

Optimal Crossing Minimization (Optimale Minimierung von Kantenkreuzungen)

Sebastian Weiß

Technische Universität München
sebastian13.weiss@tum.de

Abstract. In vielen Einsatzgebieten wird die Zeichnung eines Graphen mit möglichst wenigen Kantenkreuzungen benötigt. In dieser Arbeit gebe ich eine Einführung in das ordered-based-optimal-crossing-minimization-Verfahren (OOCM) ([CMB08]), welches dieses Problem löst. Obwohl das Problem \mathcal{NP} -vollständig ist, ist dieses Verfahren in der Lage, kleine und dünn besetzte Graphen mit bis zu 100 Knoten zu lösen.

1 Problem und Lösungsidee

Bei der Verlegung von Leiterbahnen auf Platinen ist es sinnvoll, möglichst wenig Kreuzungen zu haben. Jede Kreuzung benötigt dort eine Drahtverbindung auf der anderen Seite der Platine und ist somit mit erhöhten Platz und Kosten verbunden. Deshalb ist es wichtig, ein Verfahren zu finden, welches einen Graphen mit möglichst wenig Kantenkreuzungen zeichnet.

Das Problem ist jedoch \mathcal{NP} -vollständig, da bereits das Überprüfen, ob ein Graph mit einer gewissen Zahl von Kreuzungen gezeichnet werden kann, exponentielle Laufzeit benötigt (angenommen $\mathcal{P} \neq \mathcal{NP}$). In der Praxis benutzt man deshalb häufig Heuristiken. In dieser Arbeit wird dagegen ein Verfahren präsentiert, mit dem die Minimierung der Kantenkreuzungen optimal gelöst wird.

Einige kurze Definitionen sind für das Verständnis des Problems notwendig:

Definition 1. Sei G ein einfacher Graph. Dann ist $cr(G)$ die kleinste Anzahl an Kreuzungen, die beim Zeichnen des Graphen in einer Ebene notwendig sind. $cr(G)$ heißt Crossing Number des Graphen G .

Insbesondere ist $cr(G)=0$ für planare Graphen.

Definition 2 (Crossing Number Entscheidungsproblem). Sei $G=(V,E)$ ein einfacher Graph und sei $K \in \mathbb{N}$. Gilt $cr(G) \leq K$?

In [GJ83] wurde gezeigt, dass dieses Problem \mathcal{NP} -vollständig ist.

Um das Problem dennoch lösen zu können, werden Algorithmen benötigt, die die Crossing Number berechnen und gleichzeitig eine Lösung des Graphs mit dieser Zahl von Kreuzungen liefern. Ein Algorithmus, der dies erfüllt, wird hier vorgestellt.

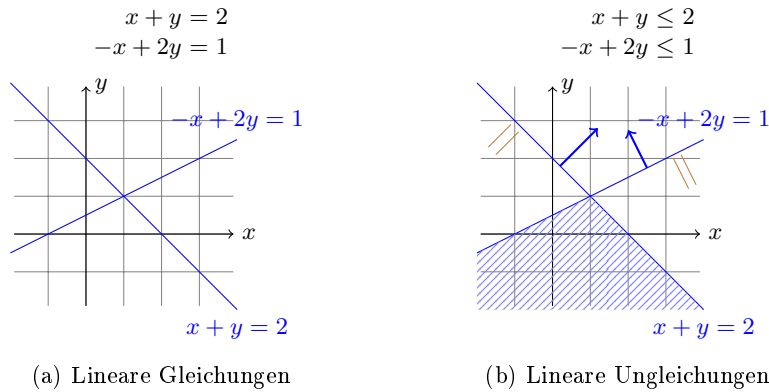


Fig. 1: Lineare Gleichungs- und Ungleichungssysteme

2 Lineare Programmierung

Für die Lösung des Optimierungsproblems müssen lineare Ungleichungssysteme optimiert werden. Diese Optimierung nennt man Lineare Programmierung (LP). In diesem Kapitel wird kurz in Lineare Programmierung eingeführt und ein Lösungsalgorithmus vorgestellt.

2.1 Lineare Ungleichungen

Bei linearen Gleichungssystemen sind die Gleichungen von der Form $Ax = b$ mit $A = (a_1, a_2, \dots, a_m)^T \in \mathbb{R}^{n \times m}$, $x \in \mathbb{R}^n$, $b = (\beta_1, \beta_2, \dots, \beta_m)^T \in \mathbb{R}^m$. Lineare Ungleichungssysteme in natürlicher Form sind dagegen von der Form

$$Ax \leq b \quad (1)$$

Jede Zeile $a_j^T x \leq \beta_j, j \in [m]$ definiert einen n -dimensionalen Halbraum. Dieser Halbraum ist auf einer Seite durch die Hyperebene $a_j^T x = \beta_j$ begrenzt und breitet sich auf der anderen Seite ins unendliche aus. Der Normalenvektor a_j steht senkrecht auf der Hyperebene und zeigt immer vom Halbraum weg (Abb. 1).

