ETH zürich
Departement Informatik

Algorithmik für schwere Probleme

Prof. Dr. Juraj Hromkovič
Dr. Hans-Joachim Böckenhauer
Dr. Richard Královič
http://www.ita.inf.ethz.ch/alscpr17

Skript: Parametrisierte Algorithmen

Zürich, 9. August 2017

Idee. Finde einen Parameter in der Eingabe, so dass die Laufzeit nur von diesem Parameter exponentiell abhängt, aber polynomiell von der Eingabegrösse.
$\Longrightarrow$ effiziente Algorithmen für Eingaben mit kleinem Parameterwert

Definition 1. Sei U ein Berechnungsproblem und L die Sprache der Eingaben für U. Die Funktion $\mathrm{Par}\colon L \to \mathbb{N}$ heisst *Parametrisierung* von U, falls gilt:

(i) Par ist in Polynomzeit berechenbar.

(ii) Für unendlich viele $k \in \mathbb{N}$ ist die *Parameter-k-Menge* für U

$$\mathrm{Set}_U(k) = \{x \in L \mid \mathrm{Par}(x) = k\}$$

unendlich gross.

Definition 2. Ein Algorithmus A ist ein Par-*parametrisierter* Polynomzeit-Algorithmus für U, falls gilt:

(i) A löst U.

(ii) Es existieren ein Polynom p und eine (berechenbare) Funktion $f\colon \mathbb{N} \to \mathbb{N}$, so dass für jedes $x \in L$ gilt:

$$\mathrm{Time}_A(x) \leq f(\mathrm{Par}(x)) \cdot p(|x|)\,.$$

Definition 3. Wenn ein Par-parametrisierter Algorithmus A für ein Problem U existiert, dann nennen wir U *fixed-parameter-tractable* (*fpt*) bezüglich Par. Wir nennen A in diesem Fall auch einen *fpt-Algorithmus* für U.

Bemerkung. Die Bedingung (ii) in Definition 1 garantiert, dass der Parameter nicht von der Eingabegrösse abhängt (d. h., für jeden Parameterwert gibt es beliebig grosse Instanzen mit diesem Wert).
$\Longrightarrow$ Das verhindert Parametrisierungen wie $\mathrm{Par}(x) = |x|$.

Beobachtung. *Jeder pseudopolynomielle Algorithmus A für ein Zahlproblem U ist ein $|MaxInt|$-parametrisierter Algorithmus.*

Ziel. Entwurfsmethoden für parametrisierte Algorithmen kennenlernen.

1. Methode: Datenreduktion und Kernbildung (kernelization)

Idee. Finde Transformationsregeln, um die gegebene Eingabe (und gegebenenfalls den Parameter) so zu verkleinern, dass schliesslich eine Instanz übrig bleibt, deren Grösse nur vom Parameter abhängt (*Kern*, kernel).
$\Longrightarrow$ Transformation in Polynomzeit abhängig von der Eingabelänge, dann exponentieller Aufwand auf dem Kern.

Beispiel 1 (Vertex Cover (VC)).

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl k.

Frage: Gibt es einen VC der Grösse $\leq k$?

Parameter: k.

Beobachtung. *Sei $G = (V, E)$ ein Graph, sei $S \subseteq V$ ein VC von G mit $|S| \leq k$. Dann enthält S alle Knoten vom Grad $> k$.*

Beweis. Sei u ein Knoten mit $\deg_G(u) > k$. Dann ist u inzident zu $> k$ Kanten. Wenn $u \notin S$ gilt, dann müssen diese Kanten von paarweise verschiedenen anderen Knoten überdeckt werden. Daraus folgt $|S| \geq \deg_G(u) > k$, was ein Widerspruch zur Annahme $|S| \leq k$ ist. □

