

Universität Konstanz

# Ein effizienter FPT-Algorithmus zur 1-seitigen Kreuzungsminimierung

**Silke Werz**

Seminar: Zeichnen von Graphen  
Prof. Ulrik Brandes

WS 2004/05

basierend auf dem Artikel von Vida Dujmović und Sue Whitesides. "An Efficient Fixed Parameter Tractable Algorithm for 1-Sided Crossing Minimization". *Proc. GD'02*, Springer LNCS 2528, pages: 118-129, 2002.

## Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>2</b>
<b>2</b>	<b>Problemstellung und Grundlagen</b>	<b>3</b>
2.1	Problemstellung . . . . .	3
2.2	Grundlagen . . . . .	3
<b>3</b>	<b>Der Algorithmus</b>	<b>8</b>
3.1	Das Prinzip . . . . .	8
3.2	Der Algorithmus . . . . .	9
3.3	Beispiel . . . . .	11
<b>4</b>	<b>Korrektheit und Laufzeit</b>	<b>15</b>
4.1	Korrektheit . . . . .	15
4.2	Laufzeit . . . . .	16
<b>5</b>	<b>FPT-Algorithmen</b>	<b>17</b>
<b>6</b>	<b>Zusammenfassung und Ausblick</b>	<b>18</b>

## 1 Einführung

Beim Zeichnen von Graphen trifft man auf verschiedene Arten von Layouts. Unter anderem gibt es Graphen, bei denen die Knoten auf Lagen angeordnet sind und die Kanten zwischen den Lagen verlaufen. Diese nennt man deshalb auch Lagen-Layouts. Dabei kann man ohne Einschränkung annehmen, dass die Kanten nur zwischen Knoten von benachbarten Lagen existieren. Dies kann durch Einführung von sog. Dummy-Knoten erreicht werden.

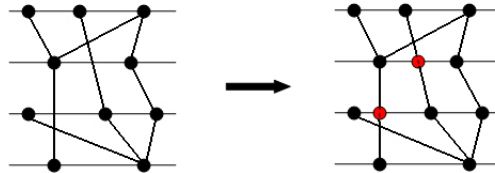


Abbildung 1: Einführung von Dummy-Knoten (rot).

Um solche Lagen-Layouts nun möglichst lesbar darzustellen, versucht man die Anzahl der Kantenkreuzungen zu minimieren. Diese Zahl hängt also von der Anordnung der Knoten innerhalb einer Lage ab. Leider ist dieses Problem  $NP$ -vollständig [2]. Weshalb man auch versucht, es auf ein kleineres Problem zu reduzieren. Man beginnt nun bei der untersten Lage und legt für deren Knoten eine Ordnung fest. Dann ordnet man die Knoten der darauffolgenden Lage so an, dass die Kantenkreuzungszahl minimal wird. Dies führt man sukzessive weiter, bis für alle Lagen eine Ordnung definiert wurde. Betrachtet man das Problem nun auf diese Weise, so kann man es reduzieren auf die 1-seitige Kreuzungsminimierung. Dabei wird ein Graph mit 2 Lagen betrachtet, wobei in einer Lage die Anordnung der Knoten fest ist. Doch auch dieses reduzierte Problem ist immer noch  $NP$ -vollständig [7].

Nun gibt es verschiedene Ansätze zur Lösung. Eine Möglichkeit ist es, Heuristiken dafür anzugeben (z.B. [3], [7]). Das Problem von Heuristiken ist jedoch, dass man für die Qualität der Lösung nur sog. Gütegarantien angeben kann. Deshalb wird hier ein Algorithmus vorgestellt, der auf einer anderen Methode beruht. Der Algorithmus gehört zu den "fixed parameter tractable"-Algorithmen, bzw. zur Klasse FPT. Diese Algorithmen zeichnen sich dadurch aus, dass für einen festen Parameter  $k$ , der hier die Anzahl Kantenkreuzungen angibt, das Problem in  $f(k) \cdot n^\alpha$  gelöst wird. Wobei  $\alpha$  eine Konstante ist und  $n$  die Anzahl der Knoten. Dies trifft auch hier zu, da die Laufzeit der 1-seitigen Kreuzungsminimierung  $O(\phi^k \cdot n^2)$  ist, mit  $\phi = \frac{1+\sqrt{5}}{2}$ .

In Kapitel 2 wird das Problem formuliert und es werden Grundlagen und Ergebnisse vorgestellt, die für den Algorithmus wichtig sind. Im Folgenden wird dann der Algorithmus detailliert angegeben, sowie ein Beweis für dessen Korrektheit und Laufzeit. Abschließend werde ich noch kurz etwas genauer auf FPT-Algorithmen eingehen.

## 2 Problemstellung und Grundlagen

### 2.1 Problemstellung

*Problem:* 1-SEITIGE KREUZUNGSMINIMIERUNG  $\langle G, \pi_1, k \rangle$

*Gegeben:* Ein bipartiter Graph  $G = (V, E)$  mit  $V = L_1 \cup L_2$  und  $E = L_1 \times L_2$ . Dabei sind  $L_1$  und  $L_2$  disjunkte Knotenmengen, die auf zwei Lagen angeordnet sind. Weiterhin ist eine feste Ordnung  $\pi_1$  für die Knotenmenge  $L_1$  gegeben und ein  $k \in \mathbb{N}$ .

*Frage:* Existiert ein 2-Lagen-Layout  $(G, \pi_1, \pi_2)$ , welches die Ordnung  $\pi_1$  respektiert und höchstens  $k$  Kantenkreuzungen hat?

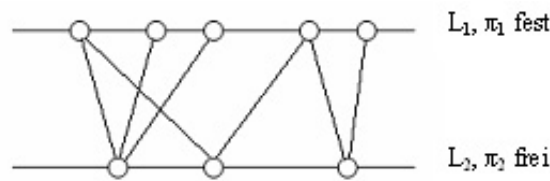


Abbildung 2: Ein 2-Lagen-Layout mit fester Ordnung  $\pi_1$  auf  $L_1$  und freier Lage  $L_2$ .

### 2.2 Grundlagen

Von nun an ist  $G$  immer ein bipartiter Graph mit fester Ordnung  $\pi_1$  auf der oberen Lage  $L_1$ . Ziel ist es, die Anzahl Kantenkreuzungen zu reduzieren.

#### Definition 1:

1. Für ein 2-Lagen-Layout  $(G, \pi_1, \pi_2)$  gibt  $cr(G, \pi_1, \pi_2)$  die Anzahl Kantenkreuzungen an.
2. Die minimale Anzahl Kreuzungen wird durch  $opt(G, \pi_1) = \min_{\pi_2} \{cr(G, \pi_1, \pi_2)\}$  ausgedrückt.
3. Für ein Problem  $\langle G, \pi_1, k \rangle$  und zwei Knoten  $v, w \in L_2$  gibt die Kreuzungszahl  $c_{vw}$  an, wie viele Kreuzungen die Kanten, die inzident zu  $v$  sind, mit Kanten, die inzident zu  $w$  sind, in einer Zeichnung mit  $v < w$  erzeugen. Die Kreuzungszahl  $c_{wv}$  ist für  $w < v$ .

