vertikale Bedingung. Um sicherzustellen, dass sich solche Zeilen widerspruchsfrei zusammenfassen lassen, ist Voraussetzung (4.10) notwendig. Dies wird im Anschluss an den Beweis an einem Beispiel erläutert.

Die Additionen und Subtraktionen auf M induzieren eine Zerlegung von $\mathcal{M}$ in $\mathcal{M}^+$ und $\mathcal{M}^-$: Werden zwei Zeilen durch ihren Differenzvektor ersetzt, dann ordne eine Zeile der Menge $\mathcal{M}^+$ und eine der Menge $\mathcal{M}^-$ zu. Werden zwei Zeilen durch ihre Summe ersetzt, dann ordne beide derselben Menge zu. Nach Schritt 1, 2 und 3 besitzt M in jeder Spalte maximal einen Eintrag mit Wert 1 oder -1, alle anderen Einträge in dieser Spalte sind gleich 0. Damit folgt dass der Differenzvektor der Zeilen in $\mathcal{M}^+$ und der Zeilen in $\mathcal{M}^-$ ebenfalls nur die Einträge -1, 0 und 1 besitzt. A' ist total unimodular, somit ist auch A total unimodular. ■

Korollar 4.15
Unter Voraussetzung (4.10) besitzt die LP-Relaxierung von ($\star$) eine ganzzahlige Optimallösung.

Voraussetzung (4.10) wird benötigt, um die Zeilen in Schritt 3 zu einem Zeilenvektor mit Einträgen -1, 0 und 1 zusammenfassen zu können. In Abbildung 4.14 war Voraussetzung (4.10) nicht erfüllt, da a_v und c_v in der horizontalen Flusserhaltungsbedingung mit demselben Vorzeichen auftreten, a_h und c_h in der vertikalen Flusserhaltungsbedingung aber mit unterschiedlichen Vorzeichen. Die Zeilenvektoren lassen sich dann nicht mehr problemlos zusammenfassen:

		a_h	a_v	c_h	c_v		
hor. FEB	($\cdots$	0	-1	0	-1	$\cdots$	)
vert. FEB	($\cdots$	1	0	-1	0	$\cdots$	)
Typ III-Bed.	($\cdots$	1	1	0	0	$\cdots$	)
Typ III-Bed.	($\cdots$	0	0	1	1	$\cdots$	)

Damit Einträge verschwinden, werden die Zeilen 1 und 3 sowie die Zeilen 2 und 4 durch ihre Summenvektoren ersetzt. Dadurch entstehen folgende Zeilenvektoren:

	a_h	a_v	c_h	c_v		
($\cdots$	1	0	0	-1	$\cdots$	)
($\cdots$	1	0	0	1	$\cdots$	)

Diese lassen sich nicht addieren oder subtrahieren, sodass ein Vektor mit Einträgen -1, 0 oder 1 entsteht.

Probeweise kann man die Determinante der obigen Matrix berechnen:

$$\det \begin{pmatrix} 0 & -1 & 0 & -1 \\ 1 & 0 & -1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} = 2,$$

diese Teilmatrix von A' ist also nicht unimodular.

5 Experimentelle Analyse

Um den eigenen Ansatz mit anderen Heuristiken zu vergleichen, wurde eine experimentelle Analyse verschiedener Kompaktierungsheuristiken durchgeführt. In den ersten beiden Abschnitten wird zunächst kurz auf die Implementierung und die Auswahl der Testinstanzen eingegangen, bevor anschließend die Ergebnisse der Analyse vorgestellt werden.

5.1 Implementierung

Der eigene Ansatz wurde in C++ implementiert. Dabei wurden verschiedene Bibliotheken und Auszeichnungssprachen genutzt, die grundlegende Datenstrukturen, Algorithmen und Schnittstellen zum Graphenzeichnen bereitstellen.

5.1.1 OGDF

Das Open Graph Drawing Framework (OGDF) ist eine frei verfügbare C++-Bibliothek, die Algorithmen und Datenstrukturen zum Zeichnen von Graphen beinhaltet. Der OGDF-Algorithmus zum Zeichnen orthogonaler Graphen basiert auf dem Topology-Shape-Metrics-Ansatz. Ein Graph wird zunächst planar eingebettet, bevor eine orthogonale Repräsentation des Graphen bestimmt wird. Diese Schritte wurden beibehalten, Änderungen wurden nur am Kompaktierungsschritt vorgenommen: Zusätzlich zu der rechteckigen Dissecting-Methode, der konstruktiven Heuristik und der Verbesserungsheuristik, die bereits in OGDF enthalten sind, wurden der Turn-Regularity-Ansatz und der eigene Ansatz implementiert. Dem Anwender wurde die Möglichkeit gegeben, zwischen diesen verschiedenen Dissecting- und Kompaktierungsmethoden zu wählen (vgl. [Chimani et al. 2013], [OGDF 2012]).

5.1.2 COIN

Zum Lösen der linearen Programme, die im Rahmen des eigenen Ansatzes und des Turn-Regularity-Ansatzes aufgestellt wurden, wurden COIN und der darin integrierte CLP-Löser benutzt. COIN (Computational Infrastructure for Operations Research)

besteht aus mehreren frei verfügbaren C++-Bibliotheken zum Lösen von Optimierungsproblemen und beinhaltet unter anderem eine Implementierung des Simplex-Verfahrens sowie ein Branch-and-Bound-Verfahren zum Lösen ganzzahliger linearer Programme (vgl. [COIN 2012]).

5.1.3 GML und GraphML

Die Graph Modelling Language (GML) und deren Weiterentwicklung GraphML bieten die Möglichkeit, Zeichnungen von Graphen anwendungsunabhängig zu speichern. GML und GraphML sind frei verfügbare Auszeichnungssprachen, die auf Textdateien bzw. XML-Dateien basieren. Verschiedene Bibliotheken und Programme, wie OGDF oder der Grapheditor yEd, besitzen Schnittstellen, um GML- und GraphML-Dateien einzulesen und auszugeben (vgl. [Brandes et al. 2013]).

5.2 Testinstanzen

Für die experimentelle Analyse wurden verschiedene Arten von Testinstanzen verwendet:

- **Typ-A-Instanzen** sind 4-planare, 2-zusammenhängende Zufallsgraphen. Dazu wurden zunächst 1.000 Graphen mit zufälliger Knotenzahl $10 \leq |V| \leq 1.000$ und zufälliger Kantenzahl $10 \leq |E| \leq 2.500$ generiert. Alle Kantenkreuzungen in nicht-planaren Graphen wurden durch zusätzliche Knoten ersetzt. An allen Knoten v mit $\deg(v) > 4$ wurden Kanten entfernt. Abschließend wurde die größte 2-zusammenhängende Komponente des Graphen bestimmt, alle anderen Komponenten wurden verworfen. Die entstandenen Graphen bestehen aus 162 bis 878 Knoten und 266 bis 1.415 Kanten.