Jede Ungleichung kann in diese Form gebracht werden:

$$\begin{aligned} a^T x \geq \beta &\Rightarrow -a^T x \leq -\beta \\ a^T x = \beta &\Rightarrow a^T x \leq \beta \\ &\quad a^T x \geq \beta \end{aligned} \quad (2)$$

Der von den Halbräumen eingeschlossene Bereich wird **gültige Region** (engl. feasible region) genannt und ist als $\mathcal{P} = \{x \in \mathbb{R}^n : Ax \leq b\}$ definiert. Ist \mathcal{P} beschränkt, so nennt man \mathcal{P} ein **Polytop**. Im folgenden gehen wir immer von Polytopen aus. Die gültige Region ist immer konvex [Gri13].

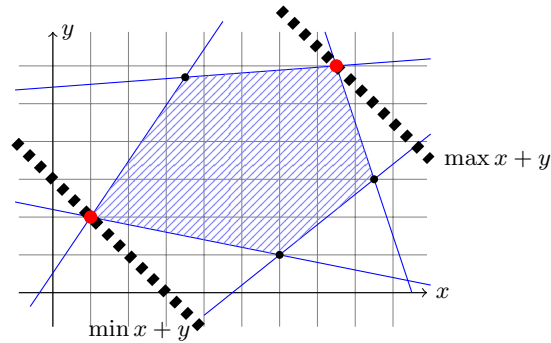


Fig. 2: gültige Region, Zielfunktion und Optimierer

2.2 Kostenfunktion und Optimum

Eine Funktion $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}$ definiert durch $\varphi(x) := c^T x$ mit $x, c \in \mathbb{R}^n$ heißt **Kostenfunktion**. Das Ziel der Linearen Programmierung ist nun, den Wert der Kostenfunktion zu maximieren (Maximierungsproblem) oder zu minimieren (Minimierungsproblem).

Ein **Lineares Optimierungsproblem** in natürlicher Form ist definiert als:

$$\begin{aligned} \max \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \end{aligned} \tag{3}$$

Analog zu linearen Ungleichungen können Maximierungsprobleme und Minimierungsprobleme ineinander umgewandelt werden: $\min c^T x \equiv -\max -c^T x$. Ein Punkt $x^* \in \mathcal{P}$ heißt **Optimierer** des LP falls $\neg \exists x \in \mathcal{P}, x \neq x^* \wedge \varphi(x) > \varphi(x^*)$. Also es gibt keinen anderen Punkt innerhalb der gültigen Region, der einen höheren Wert der Kostenfunktion besitzt.

Eine wichtige Eigenschaft ist, dass, falls die gültige Region nicht leer ist, es eine Ecke des Polytops gibt, welche optimal ist (siehe Abb. 2). Für einen Beweis siehe [Gri13].

Dadurch lässt sich die Suche nach einem Optimum auf die Suche nach einer optimalen Ecke einschränken. Allerdings ist das einfache Durchlaufen aller Ecken nicht praktikabel, da es davon exponentiell viele geben kann. Ein besserer Algorithmus muss also gewählt werden.

2.3 Simplex-Algorithmus

Die Idee des Simplex-Algorithmus in seiner geometrischen Form ist in Algorithmus 1 zusammengefasst.

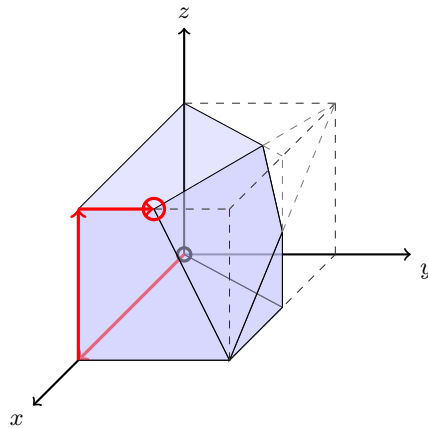


Fig. 3: Simplex-Algorithmus

Algorithmus 1 Simplex-Algorithmus

- 1: Starte bei einer beliebigen Ecke
 - 2: **loop**
 - 3: Berechne eine Kante an der aktuellen Ecke, die in sich in Richtung der Kostenfunktion verbessert
 - 4: **if** Kante gefunden **then**
 - 5: Gehe entlang dieser Kante bis eine neue Ecke gefunden wurde, diese Ecke ist die neue aktuelle Ecke
 - 6: **else**
 - 7: **return** optimale Ecke gefunden
 - 8: **end if**
 - 9: **end loop**
-

Der Simplex-Algorithmus nutzt dabei aus, dass Optimierer nur unter den Ecken gesucht werden müssen und das das Polytop konvex ist. Durch die Konvexitätseigenschaft ist garantiert, dass der Algorithmus nicht in einem lokalen Optimum verharret, sondern immer die globale Lösung findet.

Eine Anwendung des Simplex-Algorithmus ist in Abb. 3 zu sehen. Der schwarze Kreis markiert das Optimum, der rote Kreis ist die Startecke und die roten Kanten sind die einzelnen Schritte des Algorithmus.

2.4 Ganzzahlige Lineare Programmierung

Bisher sind alle Variablen reelle Zahlen. Um mit diesen Variablen aber Kantenkreuzungen zu realisieren, muss der Vektor der Variablen x aus $\{0, 1\}^n$ gewählt werden.

