

A Linear-Time Algorithm For Finding Tree-Decompositions Of Small Treewidth

Maximilian F. Göckel – *uzkns@student.kit.edu*

Institut für Theoretische Informatik - Proseminar Algorithmen für NP-schwere Probleme



Eine Baumzerteilung eines Graphen $G = (V, E)$ ist ein Tupel (X, T) wo $T = (I, F)$ ein Baum ist und $X = \{X_i | i \in I\}$ eine Familie von Teilmengen von V wobei jedes X_i einen Knoten in T darstellt.

1. $\bigcup_{i \in I} X_i = V$
2. $\forall (v, w) \in E : \exists i \in I : v, w \in X_i$
3. $\forall w \in X_i, X_j : \text{Jedes } X_k \text{ im Pfad zwischen } X_i, X_j \text{ enthält } w$

Tree-decomposition

Veranschaulichung

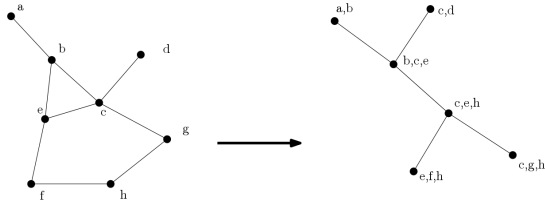


Figure: 1

Jede Baumzerteilung hat eine "Baumweite" (treewidth).

- Baumweite einer Zerteilung: $\max(|X_i|_{i \in I} - 1)$ ("Zerteilungsweite")
- Baumweite eines Graphen: Minimale Zerteilungsweite aller Zerteilungen

Eine Baumzerteilung der Weite max. k heißt auch k -Baumzerteilung oder k -Zerteilung.

Die folgenden Aussagen zu k -Bäumen sind äquivalent:

1. $G = (V, E)$ ist ein k -Baum
2. G ist verbunden und hat eine k -Clique, aber keine $(k + 2)$ -Clique und
 - Jeder minimale Separator von G ist eine k -Clique oder
 - \forall nicht-adjazenten Knotenpaare $x, y \in V \exists k$ Wege $x \rightarrow y$
3. G ist verbunden, $|E| = k|V| - \frac{1}{2}k(k + 1)$ und jeder minimale Separator von G ist eine k -Clique
4. G hat Knoten v mit 3 Eigenschaften:
 - $\deg(v) = k$ und
 - Nachbarknoten von v formen eine k -Clique und
 - $G \setminus v$ ist k -Baum

Jeder komplette Graph mit k Knoten ist damit auch ein k -Baum.

Andersherum: Ein k -Baum-Graph G mit $n \geq k$ Knoten kann aus einem k -Baum-Graph $H = (V, E)$ mit $n - 1$ Knoten wie folgt erstellt werden:

- Zu H einen Knoten u hinzufügen ($|V| = (n - 1) \rightarrow |V| = n$)
- Knoten u mit bel. Knoten v_1, \dots, v_k verbinden

Damit wird Aussage 4 erfüllt.

Graph $G = (V, E)$ ist partieller k -Baum \Leftrightarrow

- G ist Teilgraph eines k -Baumes oder
- G hat Baumweite max. k

- Maximum-Weight Independent Set in Linearzeit lösbar
- Hohe Baumweite \Leftrightarrow Hohe Komplexität in der Systemanalyse
- Erkennung von partiellen k-Bäumen
- Pfadweite
- Graphen-Erkennungsalgorithmen



Ein Knoten v ist ...

- ... "von niedrigem Grad" wenn $\deg(v) \leq d$
 - $d = 2k^3 \cdot (k + 1) \cdot (4k^2 + 12k + 16)$
 - Analog: Hoher Grad $\Leftrightarrow \deg(v) > d$
 - Auch "low-deg.-" und "high-deg.-Knoten" genannt
- ... "Freundlich" wenn er low-deg. und adjazent zu einem weiteren low-deg.-Knoten ist
- ... "Simplizial" wenn alle Nachbarn in einer Clique sind
- ... "I-Simplizial" wenn simp. in G' und $\deg(v) \leq k$ in G



Verbesserter Graph G'

Erstellung und Eigenschaften

$G' = (V, E')$ ist $G = (V, E)$ mit Kanten $(v, w) \in E' \forall v, w \in V$ sodass v, w min. $k + 1$ gem. Nachbarn mit Grad max. k haben.