- **Typ-B-Instanzen** stammen aus der „Rome-Testsuite“, einer Menge von 11.582 Graphen, die erstmals 1997 von Wissenschaftlern zweier römischer Universitäten verwendet wurde. Diese Testsuite hat sich zu einem Standard im Graphenzeichnen entwickelt und wird in vielen experimentellen Vergleichen verwendet (vgl. [Di Battista et al. 1997], [ROME 1997]).

  Die Graphen in dieser Testsuite sind im Allgemeinen weder 2-zusammenhängend noch 4-planar. Da der eigene Ansatz nur für 2-zusammenhängende Graphen entwickelt wurde, wurden alle nicht-2-zusammenhängenden Graphen verworfen, übrig blieben 58 Instanzen. Sollte eine Instanz nicht planar sein, werden im Rahmen des Topology-Shape-Metrics-Ansatzes Kantenkreuzungen durch künstliche Knoten ersetzt. Alle Knoten v mit $\deg(v) > 4$ werden anschließend expandiert. Im Anschluss an die Kompaktierungsphase werden alle künstlichen Knoten wieder entfernt.

5.3 Ergebnisse

Im Folgenden werden die Ergebnisse der experimentellen Analyse vorgestellt. Dabei wurden folgende Dissecting-Methoden verwendet:

- **Rechteckige Dissecting-Methode**, wie in Abschnitt 3.2.1 beschrieben.
- **Klassischer Turn-Regularity-Ansatz**, wie in Abschnitt 3.4.2 beschrieben. Die Ausrichtung der Dissecting-Kanten wird dabei zufällig gewählt und schon während des Dissecting-Schrittes festgelegt.
- **Eigener Turn-Regularity-Ansatz:** Im Unterschied zum klassischen Turn-Regularity-Ansatz wird die Ausrichtung der Dissecting-Kanten noch nicht festgelegt.

Zur Kompaktierung wurden die folgenden Algorithmen benutzt:

- **Konstruktive Heuristik**, wie in Abschnitt 3.2.1 beschrieben.
- **Konstruktive Heuristik mit anschließender Verbesserungsheuristik**, wie in Abschnitt 3.2.2 beschrieben.
- **Lösen des IP:** Das Kompaktierungsproblem wird als IP formuliert, wie in Kapitel 4 beschrieben.
- **Lösen der LP-Relaxierung:** Statt des IP wird die LP-Relaxierung gelöst.

Hierbei ist zu beachten, dass nicht alle Kombinationen aus Dissecting-Methoden und Kompaktierungsheuristiken zulässig sind. Wurde als Dissecting-Methode einer der beiden Turn-Regularity-Ansätze gewählt, so schließt dies eine Anwendung der konstruktiven Heuristik oder Verbesserungsheuristik aus, da die entstehenden Flächen nicht rechteckig sind. Bei Anwendung des eigenen Ansatzes schließt die Kompaktierung außerdem den beschriebenen Postprozessor-Schritt mit ein.

Eine Übersicht über alle Dissecting- und Kompaktierungsmethoden und die im Folgenden verwendeten Abkürzungen findet sich in Tabellen 5.1 und 5.2.

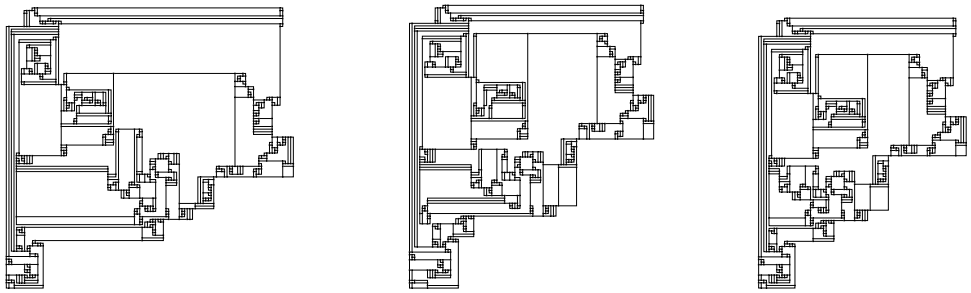
Dissecting-Methode	**Abkürzung**
rechteckige Methode	R
klassischer Turn-Regularity-Ansatz	TRC
eigener Turn-Regularity-Ansatz	TRO

Tab. 5.1: Dissecting-Methoden

Kompaktierungs-Methode	Abkürzung
konstruktive Heuristik	C
konstruktive Heuristik mit anschl. Verbesserungsheuristik	CI
Lösen des IP	IP
Lösen der LP-Relaxierung	LP

Tab. 5.2: Kompaktierungsmethoden

Abbildung 5.1 zeigt beispielhaft eine Testinstanz, Instanz „A293.gml“, nach Anwendung der verschiedenen Heuristiken. Die zugehörigen Ergebnisse sind in Tabelle 5.3 angegeben.



(a) R-CI **(b)** TRC-IP **(c)** TRO-IP

Abb. 5.1: Instanz „A293.gml“ bei Anwendung verschiedener Heuristiken

Heuristik	$\|E\|$	$\|V\|$	# Dissecting-Kanten	Gesamtkantenlänge	Breite	Höhe	Fläche	# B&B-Iterationen	Postprozessor-Verbesserung	Laufzeit Dissecting (Sek.)	Laufzeit Kompaktierung (Sek.)
R-C	688	1.032	5.716	98.314	170.978	211.193	$3{,}2{\cdot}10^{15}$			0,16	11,89
R-CI	688	1.032	5.716	3.276	5.420	5.290	$2{,}9{\cdot}10^{12}$			0,13	23,82
TRC-LP	688	1.032	11	3.154	4.665	5.400	$2{,}5{\cdot}10^{12}$			0,01	5,57
TRC-IP	688	1.032	11	3.154	4.665	5.400	$2{,}5{\cdot}10^{12}$	0		0,01	5,20
TRO-LP	688	1.032	11	2.882	3.930	5.106	$2{,}0{\cdot}10^{12}$		139	0,01	5,04
TRO-IP	688	1.032	11	2.882	3.930	5.106	$2{,}0{\cdot}10^{12}$	0	139	0,01	5,22

Tab. 5.3: Detaillierte Ergebnisse für die Instanz „A293.gml“

Auf einer der 1.058 Testinstanzen wurde bei den Branch-and-Bound-Iterationen das Zeitlimit von 3.600 Sekunden überschritten. Das Branch-and-Bound-Verfahren wurde daraufhin abgebrochen, mit dem eigenen Ansatz wurde keine Zeichnung erzeugt. Diese Instanz wurde bei der folgenden Analyse daher nicht berücksichtigt.