Wie in Abb. 4 zu sehen ist, kann aus einer fraktionellen Lösung nicht direkt auf eine diskrete Lösung geschlossen werden. Deshalb müssen alle Variablen,

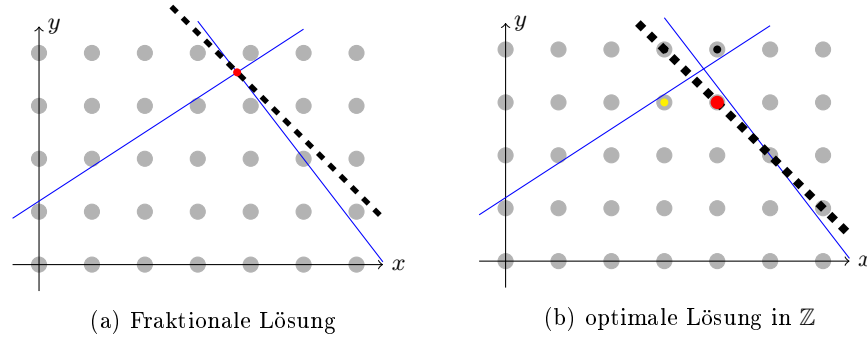


Fig. 4: Integer Linear Programming

die einen fraktionellen Anteil besitzen, gerundet werden. Alle Kombinationen von Auf- und Abrunden müssen durchgerechnet werden. Im Beispiel (Abb. 4) entstehen die beiden schwarz markierten Lösungen dadurch, dass einmal beide Variablen aufgerundet werden, und einmal die x-Variable abgerundet und die y-Variable aufgerundet wird. Diese beiden Lösungen liegen aber nicht mehr in der gültigen Region. Rundet man dagegen beide Variablen ab, landet man bei der Lösung markiert durch einen gelben Punkt. Diese Lösung liegt in der gültigen Region, ist aber nicht optimal. Lediglich wenn man x aufrundet und y abrundet erreicht man einen Optimierer dieses Optimierungsproblems.

Ein lineares Optimierungsproblem der Form

$$\begin{aligned} \max c^T x \\ Ax \leq b \\ x_i \in \mathbb{Z}, \forall i \in I \subseteq [n] \end{aligned} \quad (4)$$

nennt sich "Mixed Integer Linear Program" (MILP) falls $I \subset [n]$, bzw. "**Integer Linear Program**" (ILP) falls $I = [n]$.

Während die Lösung eines LP über den reellen Zahlen polynomielle Laufzeit besitzt, so liegt die Lösung eines MILP/ILP in \mathcal{NP} .

3 Lösung des Problems

Nun soll die Minimierung der Kantenkreuzungen des Graphen $G=(V,E)$ mithilfe von Integer Linearer Programmierung gelöst werden. Dazu werden folgende Variablen definiert:

$$x_{\{e,f\}} = \begin{cases} 1, \text{ falls Kante } e \text{ und Kante } f \text{ sich kreuzen} \\ 0, \text{ sonst} \end{cases}, \forall \{e,f\} \in \binom{E}{2} \quad (5)$$

Somit markiert jede x-Variable, die auf 1 gesetzt ist, eine Kreuzung zwischen zwei Kanten des Graphen (Abb. 5). Für jede x-Variable, die auf 1 gesetzt ist, wird ein Dummy-Knoten eingeführt, der genau diese Kreuzung markiert. Wie in der Abbildung zu erkennen ist, werden die zwei Kanten e und f gelöscht und vier neue Kanten von den vier angrenzenden Knoten zum neuen Dummy-Knoten eingefügt. Diesen Vorgang nennt man **Realisierung** des Graphen. Der entstandene Graph heißt **realisierter Graph**. Wurden genug Dummy-Knoten eingefügt, wird der realisierte Graph planar und kann mit einem normalen Algorithmus zum Zeichnen planarer Graphen gezeichnet werden.

Die Anzahl dieser Kreuzungen soll minimiert werden, also besitzt die Kostenfunktion die Form:

$$\min \sum_{\{e,f\} \in \binom{E}{2}} x_{\{e,f\}} \quad (6)$$

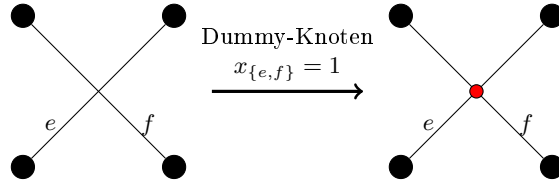


Fig. 5: x-Variablen und Dummy-Nodes

3.1 Realisierungsproblem

Leider reichen diese Variablen im Allgemeinen nicht aus. Es ist ohne weitere Einschränkung nicht möglich, allein aus den x-Variablen den Graphen in polynomialer Zeit zu rekonstruieren:





Definition 3 (Realisierungsproblem). Sei $G=(V, E)$ ein einfacher Graph und $x \in \{0,1\}^{\binom{E}{2}}$. Existiert eine Zeichnung von G , sodass sich genau dann die Kanten e und f kreuzen, wenn $x_{\{e,f\}} = 1$?