Die folgende Behauptung trifft Aussagen über eine untere und obere Schranke für die Anzahl Kreuzungen, wenn man die Knotenpaare einzeln betrachtet.

**Behauptung 1:** [6] Die Gesamtanzahl von Kreuzungen in einem 2-Lagen-Layout  $(G, \pi_1, \pi_2)$  ist:

$$cr(G, \pi_1, \pi_2) = \sum_{\forall v < w \in \pi_2} c_{vw}, \quad (1)$$

die Summe geht hier über alle geordneten Paare  $v < w$  von  $\pi_2$  weiterhin gilt:

$$\sum_{v,w \in L_2} \min(c_{vw}, c_{wv}) \leq \text{opt}(G, \pi_1) \leq \sum_{v,w \in L_2} \max(c_{vw}, c_{wv}), \quad (2)$$

die Summe geht hier über alle ungeordneten Paare  $v < w$  von  $L_2$ .

Dies waren einige grundlegende Notationen. Jetzt möchten wir noch ein paar Eigenschaften von optimalen Zeichnungen, d.h. Zeichnungen mit der minimalen Anzahl an Kreuzungen, betrachten. Da der Algorithmus eine optimale Zeichnung erzeugt, ist es wichtig, vorher deren Eigenschaften zu kennen.

**Definition 2:**

1. Für einen Knoten  $v$  in  $L_2$  ist die Nachbarschaft von  $v$  durch alle Knoten aus  $L_1$  gegeben, die eine Kante mit  $v$  haben. Dabei ist  $l_v$  der linkeste Nachbar und  $r_v$  der rechteste Nachbar von  $v$ .
2. Wir betrachten nun zwei Knoten  $v$  und  $w$  in  $L_2$ . Dann gilt:  
 $v$  und  $w$  sind ein geeignetes Paar, falls  $r_v \leq l_w$  oder  $r_w \leq l_v$ .
3. Falls  $l_v = r_v = l_w = r_w$ , so sind  $v$  und  $w$  ein trivial geeignetes Paar.

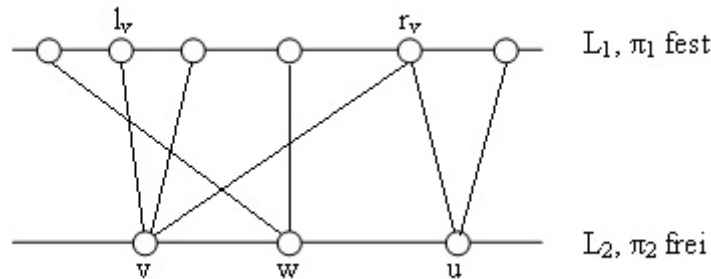


Abbildung 3: Das Paar  $v, w$  ist ungeeignet, aber  $v, u$  und  $w, u$  sind geeignete Paare.

Die folgende Behauptung zeigt, wie wichtig geeignete Paare für optimale Layouts sind.

**Behauptung 2:** Ein Knotenpaar  $v, w \in L_2$  ist genau dann ungeeignet, wenn  $c_{vw} \geq 1$  und  $c_{wv} \geq 1$ .

Für geeignete Paare ist es also möglich sie so zu ordnen, dass sie keine Kreuzungen erzeugen. Dies ist dann der Fall, wenn für  $v < w$  gilt  $r_v \leq l_w$  bzw. wenn für  $w < v$  gilt  $r_w \leq l_v$ . Dies führt zu folgender Definition.

**Definition 3:** Ist ein geeignetes Paar  $v, w$  so geordnet, dass es keine Kreuzungen erzeugt, so liegt es in seiner natürlichen Ordnung vor.

Der Zusammenhang von geeigneten Paaren und natürlicher Ordnung liefert einige Folgerungen für das Layout des Graphen.

**Behauptung 3:** Gegeben sei ein geeignetes Paar  $v, w$  mit natürlicher Ordnung  $v < w$ . Dann gilt für jede Ordnung  $\pi_2$  in  $(G, \pi_1, \pi_2)$ :

- (i)  $c_{vw} = 0$
- (ii) falls  $r_v \neq l_w$ , dann ist  $c_{wv} = d_v \cdot d_w$  wobei  $d_v$  der Grad von  $v$  ist.
- (iii) falls  $r_v = l_w$ , dann ist  $c_{wv} = (d_v \cdot d_w) - 1$
- (iv) Wenn  $v$  und  $w$  nicht trivial geeignet sind, so ist  $c_{wv} > 0$

Das folgende Lemma ist Basis für unseren Algorithmus.

**Lemma 1:** Für festes  $\pi_1$  sei  $\Gamma_{opt} = (G, \pi_1, \pi_{opt})$  eine Zeichnung mit der minimalen Anzahl an Kreuzungen. Dann treten alle geeigneten Paare in  $\pi_{opt}$  in ihrer natürlichen Ordnung auf.

Für den Beweis von Lemma 1 wird Lemma 2 benötigt.

**Lemma 2:** Sei  $(G, \pi_1, \pi_2)$  eine Zeichnung mit fester Ordnung  $\pi_1$ . Für  $1 \leq i \leq |L_2|$ , sei  $v_i$  der  $i$ -te Knoten in  $\pi_2$ . Verschiebt man nun den Knoten  $v_i \in L_2$  um  $t$  Stellen, über  $v_{i+1}, v_{i+2}, \dots, v_{i+t}$ , nach rechts, so entsteht eine neue Zeichnung  $(G, \pi_1, \pi'_2)$  mit:

$$cr(G, \pi_1, \pi'_2) = cr(G, \pi_1, \pi_2) + \sum_{j=1}^t (c_{v_{i+j}v_i} - c_{v_i v_{i+j}}). \quad (3)$$

Ähnlich ist es, wenn  $v_i$  nach links verschoben wird. Dann sieht die Gleichung wie folgt aus:

$$cr(G, \pi_1, \pi'_2) = cr(G, \pi_1, \pi_2) - \sum_{j=-1}^{-t} (c_{v_{i+j}v_i} - c_{v_i v_{i+j}}). \quad (4)$$

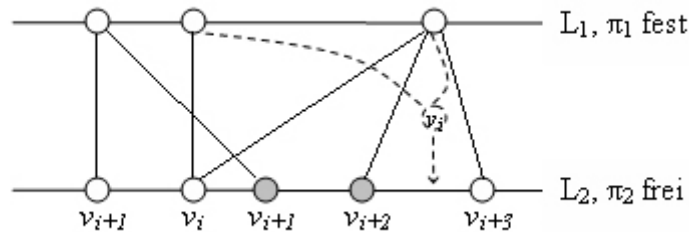


Abbildung 4: Illustration für Beweis von Lemma 2.