LEMMA 4.1.: $\text{tw}(G) \leq k \Leftrightarrow \text{tw}(G') \leq k$.

Außerdem ist jede k -Zerteilung von G auch eine k -Zerteilung von G' und umgekehrt.

Maximum Matching $M \subseteq E$

$M \subseteq E$ ist Maximum Matching in $G = (V, E) \Leftrightarrow$ Keine 2 Kanten aus M haben gemeinsamen Endknoten und $|M|$ maximal


Ein Maximum Matching kann in $O(|V|)$ gefunden werden, wenn die Baumweite durch ein k gebunden ist.



Eingabe: Graph $G = (V, E)$ mit $|V| = n$ und Konstante $k \in \mathbb{N}$.
Der Graph wird als Adjazenzliste übergeben.

Ausgabe in $O(n)$:

- "Baumweite von G ist größer als k "
- "Baumweite von G ist maximal k "
 - Baumzerteilung von G mit Baumweite k

Für "sehr kleine"  Graphen werden andere bekannte Algorithmen genutzt, ansonsten wird wie folgt vorgegangen:

LEMMA 4.2.: G hat Baumweite max. $k \Leftarrow 1$ von 2 gilt mindestens:

- G hat min. $\frac{|V|}{4k^2+12k+16} =: \lambda$ Friendly-Knoten
- G' hat min. $\frac{1}{8k^2+24k+32} \cdot |V|$ I-simp.-Knoten

Algorithmus hat eine Fallunterscheidung ab λ Friendly-Knoten.
Die Anzahl Friendly-Knoten in G wird mit n_f notiert.



1. Maximum-Matching $M \subseteq E$ finden
2. Jede Kante in M kontrahieren um Graphen $\tilde{G} = (\tilde{V}, \tilde{E})$ zu erhalten
3. Kompletten Algorithmus auf \tilde{G} ausführen um Baumzerteilung (Y, T) von \tilde{G} auszugeben
→ Wenn Baumweite von $\tilde{G} > k \Rightarrow$ **STOP** (LEMMA 3.4.)
4. Mit LEMMA 3.3. $2k + 1$ -Zerteilung (X, T) von G aus (Y, T) erstellen
5. Mit THEOREM 2.4. k -Zerteilung von G errechnen
→ Wenn Baumweite von $G > k \Rightarrow$ **STOP**

1. Maximum Matching $M \subseteq E$ finden

Ein Maximum Matching kann greedy in $O(|V| + |E|)$ gefunden werden.

LEMMA 2.3.: " $\text{tw}(G) \leq k \Rightarrow |E| \leq k|V| - \frac{1}{2}k(k+1)$ "

$\Rightarrow |E| \in O(n)$

\Rightarrow Max. Matching kann in $O(n)$ gefunden werden

2. Jede Kante in M kontrahieren um Graphen \tilde{G} zu erhalten

Eine Kante kann in $O(1)$ kontrahiert werden, liegt der Graph als Adjazenzliste vor.

LEMMA 2.3.: " $\text{tw}(G) \leq k \Rightarrow |E| \leq k|V| - \frac{1}{2}k(k+1)$ "

$\Rightarrow |M| \in O(|E|) \in O(|V|)$

\Rightarrow Alle Kanten können in $O(n)$ kontrahiert werden.

3. Kompletten Algorithmus auf \tilde{G} ausführen um Baumzerteilung (Y, T) von \tilde{G} auszugeben

Ein Maximum Matching hat min. $\frac{n_f}{2 \cdot (2k^3(k+1)(4k^2+12k+16))}$ Kanten.

Für jeden Friendly-Knoten gilt:

- Er ist Endpunkt von einem $m \in M$ oder

- Er ist adjazent zu einem Friendly-Knoten, der Endpunkt ist

$\Rightarrow \forall m \in M$ werden max. $2d$ Friendly-Knoten assoziiert, die Endpunkt sind oder adjazent zu einem Friendly-Endpunkt sind. \Rightarrow : Ist ein Friendly-Knoten nicht assoziiert so ist M nicht maximal $\Rightarrow |M| \geq \frac{n_f}{2d}$

3. Kompletten Algorithmus auf \tilde{G} ausführen um Baumzerteilung (Y, T) von \tilde{G} auszugeben

Ein Maximum Matching hat min. $\frac{n_f}{2 \cdot (2k^3(k+1)(4k^2+12k+16))}$ Kanten.