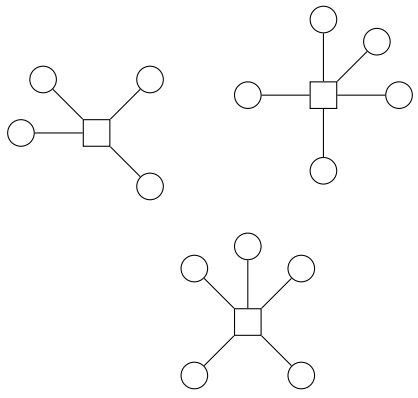
$\Longrightarrow$ **Datenreduktionsregel DRR:** Sei $\Big(G = (V, E), k\Big)$ eine VC-Eingabe und u ein Knoten aus V mit $\deg_G(u) > k$. Dann gilt

$$(G, k) \in \text{VC} \iff (G - \{u\}, k - 1) \in \text{VC}\,.$$

$\Longrightarrow$ **Kern:** Teilinstanz (G', k'), so dass $k' \leq k$ und $G' = (V', E')$ mit $\deg_{G'}(u) \leq k'$ für alle $u \in V'$.

Beobachtung. *Sei G ein Graph mit einem VC der Grösse m und maximalem Knotengrad k. Dann hat G höchstens $m \cdot (k + 1)$ nicht isolierte Knoten.*

Beweis. Diese m Knoten des VC haben jeweils $\leq k$ Nachbarn. Im schlimmsten Fall sind alle disjunkt (die Quadrate in der Abbildung bezeichnen die Knoten des VC, die runden Knoten ihre Nachbarn): □



$\Longrightarrow$ Grösse des Kerns für JA-Instanzen (isolierte Knoten können einfach weggelassen werden): $k'(k'+1)$ hängt nur von k' (und damit von k) ab: $O(k^2)$.

Algorithmus VC-Kern

Eingabe: ungerichteter Graph $G = (V, E)$, $k \in \mathbb{N}$.

Schritt 1: Wende die Reduktionsregel DRR solange wie möglich an, erhalte (G', k') mit $k' \leq k$ und $\deg_{G'}(u) \leq k'$ für alle $u \in V(G')$.

Schritt 2: Falls G' mehr als $k' \cdot (k'+1)$ Knoten hat, gib NEIN aus. Sonst berechne die Lösung für (G', k') mit vollständiger Suche.

Ausgabe: Die in Schritt 2 berechnete Lösung.

Theorem 1. *VC-Kern ist ein k-parametrisierter Polynomzeit-Algorithmus für VC.*

Beweis. Wir beweisen die Korrektheit und Laufzeitschranke einzeln.

1. Korrektheit: Die Korrektheit folgt unmittelbar aus der Korrektheit von DRR.

2. Laufzeit: Anwendung von DRR: abhängig von der verwendeten Datenstruktur, einfach in $O(n)$ für die Eingabegrösse n.

   $\Longrightarrow$ $O(k \cdot n)$ für Schritt 1

   Schritt 2: Vollständige Suche auf Graph der Grösse $k'(k'+1)$: Teste $\binom{k'(k'+1)}{k'}$ mögliche Vertex Cover jeweils in $k' \cdot (k'+1)$ Zeit.

   $\Longrightarrow$ $O\left(k^2 \binom{k(k+1)}{k}\right) \subseteq O(k^2 \cdot (k^2+k)^k) \subseteq O(k^{2k+2})$

   $\Longrightarrow$ Die Gesamtlaufzeit ist in $O(k \cdot n + k^2(k^2+k)^k) \subseteq O(k^{2k+2} \cdot n)$ $\square$

Theorem 2. *Sei U ein Berechnungsproblem und* Par *eine Parametrisierung für U. Dann ist U* Par*-parametrisierbar genau dann, wenn ein Kern von U bezüglich* Par *in polynomieller Zeit berechnet werden kann.*

Beweis. Wir zeigen die Äquivalenz durch zwei Implikationen.

$\Longleftarrow$**:** Wenn ein Kern in polynomieller Zeit berechnet werden kann, reicht eine vollständige Suche für den Kern aus, um U mit einem fpt-Algorithmus zu lösen.

$\Longrightarrow$**:** Sei U Par-parametrisierbar mit dem fpt-Algorithmus A mit Laufzeit $f(k) \cdot n^c$ für den Parameterwert k und eine Konstante c.

Idee: A auf gegebener Instanz für n^{c+1} Schritte laufen lassen:

1. Fall: A terminiert $\Longrightarrow$ in polynomieller Zeit Lösung berechnet, gib trivialen Kern aus (z. B. triviale JA-Instanz oder NEIN-Instanz bei Entscheidungsproblem).