**Beweis** (zu Lemma 2): o.B.d.A. wird  $v_i$  in  $(G, \pi_1, \pi_2)$  um  $t$  Stellen nach rechts verschoben. Somit entsteht eine neue Zeichnung  $(G, \pi_1, \pi'_2)$ . Die einzigen Knotenpaare, deren relative Ordnung sich ändert, sind die Paare  $v_i, v_j$  für  $i+1 \leq j \leq i+t$  (siehe Abb. 4). Dies sind auch die einzigsten Paare, für die sich die Kreuzungszahl ändern kann. Genauer ausgedrückt, ändert sich die Kreuzungszahl von  $v_i, v_j$  für  $j \in [i+1, i+t]$  von  $c_{v_i v_j}$  zu  $c_{v_j v_i}$ . Ersetzt man diese Änderungen nun in Gleichung (1) von Behauptung 1, so ergibt sich obige Gleichung (3) bzw. (4).  $\square$

**Beweis** (von Lemma 1):

*Annahme:*  $\Gamma_{opt}$  enthält ein geeignetes Paar  $v, w$ , dessen Ordnung in  $\pi_{opt}$  nicht die natürliche Ordnung ist. Ist o.B.d.A.  $v < w$  die natürliche Ordnung, so gilt in  $\pi_{opt}$   $w < v$ .

Wegen Behauptung 3 gilt somit:  $c_{vw} = 0$  und  $c_{wv} > 0$ .

Wir wollen nun einen Widerspruch herbeiführen, indem man  $v$  oder  $w$  so verschiebt, dass eine bessere Zeichnung  $\Gamma_{new}$  entsteht, mit  $cr(\Gamma_{new}) < cr(\Gamma_{opt})$ .

Sei  $i$  die Position von  $w$  und  $j$  die Position von  $v$ . Wegen  $w < v$  in  $\pi_{opt}$  gilt deshalb  $i < j$ .

1. Fall:  $|j - i| = 1$

d.h.  $v$  und  $w$  sind benachbarte Knoten. Deshalb kann man  $v$  und  $w$  vertauschen, ohne andere Knotenpaare zu beeinträchtigen. Vertauscht man also  $v$  und  $w$ , so erhält man mit Gleichung (3) bzw. (4) aus Lemma 2:

$$cr(\Gamma_{new}) = cr(\Gamma_{opt}) - \underbrace{c_{wv}}_{>0} + \underbrace{c_{vw}}_{=0} \implies cr(\Gamma_{new}) < cr(\Gamma_{opt}).$$

Dies aber ist ein Widerspruch dazu, dass  $\Gamma_{opt}$  optimal ist.

2. Fall:  $|j - i| > 1$

Seien  $u_{i+1}, u_{i+2}, \dots, u_{j-1}$  die Knoten zwischen  $w$  und  $v$  in  $\pi_{opt}$ . Betrachte  $u_{i+1}, \dots, u_{j-1}$  als festen Block  $U$ , der innerhalb nicht verändert wird (siehe Abbildung 4).

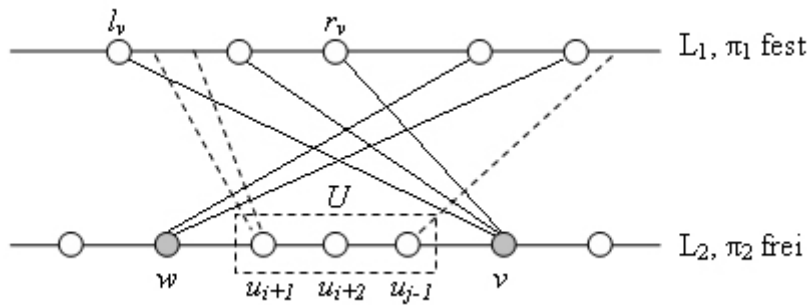


Abbildung 5:  $\Gamma_{opt}$  für den Fall  $|j - i| > 1$

Verschiebt man nun  $v$  oder  $w$  von einer Seite des Blocks  $U$  auf die andere Seite, so sind nur Kreuzungszahlen von Paaren  $u, w$  und  $u, v$  für  $u \in U$  betroffen.

Definiere  $c_{Up} := \sum_{u \in U} c_{up}$ , wobei  $p$  rechts von  $U$  liegt und  $c_{pU} := \sum_{u \in U} c_{pu}$ , wobei  $p$  links von  $U$  liegt.

Da  $\Gamma_{opt}$  optimal ist, muss gelten:

$$c_{Uv} < c_{vU} \quad (5)$$

Andernfalls kann man  $v$  auf die linke Seite von  $U$  schieben und mit  $w$  vertauschen, was zu einer neuen Kreuzungszahl führt:  $cr(\Gamma_{new}) = cr(\Gamma_{opt}) - c_{Uv} + c_{vU} - c_{wv} + c_{vw}$ . Mit  $c_{wv} > 0$  und  $c_{vw} = 0$  gilt: falls  $c_{Uv} \geq c_{vU} \Rightarrow cr(\Gamma_{new}) < cr(\Gamma_{opt})$ , was ein Widerspruch zur Optimalität von  $\Gamma_{opt}$  ist. Also gilt  $c_{Uv} < c_{vU}$ .

Als nächstes zeigen wir, dass aus  $c_{Uv} < c_{vU}$  die Ungleichung  $c_{wU} > c_{Uw}$  folgt. Folgende Definitionen sind nötig:

$$\begin{aligned} E_R &= \{(u, s) \in \pi_{opt} \mid u \in U, s > r_v \text{ in } \pi_1\} \\ E_L &= \{(u, s) \in \pi_{opt} \mid u \in U, s < r_v \text{ in } \pi_1\} \\ N_v &= \text{Nachbarn von } v \\ N_w &= \text{Nachbarn von } w \end{aligned}$$

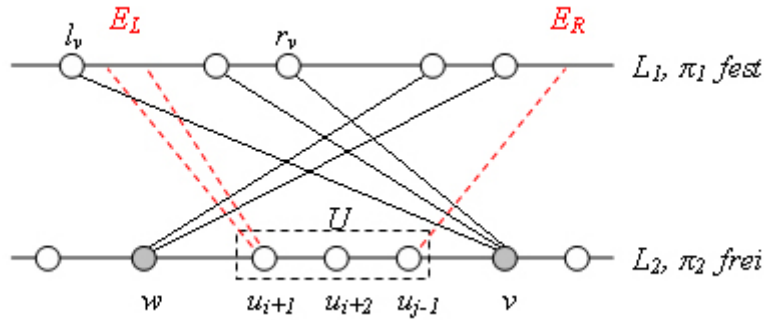


Abbildung 6: Position von  $E_R$  und  $E_L$  in diesem Graph

Wegen der Definition von  $E_R$  liegen in  $\pi_1$  alle  $s \in N_v$  vor allen Endknoten aus  $E_R$ . Wegen der Definition von  $E_L$  und weil  $v, w$  ein geeignetes Paar mit natürlicher Ordnung  $v < w$  ist, liegen in  $\pi_1$  alle  $s \in N_w$  hinter allen Endknoten von  $E_L$ . Damit kann man nun folgende Ungleichungen für die Kreuzungszahlen bestimmen:

$c_{Uv} \geq d_v \cdot |E_R|$ : Die Kanten, die zu  $v$  inzident sind und die Kanten aus  $E_R$  kreuzen sich paarweise. Dabei werden  $d_v \cdot |E_R|$  Kreuzungen erzeugt. Da  $E_R$  eine Teilmenge der Kanten ist, die zu  $U$  inzident sind, gilt  $c_{Uv} \geq d_v \cdot |E_R|$ .