In [Kra91] wurde gezeigt, dass dieses Problem \mathcal{NP} -vollständig ist.

Der Grund dafür liegt in den sogenannten Mehrfachkreuzungen: Falls die Kante e von den Kanten f und g gleichzeitig gekreuzt wird, ist die Reihenfolge der Kreuzungen nicht klar. Diese Reihenfolge entscheidet aber, wie viele Kreuzungen insgesamt notwendig sind, da die Kanten f und g ja auch von anderen Kanten gekreuzt werden können.

3.2 Vergleich der Lösungsansätze

Zum Umgehen des Realisierungsproblems existieren zwei verschiedene Ansätze:

SOCM [Ebn05] <i>subdivision-based optimal crossing minimization</i>	OOCM [CMB08] <i>ordering-based optimal crossing minimization</i>
nur einfache Kreuzungen 	mehrfache Kantenkreuzungen 
Aufspalten von Kanten 	Zusätzliche Variablen für die Reihenfolge 
$\Rightarrow \Theta(E ^4)$ Variablen	$\Rightarrow \Theta(E ^3)$ Variablen

SOCM verbietet Mehrfachkreuzungen, dadurch wird die Realisierung trivial. Es kann direkt ein Dummy-Knoten eingefügt werden, ohne die Reihenfolge zu beachten. Um jedoch eine optimale Lösung zu erreichen, muss jede Kante aufgespalten werden: Jede Kante kann von $\Theta(|E|)$ -vielen anderen Kanten gekreuzt werden.

OOCM erlaubt diese Mehrfachkreuzungen, benötigt allerdings zusätzliche Variablen, die die Reihenfolge festlegen.

3.3 OOCM: Lineare Programmierung

Da die Lösung von ILP-Problemen (siehe 2.4) exponentielle Zeit bei der Berechnung in der Anzahl der Variablen benötigt, betrachten wir im Folgenden ausschließlich das **OOCM**-Modell. In diesem Kapitel wird dieses Modell nun im Detail erläutert.

Variablen OOCM benutzt folgende zwei Klassen von Variablen:

$$x_{\{e,f\}} = \begin{cases} 1, & \text{falls Kante } e \text{ und Kante } f \text{ sich kreuzen} \\ 0, & \text{sonst} \end{cases}, \forall \{e, f\} \in \binom{E}{2} \quad (7)$$

$$y_{e,f,g} = \begin{cases} 1, & \text{falls Kante } f \text{ Kante } e \text{ vor Kante } g \text{ schneidet} \\ 0, & \text{sonst} \end{cases}, \forall (e, f, g) \in E^3 \quad (8)$$

Die x -Variablen markieren weiter die Existenz einer Kantenkreuzung. Die y -Variablen bestimmen die Reihenfolge bei Mehrfachkreuzungen (Abb. 6).

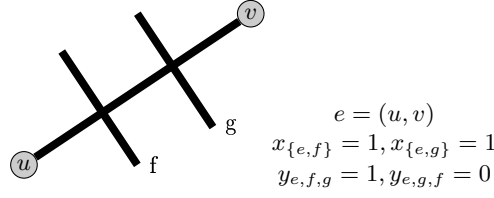


Fig. 6: x - und y -Variablen bei **OOCM**

Kostenfunktion Die Kostenfunktion ist unverändert:

$$\min \sum_{\{e,f\} \in \binom{E}{2}} x_{\{e,f\}} \quad (9)$$

Lineare Ordnung Die y -Variablen definieren eine Ordnung der Mehrfachkreuzungen im Sinne einer partiellen Ordnung. Folgende vier Klassen von Ungleichungen definieren die Funktion dieser Variablen:

$$x_{\{e,f\}} \geq y_{e,f,g}, \quad x_{\{e,g\}} \geq y_{e,f,g} \quad \forall (e,f,g) \in E^3 \quad (10)$$

$$1 + y_{e,f,g} + y_{e,g,f} \geq x_{\{e,f\}} + x_{\{e,g\}} \quad \forall (e,f,g) \in E^3 \quad (11)$$

$$y_{e,f,g} + y_{e,g,f} \leq 1 \quad \forall (e,f,g) \in E^3 \quad (12)$$

$$y_{e,f,g} + y_{e,g,h} + y_{e,h,f} \leq 2 \quad \forall (e,f,g,h) \in E^4 \quad (13)$$

Ungleichung 10 sagt aus, dass, falls eine Reihenfolge festgelegt ist, es, auch eine Mehrfachkreuzung geben soll. Ungleichung 11 beschreibt die andere Richtung: Falls eine Mehrfachkreuzung vorliegt, so muss auch die Reihenfolge festgelegt sein. Die Asymmetrie der Ordnung wird durch Ungleichung 12 gefordert, die Transitivität durch Ungleichung 13.

Damit ist eine partielle Ordnung der Kantenkreuzungen mithilfe von linearen Ungleichungen definiert.

Kuratowski-Constraints Bis jetzt können jedoch alle Variablen auf 0 gesetzt werden und eine optimale Lösung ist erreicht.