5.3.1 Vergleich der Gesamtkantenlänge

Abbildung 5.2 zeigt die Gesamtkantenlänge der 999 Typ-A-Instanzen bei Anwendung der eindimensionalen Heuristiken (R-CI), des klassischen Turn-Regularity-Ansatzes (TRC-IP) und des eigenen Ansatzes (TRO-IP). Alle Ergebnisse sind relativ zum R-CI-Ergebnis dargestellt, welches mit 100% angesetzt wurde. Der Übersichtlichkeit halber wurden die 999 Instanzen jeweils zu Gruppen von 20 bzw. einmal von 19 Instanzen zusammengefasst.

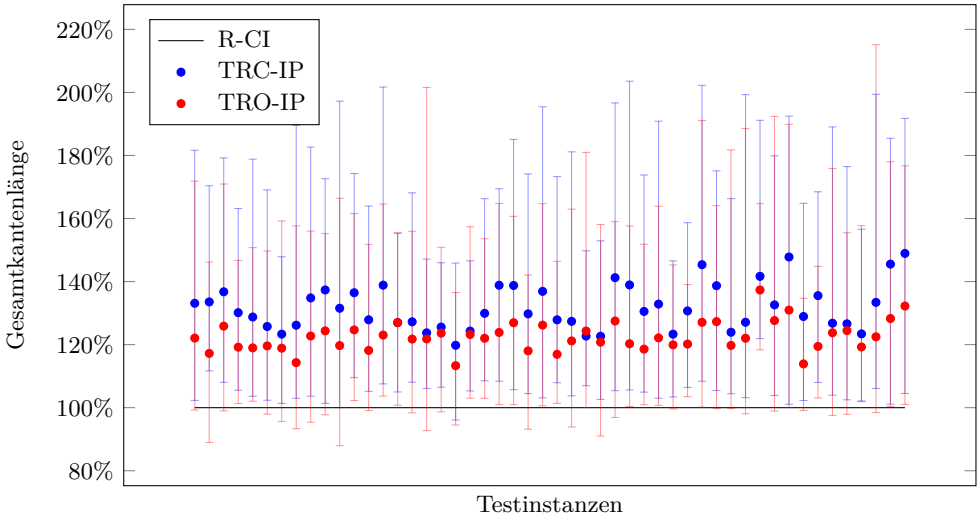
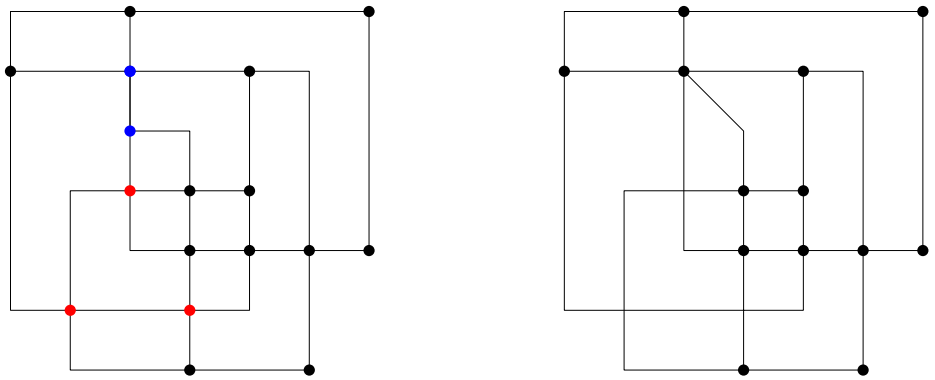


Abb. 5.2: Vergleich der Gesamtkantenlänge (Typ A)

Bei Anwendung des eigenen Ansatzes lag die Gesamtkantenlänge zwischen 87,97% und 215,12%, im Durchschnitt bei 122,48%. Bei Anwendung des klassischen Turn-Regularity-Ansatzes lag die Gesamtkantenlänge zwischen 96,09% und 203,55%, im Durchschnitt bei 131,80%. Die konstruktive Heuristik mit anschließender Verbesserungsheuristik führte also im Durchschnitt zu einer geringeren Gesamtkantenlänge als der klassische Turn-Regularity-Ansatz und der eigene Ansatz. Dies zeigt, dass die eindimensionalen Heuristiken trotz ihrer Schwächen vergleichsweise gute Ergebnisse liefern.

Besonderheiten auf Typ-B-Instanzen

An den Typ-B-Instanzen wird die Funktionsweise des Topology-Shape-Metrics-Ansatzes besonders gut deutlich, da diese Instanzen im Allgemeinen nicht 4-planar sind. Die roten Knoten in Abbildung 5.3(a) sind in der Planarisierungsphase eingefügt worden, um Kantenkreuzungen zu vermeiden. Die beiden blauen Knoten haben einen Grad-5-Knoten ersetzt. Nach Abschluss der Kompaktierungsphase wurden die künstlichen Knoten wieder entfernt, wodurch Zeichnung 5.3(b) entstanden ist.



(a) (b)

Abb. 5.3: Instanz „B1.gml“ bei Anwendung der R-CI-Heuristik

Die Ergebnisse auf Typ-B-Instanzen unterschieden sich qualitativ nicht von denen auf Typ-A-Instanzen. Auf Typ-B-Graphen führte der eigene Ansatz zu einer Gesamtkantenlänge von durchschnittlich 132,97%, der Turn-Regularity-Ansatz zu durchschnittlich 133,57%. Die Ergebnisse auf Typ-B-Instanzen sind allerdings mit Vorsicht zu genießen, da die Kantenlängen nicht in der $\|\cdot\|_2$-Norm, sondern der $\|\cdot\|_1$-Norm gemessen wurden. In Abbildung 5.3(b) beispielsweise würde dies einen Unterschied machen.