$c_{vU} \leq d_v \cdot |E_L|$ : Diese Ungleichung gilt, weil keine Kante, die inzident zu  $v$  ist, sich mit keiner Kante kreuzt, die inzident zu  $U$  ist und die nicht in  $E_L$  liegt.

$c_{wU} \geq d_w \cdot |E_L|$ : Die Kanten, die zu  $w$  inzident sind und die Kanten aus  $E_L$  kreuzen sich paarweise, so dass  $c_{wU} \geq d_w \cdot |E_L|$ .

$c_{Uw} \leq d_w \cdot |E_R|$ : Diese Ungleichung gilt, weil keine Kante, die inzident zu  $w$  ist, sich mit keiner Kante kreuzt, die inzident zu  $U$  ist und die nicht in  $E_R$  ist.

Nach diesen Vorbereitungen kann man nun zeigen, dass aus  $c_{Uv} < c_{vU}$  (5) die Ungleichung  $c_{wU} > c_{Uw}$  folgt:

$$d_v \cdot |E_R| \leq c_{Uv} \stackrel{\text{Ungl. (5)}}{<} c_{vU} \leq d_v \cdot |E_L| \Rightarrow |E_R| < |E_L|$$

mit dieser Abschätzung wiederum gilt dann:

$$c_{wU} \geq d_w \cdot |E_L| > d_w \cdot |E_R| \geq c_{Uw} \Rightarrow c_{wU} > c_{Uw}$$

Bis jetzt haben wir also für den Fall  $|j - i| > 1$  gezeigt, dass auf jeden Fall  $c_{Uv} < c_{vU}$  gelten muss. Daraus wiederum folgt aber  $c_{wU} > c_{Uw}$ . Dann kann jedoch  $w$  auf die rechte Seite von  $U$  verschoben werden, ohne die Gesamtanzahl an Kreuzungen in der Zeichnung zu erhöhen. Vertauscht man nun noch  $w$  und  $v$ , so erhält man eine Zeichnung, die weniger Kreuzungen hat als  $\Gamma_{opt}$ . Das ergibt einen Widerspruch.

Deshalb kann die Annahme, dass  $\Gamma_{opt}$  ein geeignetes Paar enthält, welches nicht in natürlicher Ordnung vorliegt, nicht stimmen. Somit ist Lemma 1 bewiesen.  $\square$

## 3 Der Algorithmus

### 3.1 Das Prinzip

Bei diesem Algorithmus wird ein Suchbaum aufgebaut, der auf den ungeeigneten Knotenpaaren basiert. Das Ziel ist es, eine Ordnung  $\pi_2$  auf den Knoten in  $L_2$  zu definieren. Dazu wird ein Suchbaum wie folgt aufgebaut: Die Knoten des Suchbaumes werden mit  $(D, B)$  beschriftet, wobei  $D$  ein transitiv abgeschlossener, gerichteter, azyklischer Graph ist, der die Ordnung auf den Knotenpaaren von  $L_2$  speichert, und  $B$  ein Budget für die Anzahl Kantenkreuzungen angibt. Ein Baumknoten  $(D, B)$  verzweigt sich in zwei Teilprobleme, indem man für ein ungeordnetes Knotenpaar  $(v, w)$  eine Ordnung  $v < w$  bzw.  $w < v$  festlegt. Für diese festgelegte Ordnung  $v < w$  wird in den Graph  $D$  eine gerichtete Kante von  $v$  nach  $w$  eingefügt. Danach wird das nächste ungeeignete Knotenpaar betrachtet. Es wird also für jeden Baumknoten der Graph  $D$  so verändert, dass immer eine gerichtete Kante  $vw$  in den Graph des Elternknotens eingefügt wird. So wächst der Graph  $D$  sukzessiv an, indem die Knotenpaare aus  $L_2$  geordnet werden. Das Budget wird bei jedem Baumknoten um die Anzahl Kantenkreuzungen reduziert, die durch das Ordnen des Paares  $(v, w)$  in  $G$  entstehen.

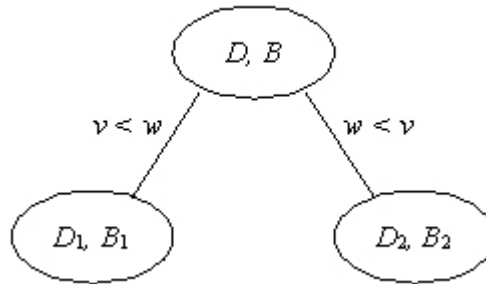


Abbildung 7: Verzweigung an einem Baumknoten

### 3.2 Der Algorithmus

Der Ablauf des Algorithmus ist in 3.3 durch ein Beispiel verdeutlicht. Das Beispiel soll also zum besseren Verständnis des Algorithmus beitragen.

*Algorithmus:* 1-SEITIGE KREUZUNGSMINIMIERUNG

*Eingabe:*  $\langle G, \pi_1, k \rangle$

*Ausgabe:*  $\pi_{opt}$ , falls  $\langle G, \pi_1, k \rangle$  eine JA-Instanz ist, NEIN sonst.

#### Schritt 0: Berechnung der Kreuzungszahlen

Berechne  $c_{vw}$  und  $c_{wv}$  für alle Knotenpaare  $(v, w)$  in  $L_2$ .

Die Berechnung einzelner Kreuzungszahlen wird abgebrochen, falls  $c_{vw}$  bzw.  $c_{wv} > k$ .

#### Schritt 1: Überprüfung auf Extremwerte

Vergleiche  $k$  mit der unteren und oberen Schranke, wie in Gleichung (2) aus Behauptung 2.

falls  $k < \sum_{(v,w)} \min(c_{vw}, c_{wv})$ , gib NEIN aus.

falls  $k \geq \sum_{(v,w)} \max(c_{vw}, c_{wv})$ , gib eine beliebige Reihenfolge für  $\pi_2$  aus.

#### Schritt 2: Initialisierung

In der Initialisierung werden vorab einige Berechnungen vorgenommen, die zum Aufbau des Suchbaumes benötigt werden.

- Berechne  $C = \{(v, w) \mid c_{vw} = c_{wv}\}$

In der Menge  $C$  werden alle Knotenpaare gespeichert, die für beide Ordnungen gleichviele Kreuzungen erzeugen. Bei diesen Paaren ist es also egal, wie sie geordnet werden. Deshalb müssen sie beim Aufbau des Suchbaumes nicht mehr beachtet werden. Das Problem wird also um diese Paare reduziert.

- $D_0$  ist ein transitiv abgeschlossener, gerichteter, azyklischer Graph.  
 $D_0 = (V_0, E_0)$  mit der Knotenmenge  $V_0 = L_2$ . Für die Kanten gilt: eine Kante  $vw \in E_0$ , falls  $c_{vw} = 0$  und  $c_{wv} \neq 0$ .  
 Der Graph enthält also Kanten zwischen allen Knotenpaaren, die in ihrer natürlichen Ordnung vorliegen und somit keine Kreuzungen erzeugen (siehe Behauptung 3). Für diese Paare ist dadurch eine optimale Ordnung festgelegt.
- $B_0$  ist ein Budget für die verbleibende Anzahl Kantenkreuzungen.

$$B_0 = k - \sum_{(v,w) \in C} c_{vw} - \sum_{vw \in D_0} c_{vw}$$

Beachte, dass  $\sum_{vw \in D_0} c_{vw} = 0$ .  $B_0$  verwaltet somit auch die Anzahl Kantenkreuzungen, die die Paare aus der Menge  $C$  erzeugen. Um diese Anzahl wird  $B_0$  reduziert.