Um die Variablen zu bestimmen, die auf 1 gesetzt werden müssen, um den Graph planar zu machen, wird der Satz von Kuratowski [Kur30] benötigt:

Ein Graph ist genau dann nicht planar, wenn er eine Unterteilung des K_5 oder $K_{3,3}$ als Teilgraph enthält

Moderne Planaritätstester, wie der Algorithmus von Boyer und Myrvold [BM04], sind in der Lage in linearer Zeit einen Graph auf Planarität zu testen. Zusätzlich liefern sie bei nicht-planaren Graphen eine Unterteilung des K_5 oder des $K_{3,3}$ mit.

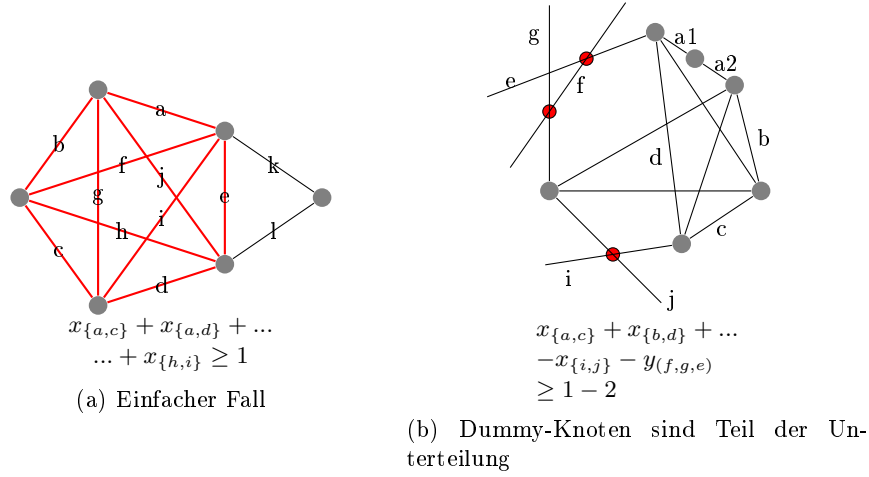


Fig. 7: Kuratowski-Constraints

Nun kann für jede solche Kuratowski-Unterteilung die Menge der Kantenkreuzungen in dieser Unterteilung bestimmt werden. Setzt man nun mindestens eine dieser Kantenkreuzung auf aktiv (die zugehörige x -Variable ist 1), so wird diese Kuratowski-Unterteilung planar (siehe Abb. 7a).

Leider ist kein polynomieller Algorithmus bekannt, der alle Kuratowski-Unterteilungen extrahiert. Deshalb müssen Ungleichungen nacheinander hinzugefügt werden, bis der Graph planar ist. Dabei kann es jedoch geschehen, dass in einer partiellen Lösung die Dummy-Knoten selbst Teil der Kuratowski-Unterteilung sind (siehe Abb. 7b). Zusätzlich sind diese Dummy-Knoten nur in dieser partiellen Lösung gesetzt, in der nächsten Lösung sind sie vielleicht nicht mehr existent. Die Variablen, die diese Dummy-Knoten erzeugen, werden im Folgenden aktive Variablen genannt.

Die Lösung bei Vorhandensein von Dummy-Knoten liegt in der Modellierung einer Bedingung. Dazu werden die aktiven x - und y -Variablen in der Ungleichung auf der linken Seite subtrahiert und auf der rechten Seite deren Anzahl ebenfalls subtrahiert. Falls nicht alle dieser x - und y -Variablen in einer neuen Lösung gesetzt sind, ist diese Ungleichung immer erfüllt. Nur falls diese Variablen gesetzt sind, definiert diese Ungleichung eine Kuratowski-Unterteilung.

Formal sind diese Ungleichungen wie folgt definiert: sei (\bar{x}, \bar{y}) die aktuelle partielle Lösung des Optimierungsproblems, sei $G[\bar{x}, \bar{y}]$ der realisierte Graph und sei $d : \binom{E}{2} \rightarrow V_{G[\bar{x}, \bar{y}]}$ eine Abbildung von den Kreuzungen zu dem dazugehörigen Dummy-Knoten (falls vorhanden). Sei $\mathcal{K} = (V_{\mathcal{K}}, E_{\mathcal{K}})$ die Knoten und Kanten der Kuratowski-Unterteilung in dieser partiellen Lösung.

Definiere $\mathcal{Z}_{\mathcal{K}}$ als die Menge der Kreuzungen eingeführt durch (\bar{x}, \bar{y}) , deren Dummy-

Knoten in V_K sind. D.h.

$$\mathcal{Z}_K[\bar{x}, \bar{y}] := \left\{ \{e, f\} \in \binom{E}{2} : \bar{x}_{\{e, f\}} = 1 \wedge d(\{e, f\}) \in V_K \right\}$$

Die Kreuzungen in \mathcal{Z}_K führen also genau die roten Dummy-Knoten in Abb. 7b ein.