2. Fall: A terminiert nicht $\Longrightarrow$ $n < f(k)$ $\Longrightarrow$ Instanz selbst ist Kern. $\square$

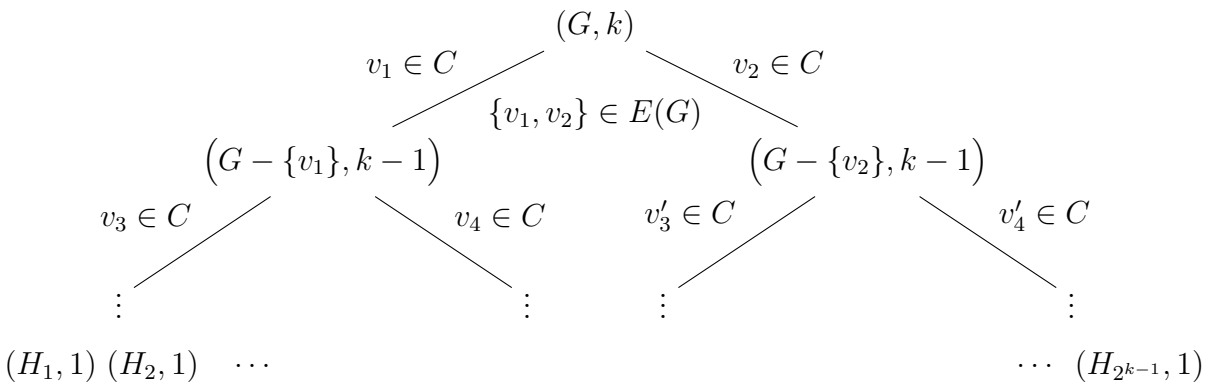
2. Methode: Tiefenbeschränkte Suchbäume

Idee. Systematische vollständige Suche ergibt Entscheidungsbaum (Suchbaum). Zeige, dass man diesen Baum (bei konstanter Verzweigung) nur in einer beschränkten Tiefe (abhängig vom Parameter) durchsuchen muss.

Beispiel 2 (Vertex Cover).

Beobachtung. *Sei $G = (V, E)$ ein Graph, $\{u, v\} \in E$ eine Kante. Jeder VC von G enthält mindestens einen der beiden Knoten u oder v.*

$\Longrightarrow$ **Divide-and-Conquer-Algorithmus.**



Rekursiver Algorithmus. Wähle für (G, k) eine beliebige Kante $\{v_1, v_2\} \in E(G)$ und löse dann die Teilprobleme $(G - \{v_1\}, k-1)$ und $(G - \{v_2\}, k-1)$.
$\Longrightarrow$ Der Suchbaum hat $k-1$ Level (da $(H, 1)$ trivial lösbar ist für alle Graphen H) und jeder Reduktionsschritt ist in $O(n)$ durchführbar.
$\Longrightarrow$ Die Gesamtlaufzeit ist in $O(2^k \cdot n)$.
$\Longrightarrow$ k-parametrisierter Algorithmus.

Beispiel 3 (Cluster-Editing-Problem).

Definition 4. Das *Cluster-Editing-Problem* ist das folgende Entscheidungsproblem.

Eingabe: Ein Graph $G = (V, E)$ und ein $k \in \mathbb{N}$.

Frage: Ist es möglich, durch Hinzufügen oder Löschen von $\leq k$ Kanten G in eine disjunkte Vereinigung von Cliquen umzuwandeln (d. h. in einen Graphen, in dem jede Zusammenhangskomponente eine Clique ist)?

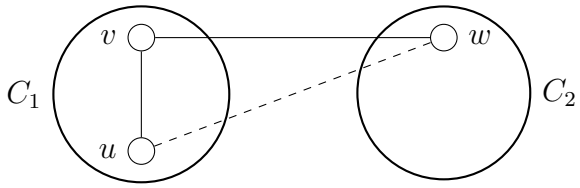
Theorem 3 (Hier ohne Beweis). *Cluster-Editing ist NP-schwer.*

Beobachtung. *Ein Graph $G = (V, E)$ besteht aus disjunkten Cliquen genau dann, wenn es keine drei paarweise verschiedenen Knoten $u, v, w \in V$ gibt, so dass $\{u, v\}, \{v, w\} \in E$ und $\{u, w\} \notin E$ ist (d. h. wenn G keinen induzierten Pfad der Länge 3 enthält).*

Beweis. Wir zeigen die Äquivalenz wieder durch zwei Implikationen.