### Schritt 3: Aufbau und Durchlaufen des Suchbaumes

In diesem Schritt wird der Suchbaum aufgebaut. Die Knoten sind mit  $(D, B)$  beschriftet. Für die Wurzel gilt  $D = D_0$  und  $B = B_0$ . Wird nun von einem Knoten mit Label  $(D, B)$  ausgegangen, so sucht man ein ungeordnetes Paar  $(v, w)$  mit  $(v, w) \notin C$ . Das heißt es gibt keine Kante  $vw$  in  $D$ , die  $v$  und  $w$  als Endknoten hat und es muss gelten:  $c_{vw} \neq c_{wv}$ . Für dieses ausgewählte Paar  $(v, w)$  werden nun in den Verzweigungen die verschiedenen Ordnungen  $v < w$  bzw.  $w < v$  festgelegt. Dabei entstehen zwei Kindknoten  $(D_1, B_1)$  und  $(D_2, B_2)$  (siehe auch Abb. 7). Für diese Kindknoten müssen nun die Strukturen  $D$  und  $B$  aktualisiert werden.

$D_1$ : Transitiver Abschluß von  $D \cup vw$

Dabei wird in den Graph  $D$  des Elternknotens die gerichtete Kante  $vw$  eingefügt. Das heißt in  $G$  liegt  $v$  vor  $w$ . Das Bilden des transitiven Abschlusses stellt sicher, dass alle Knoten, die in  $G$  vor  $v$  liegen, somit auch vor  $w$  liegen.

$$B_1 = B - c_{vw} - \sum_{pq \in M_1} c_{pq} \quad \text{mit } M_1 = \{pq \mid pq \in D_1, pq \notin D \cup vw, c_{pq} \neq c_{qp}\}$$

$D_2$ : Transitiver Abschluß von  $D \cup wv$

$$B_2 = B - c_{wv} - \sum_{pq \in M_2} c_{pq} \quad \text{mit } M_2 = \{pq \mid pq \in D_2, pq \notin D \cup wv, c_{pq} \neq c_{qp}\}$$

Beim Aufbau und Durchlaufen des Suchbaumes müssen noch einige Kriterien beachtet werden:

- Ein **Blatt** entsteht, falls es kein ungeordnetes Knotenpaar  $(v, w)$  mehr gibt, welches nicht in  $C$  ist, oder falls beide Kindbudgets negativ werden.
- Ein Blatt ist ein **Lösungsblatt**, wenn für alle  $(v, w)$  gilt:

$$\text{falls } vw \notin D \text{ und } wv \notin D \Rightarrow (v, w) \in C$$

- Der Algorithmus endet, wenn der Suchbaum ganz durchlaufen ist. Die Ausgabe ist dann

JA und  $\pi_{opt}$ , falls ein Lösungsblatt existiert

NEIN, sonst.

Zur Bestimmung der optimalen Lösung wird das Lösungsblatt mit dem größten Budget gesucht. Eine topologische Sortierung auf dem entsprechenden Graph  $D$  liefert dann die optimale Ordnung  $\pi_{opt}$ .

**Bemerkungen:** In Schritt 0 wird die Berechnung von  $c_{vw}$  gestoppt, sobald  $c_{vw} \geq k + 1$ . Der Grund dafür liegt darin, dass beim Aufbau des Suchbaumes ein Kind für  $v < w$  ein negatives Budget hätte, was wir aber vermeiden wollen.

Zum Aufbau und Durchlaufen des Suchbaumes in Schritt 3 kann Tiefensuche (DFS) oder Breitensuche (BFS) verwendet werden.

### 3.3 Beispiel

In diesem Kapitel möchte ich noch ein Beispiel präsentieren, das zur Verdeutlichung des Algorithmus beitragen soll.

*Eingabe:*  $\langle G, \pi_1 = (n_1 < n_2 < n_3), k = 4 \rangle$

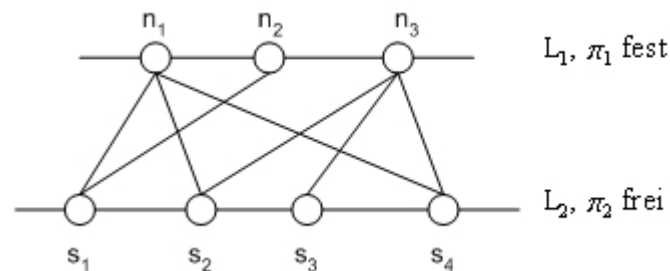


Abbildung 8: Graph  $G$ , der optimiert werden soll.

Nun wird mit Hilfe des 1-seitigen Kreuzungsminimierungs-Algorithmus eine optimale Ordnung der Knoten auf  $L_2$  gesucht, so dass die Kreuzungszahl minimal wird.

**Algorithmus:** 1-SEITIGE KREUZUNGSMINIMIERUNGSchritt 0: Berechnung der Kreuzungszahlen

$$\begin{array}{llll}
c_{s_1 s_2} = 1 & c_{s_2 s_1} = 2 & c_{s_2 s_3} = 0 & c_{s_3 s_2} = 1 \\
c_{s_1 s_3} = 0 & c_{s_3 s_1} = 2 & c_{s_2 s_4} = 1 & c_{s_4 s_2} = 1 \\
c_{s_1 s_4} = 1 & c_{s_4 s_1} = 2 & c_{s_3 s_4} = 1 & c_{s_4 s_3} = 0
\end{array}$$

Schritt 1:

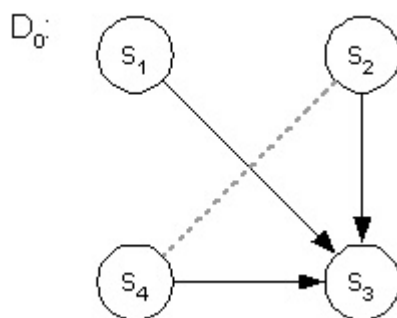
$$\sum_{(v,w)} \min(c_{vw}, c_{wv}) = 3 \quad \text{und} \quad \sum_{(v,w)} \max(c_{vw}, c_{wv}) = 9 \quad \implies 3 \leq k \leq 9$$

Das gegebene  $k = 4$  liegt also innerhalb der oberen und unteren Schranke, was bedeutet, dass  $k$  gut gewählt ist. Wäre nämlich  $k$  kleiner als die untere Schranke, so kann man dafür keine Zeichnung finden. Wäre  $k$  hingegen grösser als die obere Schranke, so bringt der Parameter  $k$  keinen Zeitgewinn mehr. Denn FPT-Algorithmen sind abhängig von einer "guten" Wahl von  $k$ .