$$\Rightarrow |\tilde{V}| = \left(1 - \frac{1}{2d(4k^2+12k+16)}\right) \cdot |V|$$



4. Mit **LEMMA 3.3**. Zerteilung (X, T) von G aus (Y, T) erstellen

$$f_M : V \mapsto \tilde{V} =$$




$$\begin{cases} f_M(v) = v & \text{Wenn } v \text{ nicht Endpunkt in } M \text{ ist} \\ f_M(v) = f_M(w) & \text{Der Knoten der bei der Kontraktion } (v, w) \in M \text{ bleibt} \end{cases}$$

(Y, T) Zerteilung von \tilde{G} , so ist (X, T) mit $X_i = \{v \in V \mid f_M(v) \in Y_i\}$
Zerteilung von G mit Weite max. $2k + 1$

5. k -Zerteilung von G errechnen

5.1. prüfen ob Weite von $G > k$ ist \Rightarrow **STOP**

THEOREM 2.4.: " $\forall k, l \in \mathbb{N} \exists$ Linearzeitalgorithmus, welcher aus einem Graph $G = (V, E)$ und einer l -Zerteilung prüft ob die Baumweite von G max. k ist und eine k -Zerteilung errechnet"

Laufzeit: $O(l^{l-2} \cdot ((2l+3)^{2l+3} \cdot (\frac{8}{3}2^{2k+2})^{2l+3})^{2l-1})$  $O(k^3)$ bei $l \in O(k)$

1. Improved-Graph G' berechnen
 $\rightarrow \exists$ l.simp.-Knoten v mit $\deg(v) = k + 1 \Rightarrow$ **STOP**
2. Alle l.simp.-Knoten in Menge SL und von G entfernen $\Rightarrow \hat{G}$ entsteht
 $\rightarrow |SL| < c_2 \cdot |V| \Rightarrow$ **STOP** (THEOREM 4.2.)
3. Algorithmus rekursiv auf \hat{G} ausführen \Rightarrow Ausgabe von Zerteilung (Y, T) von \hat{G}
 $\rightarrow \text{tw}(\hat{G}) > k \Rightarrow$ **STOP** (\hat{G} Teilgraph von $G \Rightarrow \text{tw}(G) > k$)
4. Füge SL wieder in die Zerteilung (Y, T) ein
 \Rightarrow Baumzerteilung (X, T) von G mit Baumweite max. k

1. Improved-Graph G' berechnen



- Knoten durchzählen (v_1, v_2, \dots, v_n)
- Queue Q mit Tripeln der Form $((v, w), x)$ oder $((v, w), -)$ mit $v, w, x \in V$
- Array S mit Liste $S[v_i]$ die für v_i Paare von Knoten enthält
- $\forall (v_i, v_j) \in E$ mit $i < j \rightarrow ((v_i, v_j), -)$ in Q ein stapeln
- $\forall v \in V$ mit $\deg(v) \leq k \rightarrow$ Alle Nachbargaare $v_i, v_j \in N_G(v)$ durch $((v_i, v_j), v)$ in Q ein stapeln
-

Laufzeit

Visualisierung

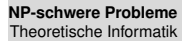


⋮

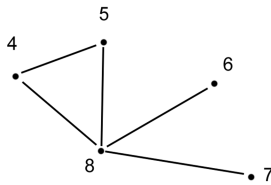
- Q zweimal bucket-sortieren, erst nach dem ersten, dann nach dem zweiten Knoten im Tupel
→ Alle Tripel $((v_i, v_j), v)$ mit gleichem i, j sind hintereinander
⇒ Es kann schnell geprüft werden welche Paare $k + 1$ gemeinsame Nachbarn haben
- Sind min. $k + 1$ Einträge $((v_i, v_j), v)$ für ein Paar v_i, v_j in Q untereinander ⇒ (v_i, v_j) ist in G'
- Für jedes obiges Paar (v_i, v_j) und wenn $((v_i, v_j), -) \in Q \rightarrow$ Füge (v_i, v_j) für jedes $((v_i, v_j), v)$ zu $S[v]$ hinzu

$S[v]$ enthält nun alle Kanten von Nachbarn von v . Da $\deg(v) \leq k$ ist $S[v]$ durch k beschränkt.

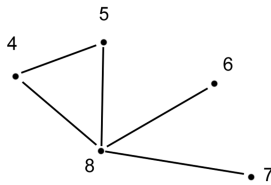
Damit kann durch ansehen von $S[v]$ entschieden werden ob v l -simplizial ist.