Vergleich des eigenen Ansatzes mit dem Turn-Regularity-Ansatz

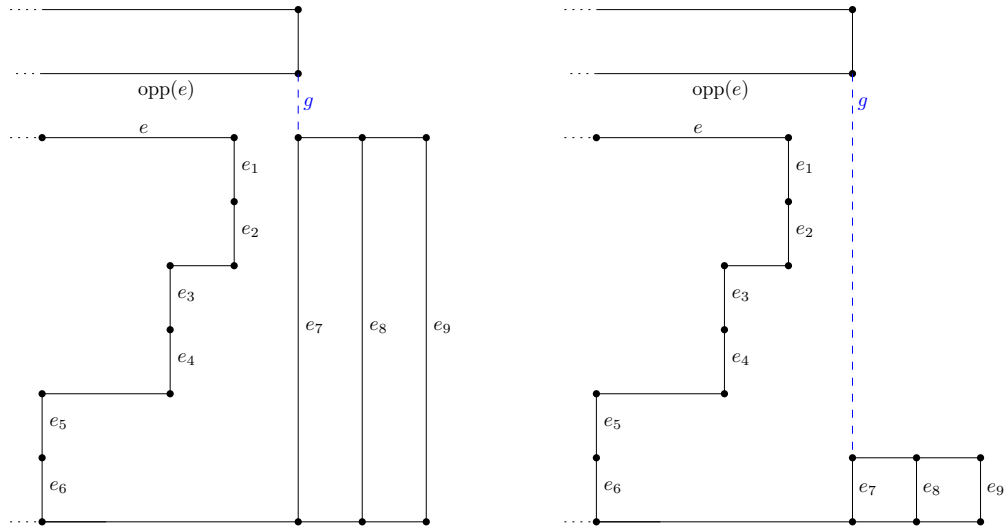
Interessant ist auch der Vergleich des Turn-Regularity-Ansatzes mit dem eigenen Ansatz. Auf 84,96% der 1.057 Instanzen lieferte der eigene Ansatz bessere Ergebnisse, auf 15,04% der Turn-Regularity-Ansatz.

Intuitiv sollte der eigene Ansatz immer zu besseren Ergebnissen führen als der Turn-Regularity-Ansatz, da er mehr Freiheiten bei der Ausrichtung der Dissecting-Kanten lässt. Da Dissecting-Kanten beim eigenen Ansatz jedoch eine negative Kantenlänge haben dürfen, sind auch mehr Nebenbedingungen notwendig, die schlimmstenfalls zu einem schlechteren Ergebnis führen als beim Turn-Regularity-Ansatz. Dies kann passieren, wenn der Sonderfall aus Abschnitt 4.1.1 auftritt und die vorletzte Kante auf einem Pfad eine Dissecting-Kante ist.

Abbildung 5.4 zeigt ein solches Beispiel. Bei Anwendung des eigenen Ansatzes entsteht Zeichnung 5.4(a). Da g, die Vorgängerkante von $\text{opp}(e)$, eine Dissecting-Kante ist, werden dem IP zwei Nebenbedingungen hinzugefügt:

$$\begin{aligned} -x_1 - x_2 - x_3 - x_4 - x_5 - x_6 + x_7 + x_g &\geq 1 \\ -x_1 - x_2 - x_3 - x_4 - x_5 - x_6 + x_7 &\geq 1 \end{aligned}$$

Dabei ist x_k der Fluss auf der dualen Kante zu e_k ist und x_g die Länge der vertikalen Dissecting-Kante g. Die Kante e_7 kann wegen der zweiten Bedingung nicht kürzer



(a) TRO-IP **(b)** TRC-IP

Abb. 5.4: Nebenbedingungen bei Auftreten des Sonderfalls

gewählt werden; e_8 und e_9 müssen dann genauso lang gewählt werden. Zeichnungen wie in Abbildung 5.4(a), die man durch „geschicktes Hinschauen“ weiter kompaktieren könnte, lassen sich beim eigenen Ansatz daher nicht vollständig vermeiden. Im Turn-Regularity-Ansatz sind die obigen Nebenbedingungen nicht notwendig, die Kantenlängen von e_7, e_8 und e_9 können dann so wie in Abbildung 5.4(b) gewählt werden. Hierbei handelt es sich jedoch, wie oben beschrieben, um einen Sonderfall, der selten aufgetreten ist. Üblicherweise lieferte der eigene Ansatz bessere Ergebnisse als der Turn-Regularity-Ansatz.

5.3.2 Laufzeitvergleich

Dissecting-Schritt

Abbildung 5.5 gibt die Laufzeit des Dissecting-Schrittes bei Anwendung der verschiedenen Methoden an. Die rechteckige Methode (R) benötigte durchschnittlich 0,096 Sekunden, der klassische Turn-Regularity-Ansatz (TRC) 0,013 Sekunden und der eigene Ansatz (TRO) 0,017 Sekunden. Die 1.057 Instanzen wurden wieder zu Gruppen von 19 bzw. 20 Instanzen zusammengefasst. Bei den letzten drei Gruppen handelt es sich um die Typ-B-Instanzen. Diese bestehen aus weniger Kanten als die Typ-A-Instanzen, daher ist die Laufzeit geringer ist als bei den Typ-A-Instanzen.

Auffällig ist, dass die Laufzeit der rechteckigen Methode immer über der Laufzeit des klassischen Turn-Regularity-Ansatzes und des eigenen Ansatzes liegt. Dies lässt sich darauf zurückführen, dass beim rechteckigen Ansatz deutlich mehr Dissecting-Kanten eingefügt werden müssen, wie in Abschnitt 5.3.4 ersichtlich wird. Die Laufzeit des Turn-Regularity-Ansatzes lag meist unter der Laufzeit des eigenen Ansatzes, da beim eigenen Ansatz zusätzliche Nebenbedingungen eingefügt werden müssen.