Schritt 2:

$$C = \{(v, w) \mid c_{vw} = c_{wv}\} = \{(s_2, s_4)\}$$

Die Datenstrukturen  $D_0$  und  $B_0$  werden jetzt initialisiert.

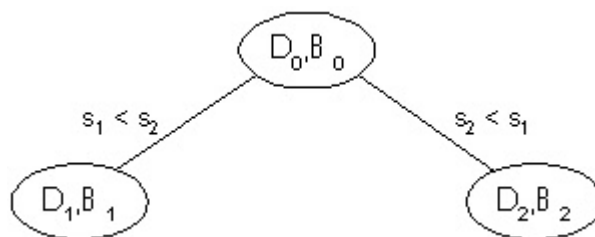


$$B_0 = k - \sum_{vw \in D_0} c_{vw} - \sum_{(v,w) \in C} c_{vw} = 4 - 0 - 1 = 3$$

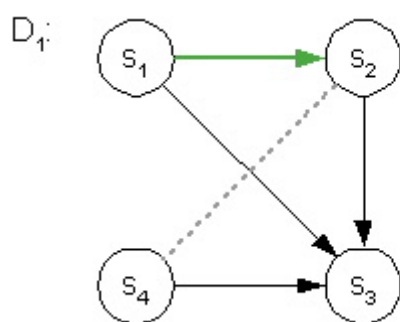
Schritt 3:

Als ungeordnetes Paar wählen wir das Paar  $(s_1, s_2)$ . Welches somit in den verschiedenen Zweigen geordnet wird.

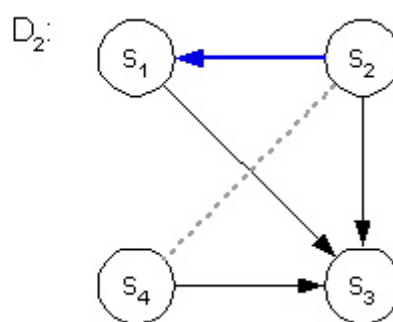
Der Suchbaum sieht also wie folgt aus.



Für die Kindknoten müssen nun die Strukturen  $D$  und  $B$  bestimmt werden, wobei bei  $D_1$  die gerichtete Kante  $s_1s_2$  eingefügt wird, und bei  $D_2$  die Kante  $s_2s_1$ . Das Budget wird dann jeweils um die Kreuzungszahlen dieser Kanten reduziert.

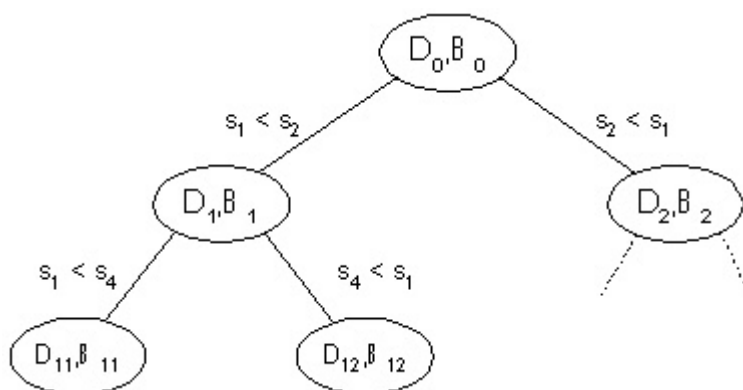


$$B_1 = 3 - 1 - 0 = 2$$

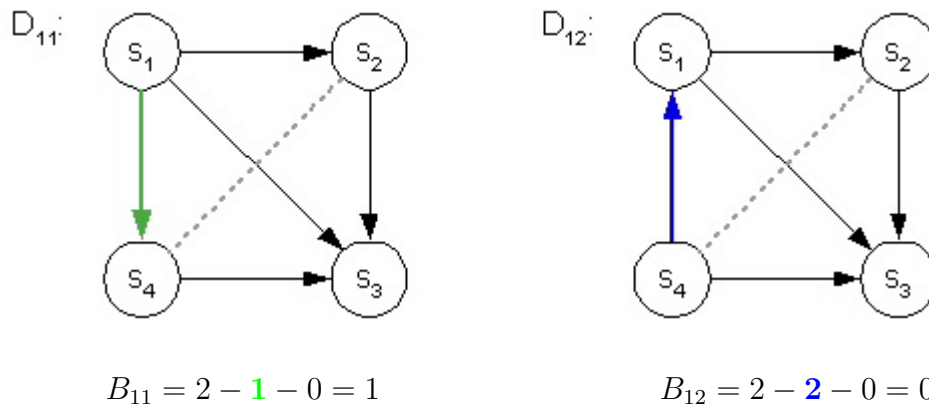


$$B_2 = 3 - 2 - 0 = 1$$

Die Kriterien für ein Blatt treffen bei beiden Baumknoten nicht zu, deshalb wird ein neues ungeordnetes Paar gesucht, für welches dann wieder zwei Kindknoten entstehen. Zuerst werden wir an dem Knoten mit Label  $(D_1, B_1)$  weitermachen. Das neue ungeordnete Paar soll  $(s_1, s_4)$  sein. Dadurch wird der Suchbaum wie folgt erweitert.



Auch jetzt müssen die Strukturen  $D$  und  $B$  wieder erneuert werden.

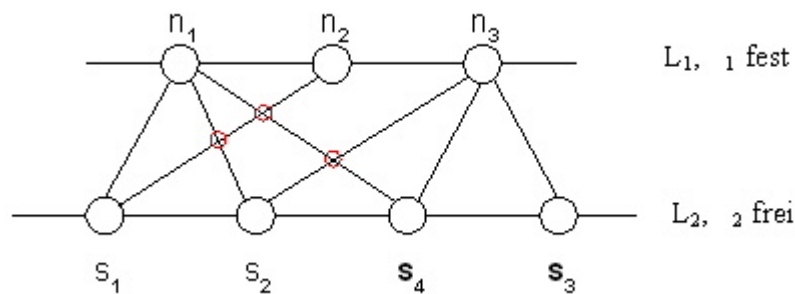


Beachtet man nun wieder die Kriterien für Blätter, so stellt man fest, dass der Knoten mit Label  $(D_{12}, B_{12})$  ein Blatt sein muss, da ansonsten die Kindknoten negatives Budget hätten.

Wenn man nun für den Knoten mit Label  $(D_{11}, B_{11})$  ein weiteres ungeordnetes Paar auswählen will, stellt man fest, dass hier das andere Kriterium für ein Blatt zutrifft. Da alle Knotenpaare entweder in  $C$  sind, oder schon eine Kante in  $D_{11}$  haben. Dies bedeutet auch, dass dieses Blatt gleichzeitig ein Lösungsblatt ist. Auch das Blatt mit Label  $(D_{12}, B_{12})$  ist ein Lösungsblatt. Das Budget  $B_{11}$  ist jedoch größer als das Budget  $B_{12}$  und auch größer als die Kindbudgets von  $B_2$ . Somit liefert eine topologische Sortierung von  $D_{11}$  eine optimale Lösung.

Führt man nun die topologische Sortierung auf  $D_{11}$  durch, so erhält man die Ordnung  $\pi_2 = (s_1 < s_2 < s_4 < s_3)$  für die Lage  $L_2$ .