$\Longrightarrow$**:** Wir zeigen die Kontraposition: Falls $u, v, w \in V$ existieren mit $\{u, v\}, \{u, w\} \in E$ und $\{v, w\} \notin E$, dann liegen u, v, w in derselben Zusammenhangskomponente von G. Wegen $\{v, w\} \notin E$ ist diese Komponente keine Clique.

$\Longleftarrow$**:** Wir nehmen an, dass G keine Vereinigung disjunkter Cliquen ist. Dann gibt es zwei disjunkte Cliquen C_1 und C_2 in G mit $|V(C_1)| \geq 2$, so dass eine Kante $\{v, w\}$ mit $v \in C_1$, $w \in C_2$ existiert. (Falls $|V(C_1)| = |V(C_2)| = 1$, wäre $C_1 \cup C_2$ eine Clique.) Seien C_1 und C_2 so gewählt, dass $|C_1|$ maximal ist. Damit existiert ein $u \in C_1$, $u \neq v$ mit $\{u, w\} \notin E$, sonst wäre $C_1 \cup \{w\}$ eine Clique, was ein Widerspruch zur Maximalität von C_1 ist. Damit gilt, dass $\{u, v\} \in E$, $\{v, w\} \in E$, aber $\{u, w\} \notin E$.



□

Algorithmus CE

Eingabe: Graph $G = (V, E)$, $k \in \mathbb{N}$.

Schritt 1: Falls G die disjunkte Vereinigung von Cliquen ist $\Longrightarrow$ Ausgabe JA

Schritt 2: Falls $k \leq 0$ $\Longrightarrow$ zurück aus diesem Aufruf mit Ausgabe NEIN

Schritt 3: Finde $u, v, w \in V$ mit $\{u, v\}, \{v, w\} \in E$ und $\{u, w\} \notin E$ (diese existieren nach der Beobachtung oben).

Rufe den Algorithmus rekursiv auf den Instanzen

$$\Big((V, E - \{\{u, v\}\}), k - 1\Big), \Big((V, E - \{\{v, w\}\}), k - 1\Big), \Big((V, E \cup \{\{u, w\}\}), k - 1\Big)$$

auf.

Theorem 4. *Der Algorithmus CE ist ein k-parametrisierter Algorithmus für Cluster-Editing.*

Beweis. Die Laufzeit ist bestimmt durch die Rekurrenz $T(k) = 3 \cdot T(k - 1)$.
$\Longrightarrow$ $T(k) = 3^k$ Aufrufe.
Schritt 1 braucht $O(|E|)$ Zeit, das Finden der Knoten in Schritt 3 braucht $O(|V|^3)$.
$\Longrightarrow$ Die Gesamtlaufzeit ist in $O(n^3 \cdot 3^k)$ für die Eingabelänge n. □

3. Methode: Dynamische Programmierung

Idee. Lösung aus Lösungen für Teilprobleme zusammensetzen, Teilprobleme so wählen, dass ihre Anzahl nur abhängig vom Parameter ist.

Beispiel 4 (Rucksackproblem mit Parameter Rucksackgrösse).
$\Longrightarrow$ analog zu dem pseudopolynomiellen Algorithmus, wobei aber für jedes mögliche Gewicht (statt für jeden möglichen Nutzen) ein Tripel gespeichert wird.

Beispiel 5 (Steinerbaum-Problem (STP)).

Definition 5. Das STP ist das folgende Optimierungsproblem.

Eingabe: Ungerichteter Graph $G = (V, E)$ mit Kantenkosten $c\colon E \to \mathbb{N}$ und Menge $S \subseteq V$ von *Terminalen*.

zulässige Lösungen: Jeder Teilbaum T von G, der alle Knoten aus S enthält (*Steinerbaum*).