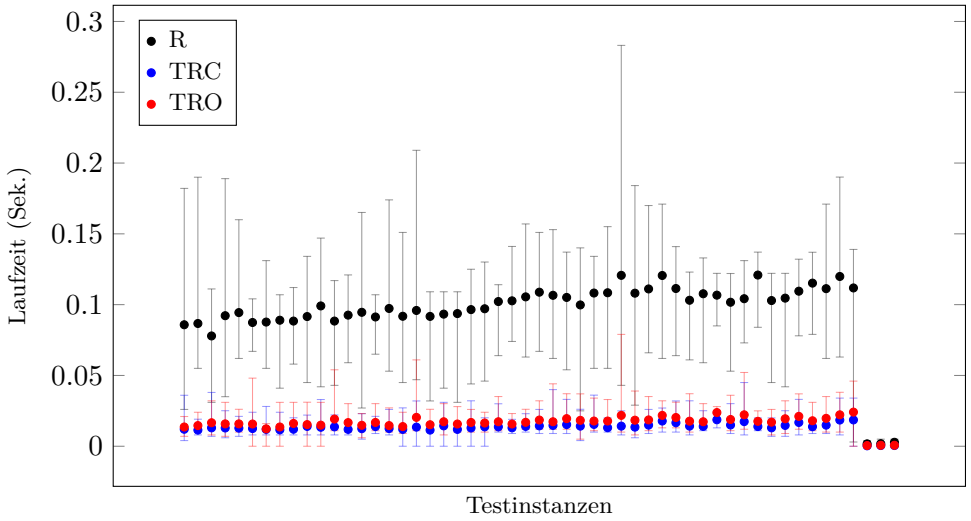


Abb. 5.5: Laufzeit (Dissecting)

Kompaktierungsschritt

Abbildung 5.6 gibt die Laufzeit des Kompaktierungsschrittes an. Bei Anwendung der konstruktiven Heuristik mit anschließender Verbesserungsheuristik lag die Laufzeit zwischen 0,138 Sekunden und 33,932 Sekunden, im Durchschnitt bei 16,814 Sekunden. Der eigene Ansatz hatte eine Laufzeit zwischen 0,039 Sekunden und 488,327 Sekunden, durchschnittlich 6,394 Sekunden. Dabei ist die Laufzeit nicht die Zeit zum reinen Lösen des IP, sondern umfasst auch das Erzeugen des horizontalen und vertikalen Netzwerkes sowie den Postprozessor-Schritt.

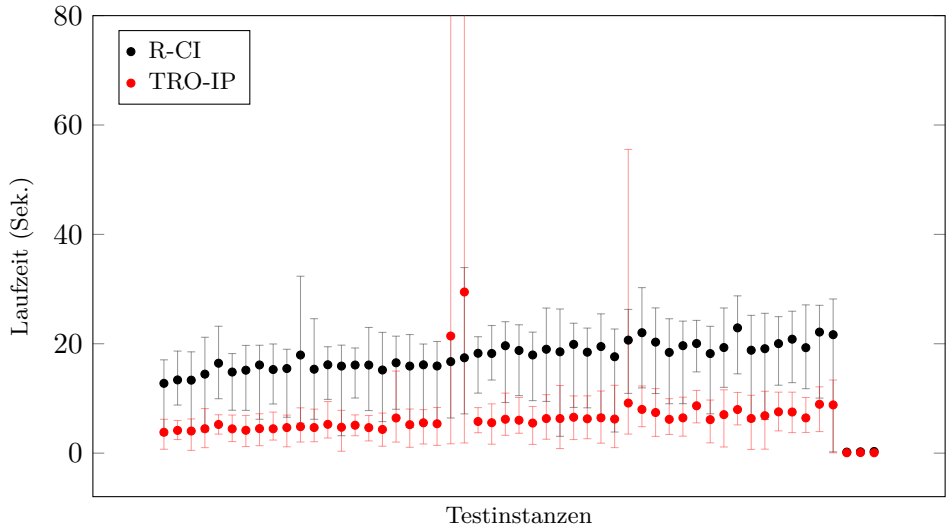


Abb. 5.6: Laufzeit (Kompaktierung)

Auf einzelnen Instanzen war der eigene Ansatz also langsamer als die eindimensionalen Heuristiken, insbesondere falls mehrere Branch-and-Bound-Iterationen durchgeführt werden mussten. Im Durchschnitt war der eigene Ansatz jedoch mehr als doppelt so schnell wie die eindimensionalen Heuristiken. Mit dem COIN-CLP-Löser wurde zudem ein vergleichsweise langsamer Löser benutzt. Kommerzielle Löser wie CPLEX können Optimierungsprobleme noch schneller lösen.

Fasst man die Laufzeit des Dissecting- und des Kompaktierungsschrittes zusammen, so benötigten die eindimensionalen Heuristiken durchschnittlich 2,64-mal so lange wie der eigene Ansatz.

5.3.3 Vergleich von LP-Relaxierung und IP

Beim Lösen der LP-Relaxierung von ($\star$) werden die Ganzzahligkeitsbedingungen fallen gelassen. Dadurch kann die LP-Relaxierung fraktionale Lösungen besitzen, die Gesamtkantenlänge kann geringer sein als beim Lösen des IP. Abbildung 5.7 gibt die relative Abweichung der LP-Lösung von der IP-Lösung an. Der Postprozessor-Schritt wurde dabei nicht durchgeführt.

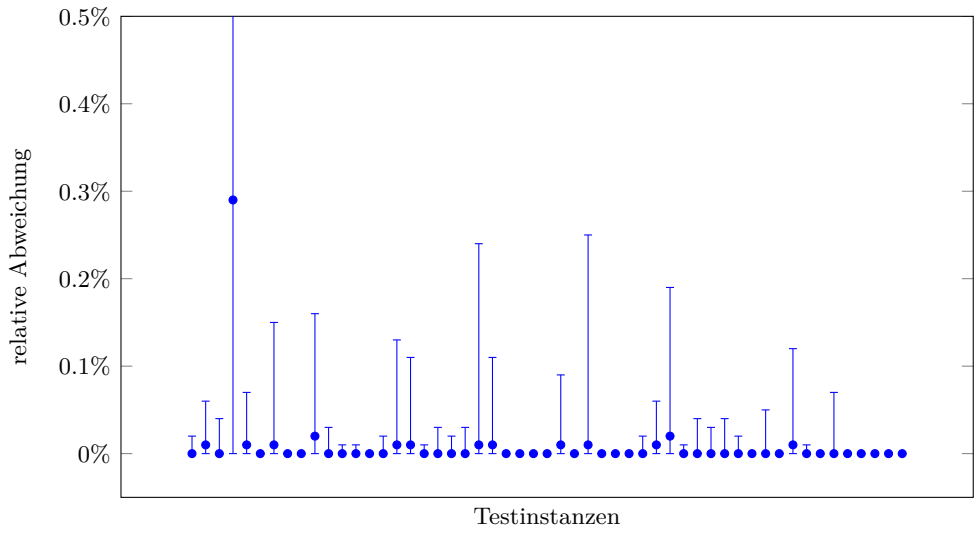


Abb. 5.7: Relative Abweichung der LP-Lösung von der IP-Lösung

Auf 95,08% der Instanzen besaßen LP und IP denselben Zielfunktionswert. Die maximale absolute Abweichung zwischen LP- und IP-Lösung lag bei 199 Längeneinheiten, die maximale relative Abweichung bei 5,72%. Es wird deutlich, dass auf den 2-zusammenhängenden Graphen, die als Testinstanzen gewählt wurden, nur selten fraktionale Lösungen auftraten. Sofern sie auftraten, war die relative Abweichung von der IP-Lösung meist gering.