Weiter definiere \mathcal{Y}_K als die minimale Beschreibung der Reihenfolgen.

$$\begin{aligned} \mathcal{Y}_K[\bar{x}, \bar{y}] := & \{ (e, f, g) \in E^3 : \{e, f\}, \{e, g\} \in \mathcal{Z}_K[\bar{x}, \bar{y}] \wedge \bar{y}_{e, f, g} = 1 \\ & \wedge \neg \exists (\{e, h\} \in \mathcal{Z}_K[\bar{x}, \bar{y}] : \bar{y}_{e, f, h} = 1 \wedge \bar{y}_{e, h, g} = 1) \} \end{aligned}$$

\mathcal{Y}_K enthält alle y -Variablen, die die Reihenfolge von Mehrfachkreuzungen in \mathcal{Z}_K festlegen. Zusätzlich werden die y -Variablen eliminiert, die durch die Transitivität der Ordnung automatisch aus den anderen y -Variablen abgeleitet werden.

In \mathcal{X}_K sind alle Kreuzungen aus \mathcal{Z}_K gelistet, die keine Mehrfachkreuzungen sind.

$$\begin{aligned} \mathcal{X}_K[\bar{x}, \bar{y}] := & \{ \{e, f\} \in \mathcal{Z}_K[\bar{x}, \bar{y}] : \bar{y}_{e, f, g} = 1 \\ & : (\forall g \in E : \{ (e, f, g), (e, g, f), (f, e, g), (f, g, e) \} \cap \mathcal{Y}_K[\bar{x}, \bar{y}] = \emptyset) \} \end{aligned}$$

Im Beispiel enthält \mathcal{X}_K somit die Kreuzung $\{i, j\}$; \mathcal{Y}_K enthält (f, g, e) .

Zuletzt werden in $\text{CrPairs}(K)$ alle Kreuzungen definiert, die die Kuratowski-Unterteilung planar machen können.

$$\begin{aligned} \text{CrPairs}(K) := & \left\{ \{e, f\} \in \binom{E_K}{2} : \bar{x}_{\{e, f\}} = 0 \right. \\ & \left. \wedge \text{Kante } e \text{ und Kante } f \text{ sind in } K \text{ nicht adjazent} \right\} \end{aligned}$$

Zwei Kanten sind in einer Kuratowski-Unterteilung adjazent, falls sie Teil der Unterteilung der gleichen Kante im gefundenen K_5 oder $K_{3,3}$ sind, oder diese Kanten benachbart sind.

Im Beispiel (Abb. 7b) enthält $\text{CrPairs}(K)$ z.B. die Kreuzungen $\{a1, c\}$, $\{a2, c\}$ und $\{b, d\}$, aber nicht $\{a1, a2\}$, da sie auf der gleichen Kante im K_5 liegen, oder $\{a1, b\}$, da sie im K_5 benachbart sind.

Mithilfe von $\text{CrPairs}(K)$, \mathcal{Y}_K und \mathcal{X}_K können nun die Kuratowski-Bedingungen formuliert werden.

$$\sum_{\{e, f\} \in \text{CrPairs}(K)} x_{\{e, f\}} - \sum_{a \in \mathcal{X}_K[\bar{x}, \bar{y}]} x_a - \sum_{b \in \mathcal{Y}_K[\bar{x}, \bar{y}]} y_b \geq 1 - |\mathcal{X}_K[\bar{x}, \bar{y}]| - |\mathcal{Y}_K[\bar{x}, \bar{y}]| \quad (14)$$

Zusammen ergibt sich folgendes ILP-Modell, mit dem die Minimierung der Kantenkreuzungen gelöst werden kann:

$$\min \left\{ \sum_{\{e,f\} \in \binom{E}{2}} x_{\{e,f\}} \text{ mit (10), (11), (12), (13) und allen (14)} \right\} \quad (15)$$

3.4 Implementierung

In der Implementierung wird das Optimierungsproblem nach folgendem Algorithmus (Alg. 2) gelöst:

Algorithmus 2 Lösung der Crossing Minimization

Input: Graph $G = (V, E)$

Output: realisierter Graph $G' = (V', E')$, Crossing Number cr

```

1: Erstelle leeres ILP-Modell
2: Erzeuge  $x$ - und  $y$ -Variablen und füge sie hinzu
3: Setze Kostenfunktion (9)
4: Füge alle LO-Bedingungen (10), (11), (12) und (13) hinzu
5: loop
6:   Löse das ILP-Modell:  $\bar{x}, \bar{y}, cr$  //  $cr$ : Wert der Kostenfunktion
7:   realisiere den Graphen  $G' \leftarrow G[\bar{x}, \bar{y}]$ 
8:   if Graph  $G'$  ist planar then
9:     return  $G', cr$  // Problem ist gelöst
10:  else
11:     $\mathcal{K} \leftarrow$  die gefundene Kuratowski-Unterteilung
12:    füge Kuratowski-Bedingung (14) zum LP-Modell hinzu
13:  end if
14: end loop
```

In meiner Implementierung habe ich als ILP-Solver `lp_solve`) und als Framework für Graphen das Open Graph Drawing Framework (OGDF) benutzt. OGDF stellt eine Implementierung des Planaritätstest von Boyer und Myrvold [BM04] bereit und auch Algorithmen zum Zeichnen planarer Graphen, die dann nach dem Lösungsvorgang benutzt werden.