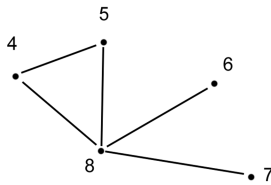
4	8	-
4	5	-
5	8	-
6	8	-
7	8	-
5	8	4
4	8	5
4	5	8
4	6	8
4	7	8
5	6	8
6	7	8



4	8	-
4	5	-
4	8	5
4	5	8
4	6	8
4	7	8
5	8	-
5	8	4
5	6	8
6	8	-
6	7	8
7	8	-



4	5	-
4	5	8
4	6	8
5	6	8
4	7	8
6	7	8
4	8	-
4	8	5
5	8	-
5	8	4
6	8	-
7	8	-



2. Alle l.simp.-Knoten in Menge SL und von G entfernen $\Rightarrow \hat{G}$ entsteht

Durch das Vorgehen von gerade eben können wir die Menge SL in $O(k \cdot n) \in O(n)$ aus S auslesen.



3. Algorithmus rekursiv auf \hat{G} ausführen

Graph \hat{G} hat nach Entfernung von $SL (1 - c_2) \cdot |V|$ Knoten.

Wie in Fall 1 sind alle rekursiven Aufrufe in $O(n)$ möglich.

4. Füge SL wieder in die Zerteilung (Y, T) ein

1. $\forall v \in SL$: Finde ein $Y_{i_v} \in Y$ in dem alle Nachbarn von v sind
 $(N_G(v) \subseteq Y_{i_v})$
2. Füge $Y_{i_v} = \{v\} \cup N_G(v)\}$ zu Y hinzu und mache es adjacent zu Y_{i_v}
 \Rightarrow Baumzerteilung von G mit Baumweite max. k

Y_{i_v} existiert für jedes v , da l.simp.-Knoten in G nicht adjacent sind und $N_G(v)$ eine Clique formt.

LEMMA 2.1.i): " (X, T) Zerteilung von G und $W \subseteq V$ formt Clique in $G \Rightarrow \exists i \in I : W \subseteq X_i$ "

4. Füge SL wieder in die Zerteilung (Y, T) ein



- $\forall l \leq k$: Nimm Queue Q_l wo alle Paare $((v_{i_1}, v_{i_2}, \dots, v_{i_l}), i)$ für $v_{i_x} \in Y_i$ und für alle $i \in I, i_1 < i_2 \dots i_l$ hinzugefügt werden
- Füge zu jedem Q_l noch alle Paare $((v_{i_1}, v_{i_2}, \dots, v_{i_l}), v)$ wo v l -simp. ist und $N_G(v) = \{(v_{i_1}, v_{i_2}, \dots, v_{i_l}) \mid i_1 < i_2 \dots i_l\}$
- Sortiere jedes Q_l l -mal, einmal für jedes v_{i_x} im Tupel. Es entsteht wieder eine Sortierung der v_i
- Für jedes Tupel $((\dots), v)$ kann nun schnell das Tupel $((\dots), i)$ gefunden werden
- Mache den neuen Knoten j_v mit $X_{j_i}\{v\} \cup N_G(v)$ zu dem i adjazent und füge ihn zu Y hinzu

Diese Schritte sind alle zusammen in $O(n)$ ausführbar.

Sämtliche Operationen sind in $O(n)$ wenn k Konstant ist.

Ist k nicht konstant, sondern als Variable Teil der Eingabe, so ist der Algorithmus nicht mehr linear. (Schritt 5.1. ist in $O(k^3)$)

Der konstante Faktor k^3 ist deutlich zu hoch für praktische Anwendung, selbst schon für $k = 4$.

Allerdings wurde bei vielen Operationen grob geschätzt. Es ist zu erwarten, dass die Konstante noch sinken wird.

Der Algorithmus ist Basis für zwei weitere Theoreme:

- \exists Linearzeit-Erkennungsalgorithmus für jede Klasse an Graphen die nicht alle planaren Graphen enthält und in ihren Minoren abgeschlossen ist
- $\forall k \in \mathbb{N} : \exists$ Linearzeitalgo der prüft ob $G = (V, E)$ Pfadweite max. k hat und eine Pfad-Zerteilung ausgibt

Außerdem ist die Erkennung von l.-simp.-Knoten sehr effizient und kann gut als Grundlage für andere Algorithmen genutzt werden.