In Abschnitt 4.4 wurde Voraussetzung (4.10) formuliert, die hinreichend dafür ist, dass die LP-Relaxierung von ($\star$) eine ganzzahlige Lösung besitzt. Voraussetzung (4.10) war auf 25,35% der Instanzen verletzt, eine fraktionale Lösung trat auf 3,50% der Instanzen auf. Dies zeigt, dass in Fällen, in denen Voraussetzung (4.10) verletzt ist, trotzdem häufig eine ganzzahlige Lösung auftritt.

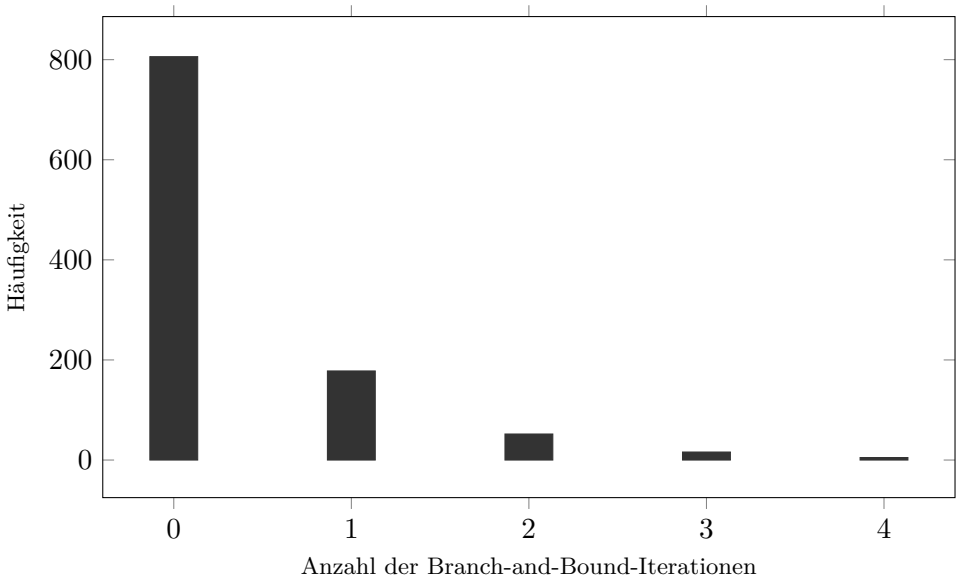


Abb. 5.8: Anzahl der Branch-and-Bound-Iterationen

Diese Beobachtung wird auch durch die Anzahl der benötigten Branch-and-Bound-Iterationen gestützt. Sieht man von der Instanz ab, auf der im Branch-and-Bound-Schritt das Zeitlimit erreicht wurde, mussten relativ wenige Branch-and-Bound-Iterationen durchgeführt werden. Abbildung 5.8 gibt die Anzahl der Branch-and-Bound-Iterationen im Kompaktierungsschritt des eigenen Ansatzes an.

Auf den 1.057 analysierten Instanzen wurden maximal vier Branch-and-Bound-Iterationen benötigt. Die Anzahl der Iterationen gibt dabei an, bis zu welcher Tiefe der Branch-and-Bound-Baum bearbeitet werden musste, d. h. für wie viele verschiedene Variablen Subprobleme gelöst wurden. Auf der überwiegenden Mehrzahl der Instanzen – 76,25% – war keine Branch-and-Bound-Iteration von Nöten, d. h. die LP-Relaxierung hat bereits eine ganzzahlige Lösung geliefert.

5.3.4 Analyse des Dissecting-Schrittes

Klau hatte 2001 festgestellt, dass die Anzahl der Dissecting-Kanten bei Anwendung des Turn-Regularity-Ansatzes signifikant niedriger ist als bei Anwendung der rechteckigen Dissecting-Methode (vgl. [Klau 2001, S. 110]).

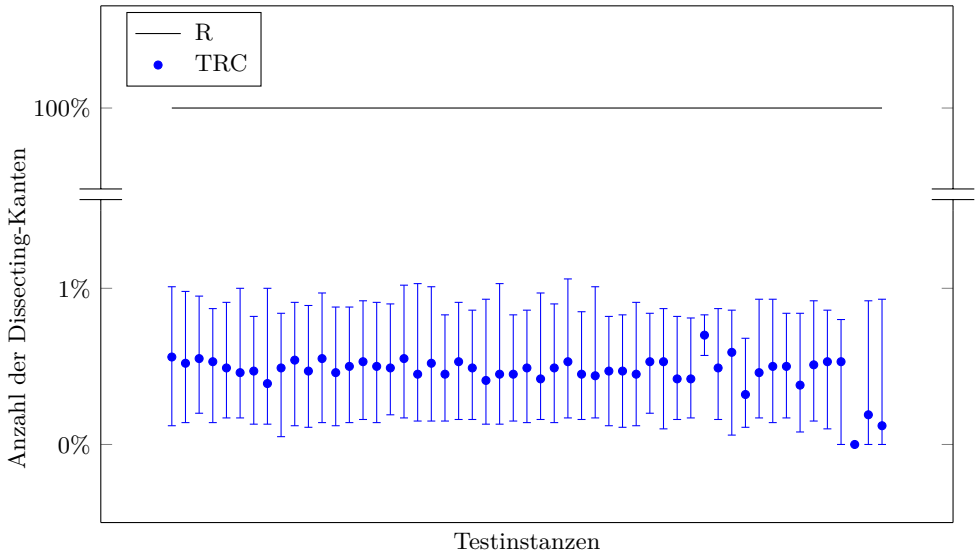


Abb. 5.9: Anzahl der Dissecting-Kanten

Diese Beobachtungen konnten in dieser Arbeit bestätigt werden: Die Anzahl der Dissecting-Kanten war bei Anwendung der rechteckigen Dissecting-Methode (R) durchschnittlich 213-mal so groß wie bei Anwendung des Turn-Regularity-Ansatzes (TRC). Abbildung 5.9 zeigt die Zahl der Dissecting-Kanten bei Anwendung der beiden Methoden.