4 Optimierung

Mit dem Algorithmus aus 3.4 lässt sich die Minimierung der Kantenkreuzungen beweisbar zur optimalen Lösung lösen. Das Verfahren benötigt jedoch exponentielle Zeit. Deshalb werden in diesem Kapitel ausgewählte Optimierungsideen vorgestellt, die die Lösung deutlich beschleunigen.

Das Ziel dieser Optimierungsschritte ist zum Einen die Anzahl der Variablen zu verringern, indem Knoten entfernt werden, und zum Anderen die Anzahl der Lösungsvorgänge des ILP-Modells zu reduzieren.

4.1 Graph verkleinern

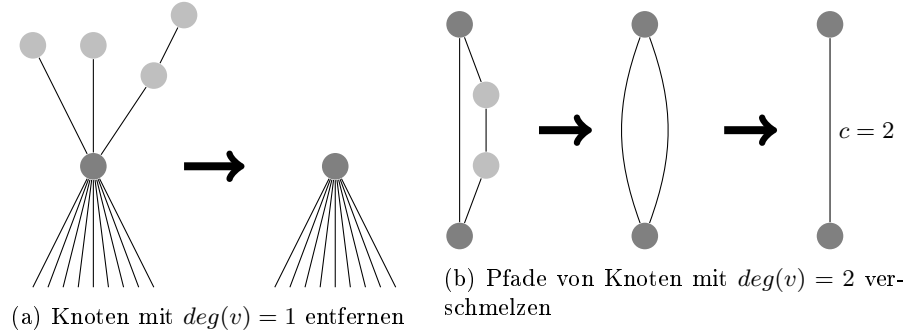


Fig. 8: Graph verkleinern

Knoten vom Grad 1 besitzen genau eine Kante zu dem benachbarten Knoten. Diese Knoten können offensichtlich immer ohne eine neue Kantenkreuzung eingefügt werden [Ebn05]. Somit können alle Knoten vom Grad 1 rekursiv aus dem Graphen entfernt werden (Abb. 8a).

Weiter bilden Knoten vom Grad 2 einen Pfad zwischen zwei Knoten mit größerem Grad. Die Kanten innerhalb dieses Pfades können auch immer ohne Kreuzungen untereinander gezeichnet werden. Daher werden solche Pfade zu einer langen Kante verschmolzen. Ein Problem entsteht, wenn sich dadurch Mehrfachkanten wie in Abb. 8b ergeben. Als Lösung werden solche Mehrfachkanten zu einer einfachen Kante reduziert und ein Gewicht $c = 2$ auf die Kante eingeführt. Dieses Gewicht zeigt an, dass bei der Umkehrung der Optimierung aus einer Kreuzung durch diese Kante zwei oder mehr Kreuzungen gemacht werden müssen.

Zusätzlich muss die Kostenfunktion dann noch angepasst werden:

$$\min \sum_{\{e,f\} \in \binom{E}{2}} c_e c_f x_{\{e,f\}} \quad (16)$$

4.2 Graph aufteilen

Man kann einen Graphen noch weiter in seine Biconnected Components aufteilen.

Definition 4 (Biconnected Graph). *Ein Graph $G=(V,E)$ heißt biconnected, falls man einen beliebigen Knoten entfernen kann und der Graph immer noch zusammenhängend bleibt.*

Definition 5 (Biconnected Components). *Die maximalen biconnected Teilgraphen von einem Graph $G=(V,E)$ nennt man Biconnected Component von G . Ein Knoten v in G heißt Artikulationsknoten, falls durch das Entfernen dieses Knotens G nicht mehr zusammenhängend ist.*

Remark 1. Jeder Knoten w in G , der kein Artikulationsknoten ist, gehört zu genau einem Biconnected Component. Jede Kante e in G gehört zu genau einem Biconnected Component.

Jeder Artikulationsknoten v gehört zu mindestens zwei Biconnected Components und trennt diese.

Um einen Graphen in Biconnected Components zu zerlegen, muss man also folglich die Artikulationsknoten finden. Abb. 9 zeigt die Zerlegung eines Graphen in seine Biconnected Components. Der schwarze quadratische Knoten ist ein Artikulationsknoten.

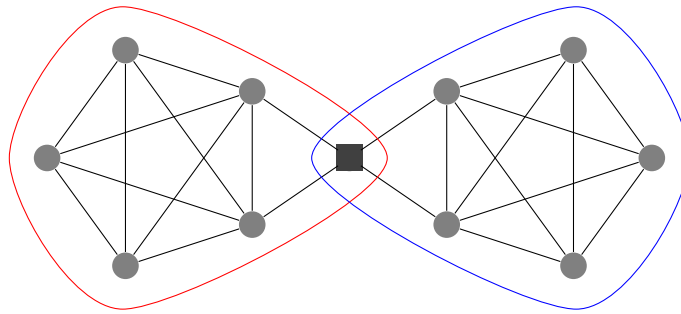


Fig. 9: Biconnected Components

Von Ebner ([Ebn05]) wurde gezeigt, dass zwischen den einzelnen Biconnected Components keine Kantenkreuzungen entstehen können. Dadurch kann man die einzelnen Komponenten getrennt lösen und spart sich somit viele Variablen. Danach werden die gelösten Teilgraphen wieder zusammengesetzt.