Damit sieht der neue Graph wie folgt aus:



Wie man nun sieht hat dieser Graph weniger Kantenkreuzungen, als der Graph zu Beginn. Beim Korrektheitsbeweis, der anschließend folgt, wird gezeigt, dass man die optimale Lösung erreicht. Somit ist eine Zeichnung mit drei Kreuzungen optimal.

## 4 Korrektheit und Laufzeit

### 4.1 Korrektheit

Um die Korrektheit des Algorithmus zu zeigen, werden zunächst zwei Invarianten definiert. Diese lauten wie folgt:

(i)  $D$  ist ein transitiv abgeschlossener, gerichteter, azyklischer Graph.

(ii)  $B = k - \sum_{(v,w) \in C} c_{vw} - \sum_{vw \in D \wedge vw \notin C} c_{vw}$

Nun muss gezeigt werden, dass diese beiden Invarianten für alle Knoten mit Label  $(D, B)$  gelten. Der Beweis erfolgt mit Induktion.

**I.A.:** (i) und (ii) gelten für die Wurzel  $(D_0, B_0)$ .

**I.S.:** Man wählt ein ungeordnetes Paar  $(v, w)$  mit  $c_{vw} \neq c_{wv}$ . Dafür werden im Suchbaum zwei Kindknoten  $D \cup vw$  und  $D \cup wv$  erzeugt. Für diese Kindknoten müssen die Invarianten nun gelten.

zu (i):  $D \cup vw$  ist azyklisch.

*denn:* Angenommen  $D \cup vw$  enthält einen gerichteten Zykel. Dann gibt es auch einen Weg von  $w$  nach  $v$ . Da  $D$  aber transitiv abgeschlossen ist, existiert deshalb auch eine Kante  $wv$ . Dies ist jedoch ein Widerspruch zu der Tatsache, dass  $(v, w)$  ein ungeordnetes Paar war. Deshalb kann  $D \cup vw$  keinen gerichteten Zykel enthalten.

Der transitive Abschluss eines gerichteten azyklischen Graphen ist wieder azyklisch. Für  $D \cup wv$  gilt dies analog. Somit gilt auch für die Kindknoten die erste Invariante. zu (ii): gilt wegen Definition von  $B$ .

Mit Hilfe dieser Invarianten zeigt man nun die Korrektheit des Algorithmus. Nach dem Aufbau und Durchlaufen des Suchbaumes hat man entweder ein Lösungsblatt gefunden, oder nicht. Falls nicht, so gibt es keine Lösung. Hat man jedoch ein Lösungsblatt gefunden, so stellen die Invarianten sicher, dass es auch eine Lösung ist. Wegen Invariante (i) weiss man, dass  $D$  ein azyklischer, gerichteter Graph ist. Dieser hat somit eine topologische Sortierung. Die topologische Sortierung gibt dann eine Ordnung  $\pi_2$  an, für die  $cr(G, \pi_1, \pi_2) = k - B$  gilt. Denn in dem Budget des Lösungsblattes ist gespeichert, wieviele Kantenkreuzungen auf dem Weg zu diesem Blatt erzeugt wurden.

Nun stellt sich die Frage, warum es ausreicht, nur die Lösungsblätter zu betrachten und die anderen Knoten zu vernachlässigen. Die Lösungsblätter speichern alle Ordnungen  $\pi_2$ , in denen alle geeigneten Paare in ihrer natürlichen Ordnung vorkommen, und für die  $cr(G, \pi_1, \pi_2) \leq k$ . Lemma 1 sagt nun, dass es ausreicht alle diese Ordnungen für  $\pi_2$  zu betrachten, um zu entscheiden, ob das Problem eine JA-Instanz ist oder eine NEIN-Instanz. Denn um die Kantenkreuzungszahl so gering wie möglich zu halten, müssen alle Paare, die bei einer Ordnung keine Kreuzungen erzeugen auch in dieser Ordnung vorliegen.

## 4.2 Laufzeit

Zur Analyse der Laufzeit wird zuerst mal eine obere Schranke für die Anzahl Knoten im Suchbaum ermittelt. Deshalb werden nur ungeordnete Paare  $(v, w)$  mit  $c_{vw} \neq c_{wv}$  betrachtet. Denn nur für diese Paare werden Kindknoten erzeugt. Betrachtet man nun das Budget der Kindknoten, so reduziert sich dieses beim 1. Kind um mindestens 1 und beim 2. Kind um mindestens 2. So ergibt sich folgende Rekursionsgleichung:

$$s_B = s_{B-1} + s_{B-2} + 1 \quad \text{für } B \geq 2; \quad s_0 = 1; \quad s_1 = 2$$

Hierbei gibt  $s_B$  die Anzahl Knoten in einem Baum an, dessen Wurzel Budget  $B$  hat. Diese Anzahl Knoten kann aus der Anzahl Knoten in den beiden Teilbäumen mit Budget  $B-1$  und  $B-2$  berechnet werden. Die Anfangsbedingungen für  $B=0$  und  $B=1$  ergeben sich wie folgt. Falls das Budget  $B=0$  ist, so muss es sich um ein Blatt handeln, da ansonsten die beiden Kindknoten negatives Budget hätten. Somit ist die Anzahl Knoten in diesem Teilbaum  $s_0 = 1$ . Ist das Budget  $B=1$  dann handelt es sich um einen Knoten, der nur einen Nachfolger (Blatt) hat. Dieser Teilbaum besteht also aus  $s_1 = 2$  Knoten. Um nun eine obere Schranke für  $s_B$  anzugeben, zeigt man mit Induktion, dass

$$s_B = F_{B+2} + F_{B+1} - 1 \quad \text{für } B \geq 0$$

wobei  $F_B = F_{B-1} + F_{B-2}$  die Fibonacci-Folge mit  $F_1 = 1$  und  $F_2 = 1$  ist.

$$\text{I.A.: } s_0 = F_2 + F_1 - 1 = 1; \quad s_1 = F_3 + F_2 - 1 = 2$$

**I.V.:** Behauptung gilt für  $s_{B-1}$  und  $s_{B-2}$

$$\begin{aligned} \text{I.S.: } s_B &= s_{B-1} + s_{B-2} + 1 \\ &= F_B + F_{B-1} - 1 + F_{B+1} + F_B - 1 + 1 \\ &= F_{B+2} + F_{B+1} - 1 \end{aligned}$$

Für die Fibonacci-Folge kann man eine obere Schranke angeben.

$s_{B_0} < \frac{\phi^{k+2}}{\sqrt{5}} + \frac{\phi^{k+1}}{\sqrt{5}} - 1 < 1,2 \cdot \phi^{k+1}$  für  $B_0 \leq k$  wobei  $\phi = \frac{1+\sqrt{5}}{2}$  der **Goldene Schnitt** ist.

Somit ergibt sich für die Anzahl Knoten im Suchbaum eine obere Schranke von  $O(\phi^k)$ .