Eine Ursache für die hohe Anzahl an Dissecting-Kanten bei der rechteckigen Methode ist, dass in der OGDF-Implementierung dieser Methode zunächst ein Expandierungsschritt durchgeführt wird: Alle Knoten v mit Grad $\deg(v) \leq 4$ werden so expandiert, dass Knoten mit Grad 2 entstehen, bei denen sich die inzidenten Kanten genau gegenüber liegen. Dieser Expandierungsschritt ist in der OGDF-Implementierung zwingend notwendig, um später die eindimensionalen Heuristiken anwenden zu können.

Würde man den Expandierungsschritt – rein hypothetisch – nicht durchführen, so würde die rechteckige Methode durchschnittlich 21-mal so viele Dissecting-Kanten benötigen wie der Turn-Regularity-Ansatz.

5.3.5 Analyse des Postprozessor-Schrittes

Im Postprozessor-Schritt wurden die Bedingungen $x_{a_h} + x_{a_v} \geq 1$ für Partnerkanten a_h, a_v fallen gelassen und die Schranken von a_h und a_v neu gewählt. Das Optimierungsproblem wurde anschließend erneut gelöst. Dadurch konnte die Gesamtkantenlänge häufig weiter reduziert werden, wie Abbildung 5.10 zeigt. Der Postprozessor-Schritt führte zu einer Verbesserung um bis zu 19,19%, im Durchschnitt um 2,89%. Dies ist darauf zurückzuführen, dass der Postprozessor-Schritt zusätzliche relative Positionen von Kitty Corners zueinander erlaubt, wie in Abschnitt 4.3 erläutert wurde.

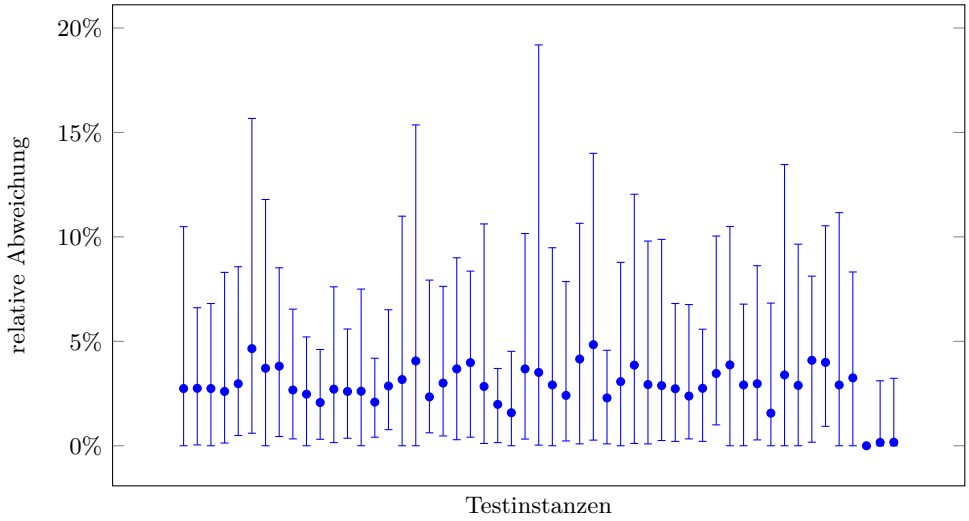


Abb. 5.10: Relative Verbesserung durch den Postprozessor-Schritt

Der Postprozessor-Schritt kann in einzelnen Fällen, die in Abschnitt 4.1.3 beschrieben wurden, zu unzulässigen Zeichnungen führen. Dann wurde die Postprozessor-Lösung verworfen und die letzte zulässige Lösung wiederhergestellt. Dieses Phänomen trat auf zwei der 1.057 Testinstanzen auf. In der überwiegenden Mehrzahl der untersuchten Fälle führte der Postprozessor-Schritt also zu zulässigen Lösungen.

6 Fazit

Ziel dieser Arbeit war die Kompaktierung orthogonaler Zeichnungen. Dabei sollten die Schwachpunkte bekannter eindimensionaler Heuristiken – ungünstige lokale Entscheidungen beim Einfügen der Dissecting-Kanten und fehlende Parallelität – umgangen werden. Es wurde ein zweidimensionaler Algorithmus für 2-zusammenhängende, 4-planare Graphen vorgestellt, bei dem das horizontale und das vertikale Subproblem als gemeinsames Minimum-Kosten-Fluss-Problem gelöst werden und bei dem die Ausrichtung der Dissecting-Kanten zunächst offen gelassen wird. Ob eine Dissecting-Kante horizontal oder vertikal verläuft, wird erst beim Lösen des entstehenden IP entschieden. Es wurde bewiesen, dass die Nebenbedingungsmatrix dieses IP unter gewissen Voraussetzungen total unimodular ist und die zugehörige LP-Relaxierung somit eine ganzzahlige Optimallösung besitzt.

In einer experimentellen Analyse wurde der eigene Ansatz mit bekannten Kompaktierungsheuristiken verglichen. Die Gesamtkantenlänge bei Anwendung des eigenen Ansatzes lag durchschnittlich 23,05% über der Gesamtkantenlänge bei Anwendung eindimensionaler Heuristiken. Auf 8,51% der 1.057 Testinstanzen lieferte der eigene Ansatz bessere Resultate als alle anderen Heuristiken. Die Laufzeit des eigenen Ansatzes lag deutlich unter der Laufzeit eindimensionaler Heuristiken, im Durchschnitt bei etwa 37,91%. Ebenfalls analysiert wurden die Abweichung der LP-Lösung von der IP-Lösung, die Anzahl der Dissecting-Kanten bei Anwendung verschiedener Heuristiken sowie die Verbesserung durch den Postprozessor-Schritt.

Es existieren verschiedene Ansatzpunkte, um den vorgestellten Algorithmus weiter zu verbessern: Der Algorithmus lässt sich auch für nicht-2-zusammenhängende Graphen implementieren, wenn Knoten zuvor geeignet expandiert werden. Lokale Entscheidungen konnten – falls sich mehrere Paare von Kitty Corners gegenseitig ausschließen – nicht vollständig vermieden werden. Da der eigene Ansatz eine vergleichsweise niedrige Laufzeit besitzt, könnte man in diesem Fall eine Divide-and-Conquer-Strategie anwenden, den Algorithmus mehrfach auf derselben Instanz ausführen und dabei unterschiedliche Kombinationen von Kitty Corners wählen.