4.3 Mehrere Kuratowski-Constraints

Es ist möglich, den Planaritätstester von Boyer und Myrvold so zu erweitern, dass er in linearer Zeit mehrere Kuratowski-Unterteilungen auf einmal findet [MPJ07]. Alle dazugehörigen Kuratowski-Ungleichungen, wie in 3.3 vorgestellt, können dann auf einmal ins ILP-Modell eingefügt werden, und die Anzahl der Iterationen, verbunden mit dem Lösen des ILP-Modells, verringert sich (Fig 10).

5 Fazit

Zusammenfassend lässt sich mithilfe des hier vorgestellten **OOCM**-Verfahrens ein Graph mit optimal wenigen Kantenkreuzungen zeichnen. Der Algorithmus benötigt jedoch exponentiell viel Zeit und ist somit nur für relativ kleine und dünne Graphen geeignet (siehe Abb. 11). Für größere Graphen sind Heuristiken dagegen praktikabler.

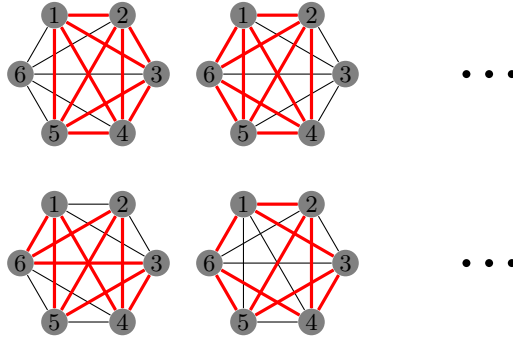


Fig. 10: Mehrere K_5 und $K_{3,3}$ im K_6

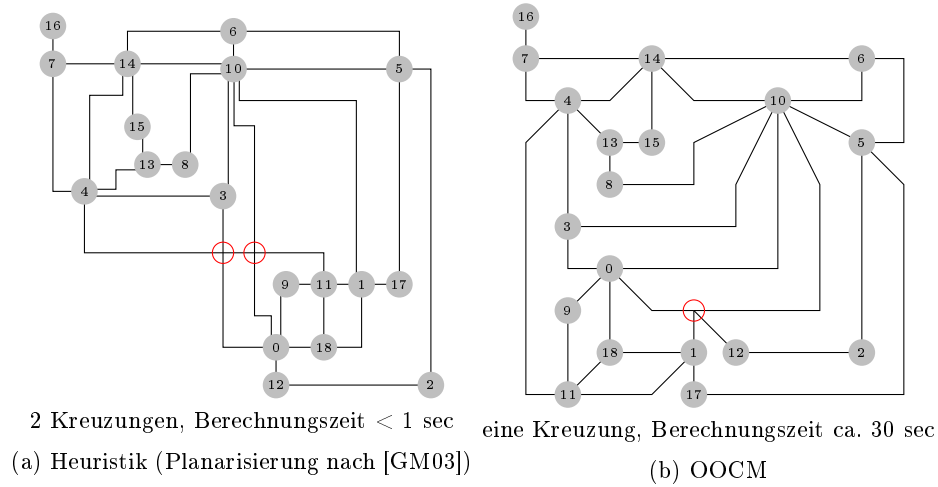


Fig. 11: Vergleich Heuristik und OOCM

References

- BM04. Boyer, J. M. and Myrvold, W. J. On the cutting edge: Simplified $O(n)$ planarity by edge addition. *Journal of Graph Algorithms and Applications*, (8(3)):241–273, 2004.
- CMB08. Markus Chimani, Petra Mutzel, and Immanuel Bomze. A new approach to exact crossing minimization. In Dan Halperin and Kurt Mehlhorn, editors, *Algorithms - ESA 2008*, volume 5193 of *Lecture Notes in Computer Science*, pages 284–296. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
- Ebn05. Dietmar Ebner. Optimal crossing minimization using integer linear programming. 2005.
- GJ83. Garey, M. R. and Johnson, D. S. Crossing number is np-complete. *SIAM Journal on Algebraic and Discrete Methods*, (4):312–316, 1983.

- GM03. Gutwenger, C. and Mutzel, P. An experimental study of crossing minimization heuristics. *11th International Symposium on Graph Drawing*, pages 13–24, 2003.
- Gri13. Gritzmann, P. *Grundlagen der mathematischen Optimierung*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- Kra91. Kratochvíl, J. String graphs. ii.: Recognizing string graphs is np-hard. *J. Comb. Theory Ser.*, (52(1)):67–78, 1991.
- Kur30. Kuratowski, Casimir. Sur le problème des courbes gauches en topologie. *Fund. Math.*, (15):271–283, 1930.
- MPJ07. Markus Chimani, Petra Mutzel, and Jens M. Schmidt. Efficient extraction of multiple kuratowski subdivisions (tr). 2007.