Um die Laufzeit für Schritt 3 anzugeben, muss man nun noch wissen, wie lange es dauert, bei jedem Knoten den Graph  $D$  zu aktualisieren, nachdem man ein geordnetes Paar  $v < w$  bzw.  $w < v$  eingefügt hat. Das Bilden des transitiven Abschlusses geschieht in  $O(|L_2|^2)$  [5]. Somit ergibt dies für Schritt 3 eine Laufzeit von  $O(\phi^k \cdot |L_2|^2)$ .

Die Schritte 0 - 2 haben eine Laufzeit von  $O(k \cdot |L_2|^2 + |L_1||L_2|)$ . Deshalb ist die gesamte Laufzeit des Algorithmus in  $O(\phi^k \cdot |L_2|^2 + |L_1||L_2|)$ .

Wie wir nun gesehen haben, hat man mit diesem Algorithmus eine Laufzeit erreicht, die nur noch exponentiell von dem Parameter  $k$  abhängt und polynomial in der Eingabegröße ist. Dies ist insbesondere ein Merkmal für FPT-Algorithmen, auf welche wir im nächsten Kapitel genauer eingehen wollen.

## 5 FPT-Algorithmen

Wie schon erwähnt, ist die 1-seitige Kreuzungsminimierung ein Beispiel für einen FPT-Algorithmus. Denn hier wird die erlaubte Kreuzungsanzahl auf einen festen Parameter  $k$  eingeschränkt, so dass die Eingabegröße, die für die schlechte Laufzeit verantwortlich ist, nicht mehr exponentiell auftritt. So ein Problem mit Eingabegröße  $n$  und festem Parameter  $k$  nennt man "*Fixed Parameter Tractable*", falls es einen Algorithmus gibt, der das Problem in einer Zeit von  $f(k) \cdot n^\alpha$  löst. Hierbei ist  $\alpha$  eine Konstante und  $f$  eine beliebige Funktion, die nur von  $k$  abhängt. Bei FPT-Algorithmen versucht man also den Teil der Eingabe, der für die schlechte Laufzeit verantwortlich ist auf einen festen Parameter  $k$  zu reduzieren. Die Funktion  $f$  ist hier also ausschlaggebend für die Güte des FPT-Algorithmus. Nur wenn  $k$  im Verhältnis zur Eingabegröße klein ist, dann ist der Algorithmus auch effizient.

Um FPT-Algorithmen zu finden, gibt es einige grundlegende Methoden.

Tiefenbeschränkter Suchbaum (Bounded Search Tree) [1]

Reduktion auf den Problemkern (Problem-Kernel)

### Tiefenbeschränkter Suchbaum

Bei dieser Methode wird ein Suchbaum so aufgebaut, dass für kleine Teilmengen mindestens eine Lösung optimal ist. Bei der 1-seitigen Kreuzungsminimierung entsprechen die "kleinen Teilmengen" den Knotenpaaren, für die dann jeweils eine Verzweigung die optimale Ordnung enthält. Wird der Suchbaum so aufgebaut, dann bekommt man in den verschiedenen Zweigen unterschiedliche Lösungen, von denen mindestens eine optimal ist. Dies ist jedoch noch abhängig von der geeigneten Wahl des Parameters  $k$ , der die Tiefe des Suchbaumes beschränkt.

### Reduktion auf den Problemkern

Hier ist die Idee, das gegebene Problem  $P$  auf ein kleineres Problem  $P'$  zu reduzieren, wobei die Größe von  $P'$  durch eine Funktion beschränkt ist, die nur von dem Parameter  $k$  abhängt. Bei der 1-seitigen Kreuzungsminimierung findet auch eine Reduktion statt. Denn bei der Initialisierung in Schritt 2, werden alle geeigneten Paare in ihrer natürlichen Ordnung vorab geordnet. Und alle Paare, für die beide Ordnungen gleichviele Kreuzungen erzeugen, werden in der Menge  $C$  zusammengefasst. Deshalb wird beim Aufbau des Suchbaumes nur noch ein kleineres Problem  $P'$  betrachtet, welches aus den ungeeigneten Knotenpaaren mit  $c_{vw} \neq c_{wv}$  besteht. Die Größe des Problems  $P'$  ist hier jedoch **nicht** abhängig von dem Parameter  $k$ . Somit wird diese Methode nur ansatzweise verwendet.

## 6 Zusammenfassung und Ausblick

Mit diesem FPT-Algorithmus hat man nun eine einfache und schnell zu implementierende Möglichkeit zur Lösung der 1-seitigen Kreuzungsminimierung. Im Gegensatz zu den Heuristiken erhält man hier eine optimale Lösung mit einer Laufzeit  $O(\phi^k \cdot n^2)$ , die bezüglich des Parameters  $k$  zwar exponentiell ist, aber in der Eingabegröße polynomial. Wir haben somit einen Algorithmus, der ein  $NP$ -vollständiges Problem effizient löst. Dies bedeutet jedoch nicht, dass  $P = NP$  gilt. Denn die Laufzeit von parametrisierten Algorithmen  $f(k) \cdot n^\alpha$  ist von der Güte der Funktion  $f$  und von dem Parameter  $k$  abhängig. Nicht für jedes Problem kann also ein 'gutes'  $k$  gefunden werden. Für viele Probleme ist der Parameter  $k$  meist schon als Teil der Eingabe gegeben. Bei der 1-seitigen Kreuzungsminimierung z.B. kann  $k$  auch durch  $1.47 \cdot \sum_{v,w \in L_2} \min(c_{vw}, c_{wv})$  berechnet werden, da  $\text{opt}(G, \pi_1) \leq \sum_{v,w \in L_2} \min(c_{vw}, c_{wv})$ . Dies wird von Nagamochi in [4] gezeigt.

In dieser Arbeit wurde der Algorithmus nur für einfache Graphen vorgestellt. Man kann ihn jedoch auch leicht auf Graphen mit Mehrfachkanten übertragen. Dafür werden die Mehrfachkanten zu einer gewichteten Kante zusammengefasst, die dann bei der Berechnung der Kreuzungszahl entsprechend berücksichtigt wird.

Für die Zukunft wird es wohl sehr interessant sein, die Methode der Parametrisierung auf andere  $NP$ -schwere Probleme, vor allem auch im Bereich 'Graphen zeichnen', anzuwenden.

## Literatur

- [1] R.G. Downey; M.R. Fellows. *Parametrized Complexity*. Springer, 1999.
- [2] M.R. Garey; D.S. Johnson. *Crossing number is NP-complete*. SIAM J. Algebraic Discrete Methods, pages 4(3): 312–316, 1983.
- [3] M. Jünger; P. Mutzel. *2-layer straightline crossing minimization: performance of exact and heuristic algorithms*. J. Graph Algorithms Appl., pages 1(1): 1–25, 1997.
- [4] H. Nagamochi. *An improved approximation to the one-sided bilayer drawing*. Proc. GD '03, volume 2912 LNCS, pages 406–818, 2003.
- [5] T. Corman; R. Rivest; C. Leiserson; C. Stein. *Introduction to Algorithms - Second Edition*. MIT Press, 2001.
- [6] G. Di Battista; P. Eades; R. Tamassia; I.G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.
- [7] P. Eades; N.C. Wormald. *Edge Crossings in Drawings of bipartite Graphs*. Algorithmica, pages 11(4): 379–403, 1994.