Literaturverzeichnis

[Ahuja et al. 1993] Ahuja, R.; Magnanti, T.; Orlin, J.: *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, 1993

[Batini et al. 1986] Batini, C.; Nardelli, E.; Tamassia, R.: A Layout Algorithm for Data Flow Diagrams. In: *IEEE Transactions on Software Engineering* 12 (1986), Nr. 4, S. 538–546

[Brandes et al. 2013] Brandes, U.; Eiglsperger, M.; Lerner, J.; Pich, C.: Graph Markup Language (GraphML). In: Tamassia, R. (Hrsg.): *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC Press, 2013, S. 517–541

[Bridgeman et al. 2000] Bridgeman, S.; Di Battista, G.; Didimo, W.; Liotta, W.; Tamassia, R.; Vismara, L.: Turn-regularity and optimal area drawings of orthogonal representations. In: *Computational Geometry* 16 (2000), Nr. 1, S. 53–93

[Chimani et al. 2013] Chimani, M.; Gutwenger, C.; Jünger, M.; Klau, G.; Klein, K.; Mutzel, P.: The Open Graph Drawing Framework (OGDF). In: Tamassia, R. (Hrsg.): *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC Press, 2013, S. 543–569

[COIN 2012] *COIN: Computational Infrastructure for Operations Research*. `http://www.coin-or.org/` (Version COINAll-1.6.0, aufgerufen am 10.11.2013), 2012

[Cornelsen u. Karrenbauer 2012] Cornelsen, S.; Karrenbauer, A.: Accelerated Bend Minimization. In: *Proceedings of the 19th International Conference on Graph Drawing*. Springer-Verlag, Berlin, 2012 (GD '11), S. 111–122

[Dantzig 1963] Dantzig, G.: *Linear programming and extensions*. Princeton University Press, Princeton, 1963 (Rand Corporation Research Study)

[Di Battista et al. 1999] Di Battista, G.; Eades, P.; Tamassia, R.; Tollis, I.: *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Upper Saddle River, 1999

[Di Battista et al. 1997] Di Battista, G.; Garg, A.; Liotta, G.; Tamassia, R.; Tassinari, E.; Vargiu, F.: An experimental comparison of four graph drawing algorithms. In: *Computational Geometry* 7 (1997), Nr. 5-6, S. 303–325

[Diestel 2012] Diestel, R.: *Graduate texts in mathematics*. Bd. 173: *Graph Theory*. Springer, 2012

[Duncan u. Goodrich 2013] Duncan, C.; Goodrich, M.: Planar Orthogonal and Polyline Drawing Algorithms. In: Tamassia, R. (Hrsg.): *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC Press, 2013, S. 223–246

[Eiglsperger et al. 2001] Eiglsperger, M.; Fekete, S.; Klau, G.: Orthogonal Graph Drawing. In: Kaufmann, M. (Hrsg.); Wagner, D. (Hrsg.): *Drawing Graphs*. Springer-Verlag, London, 2001, S. 121–171

[Garey u. Johnson 1979] Garey, M.; Johnson, D.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979

[Garey u. Johnson 1983] Garey, M.; Johnson, D.: Crossing number is NP-complete. In: *SIAM Journal on Algebraic Discrete Methods* 4 (1983), Nr. 3, S. 312–316

[Garg u. Tamassia 1997] Garg, A.; Tamassia, R.: A New Minimum Cost Flow Algorithm with Applications to Graph Drawing. In: *Proceedings of the Symposium on Graph Drawing*. Springer-Verlag, London, 1997 (GD '96), S. 201–216

[Garg u. Tamassia 2002] Garg, A.; Tamassia, R.: On the Computational Complexity of Upward and Rectilinear Planarity Testing. In: *SIAM Journal on Computing* 31 (2002), Nr. 2, S. 601–625

[Grötschel 2013] Grötschel, M.: *Einführung in die Lineare und Kombinatorische Optimierung. Skriptum zur Vorlesung im WS 2012/2013*. `http://www.zib.de/groetschel/teaching/WS1213/Skriptum_ADM_I_130326.pdf` (aufgerufen am 01.03.2014), 2013

[Jünger u. Mutzel 2003] Jünger, M.; Mutzel, P.: Technical Foundations. In: Jünger, M. (Hrsg.); Mutzel, P. (Hrsg.): *Graph Drawing Software*. Springer-Verlag, Secaucus, 2003 (Mathematics and Visualization), S. 9–54

[Karmarkar 1984] Karmarkar, N.: A New Polynomial-time Algorithm for Linear Programming. In: *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*. ACM, New York, 1984 (STOC '84), S. 302–311

[Klau 2001] Klau, G.: *A Combinatorial Approach to Orthogonal Placement Problems*. Universität des Saarlandes, Saarbrücken, Dissertaton, 2001

[Klau u. Mutzel 1999] Klau, G.; Mutzel, P.: Optimal Compaction of Orthogonal Grid Drawings. In: *Proceedings of the 7th International IPCO Conference on Integer Programming and Combinatorial Optimization*. Springer-Verlag, London, 1999, S. 304–319

[OGDF 2012] *OGDF: The Open Graph Drawing Framework*. `http://www.ogdf.net/` (Version v.2012.07, aufgerufen am 01.10.2013), 2012

[Patrignani 2001] Patrignani, M.: On the Complexity of Orthogonal Compaction. In: *Computational Geometry: Theory and Applications* 19 (2001), Nr. 1, S. 47–67

[Patrignani 2013] Patrignani, M.: Planarity Testing and Embedding. In: Tamassia, R. (Hrsg.): *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC Press, 2013, S. 1–42

[ROME 1997] *Rome-Testsuite*. `https://www.ads.tuwien.ac.at/compaction/11000.tgz` (aufgerufen am 10.11.2013), 1997

[Spisla 2012] Spisla, C.: *Relaxierung der Knickminimalität zur Kompaktierung orthogonaler Zeichnungen*. Universität zu Köln, Diplomarbeit, 2012

[Tamassia 1987] Tamassia, R.: On Embedding a Graph in the Grid with the Minimum Number of Bends. In: *SIAM Journal on Computing* 16 (1987), Nr. 3, S. 421–444

[Tamassia et al. 1988] Tamassia, R.; Di Battista, G.; Batini, C.: Automatic Graph Drawing and Readability of Diagrams. In: *IEEE Transactions on Systems, Man and Cybernetics* 18 (1988), Nr. 1, S. 61–79

[Tamassia u. Tollis 1989] Tamassia, R.; Tollis, I.: Planar grid embedding in linear time. In: *IEEE Transactions on Circuits and Systems* 36 (1989), Nr. 9, S. 1230–1234

Abbildungsverzeichnis

Tabellenverzeichnis

Definitionsverzeichnis

Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit selbstständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht zu haben.

Köln, den 30. Juni 2014

(Alexander M. Esser)