Kosten: $\text{cost}(T, (G, c, S)) = c(T)$ (Summe der Kantenkosten in T).

Ziel: Minimierung.

Beobachtung. *Wenn die Menge $Y \subseteq N$ der verwendeten Nichtterminale gegeben ist ($N = V - S$), reicht es aus, einen minimalen Spannbaum auf $S \cup Y$ zu berechnen.*
$\Longrightarrow$ *Für den Parameter $k = |N|$ gibt es einen einfachen k-parametrisierten Algorithmus (vollständige Suche über alle $Y \subseteq N$).*
***Aber**: $k = |S|$ ist ein natürlicher Parameter, der in Anwendungen eher hilfreich ist.*

Ziel. fpt-Algorithmus für diesen Parameter.

$\Longrightarrow$ Dynamische Programmierung:
Teilprobleme definiert durch alle Teilmengen von S: Für jede Teilmenge $X \subseteq S$ und jeden Knoten $w \in V - X$ berechne zwei Steinerbäume für $X \cup \{w\}$

- mit minimalen Kosten,
- minimal mit w als inneren Knoten.

Definition 6. Sei $G = (V, E)$ ein Graph mit Kantenkosten $c\colon E \to \mathbb{N}$. Seien $v, w \in V$. Dann bezeichnet $p(v, w)$ die Kosten eines kürzesten Pfades von v nach w in G.

Definition 7. Sei $G = (V, E)$, $c\colon E \to \mathbb{N}$, $S \subseteq V$ eine STP-Instanz. Seien $X \subseteq S$, $v \in V - X$. Wir bezeichnen mit $g(X)$ die Grösse (Summe von Kantenkosten) eines minimalen Steinerbaums für X und mit $g_{\text{in}}(X, v)$ die Grösse eines minimalen Steinerbaums für $X \cup \{v\}$, der v als inneren Knoten enthält.

Beobachtung. *Für alle X, v gilt $g_{\text{in}}(X, v) \geq g(X \cup \{v\})$.*

Lemma 1 (Rekursive Berechnung von g und g_{in}). *Seien $X \subseteq S$, $X \neq \emptyset$ und $v \in V - X$. Dann gelten*

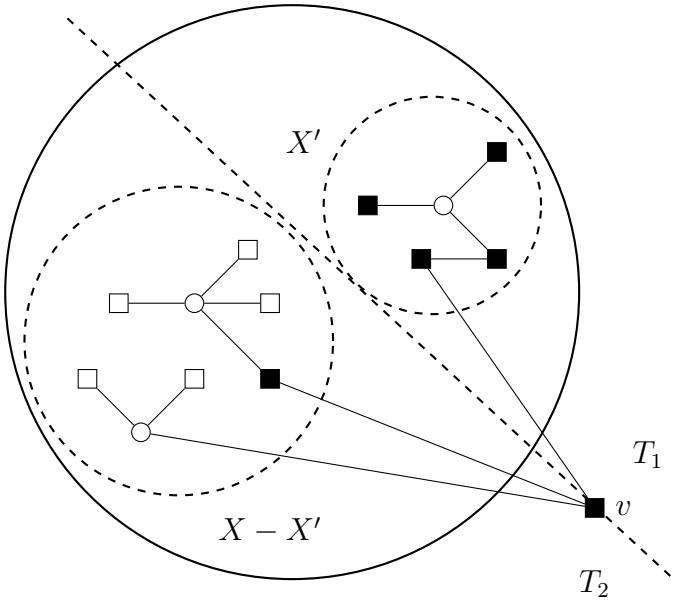
$$g_{\text{in}}(X, v) = \min_{\emptyset \neq X' \subsetneq X} \{g(X' \cup \{v\}) + g((X - X') \cup \{v\})\} \tag{1}$$

und

$$g(X \cup \{v\}) = \min\left\{\min_{w \in X}\{p(v, w) + g(X)\}, \min_{w \in V - X}\{p(v, w) + g_{\text{in}}(X, w)\}\right\}. \tag{2}$$

Beweis. Wir zeigen die beiden Gleichungen einzeln.

(1): Sei T ein Steinerbaum der Grösse $g_{\text{in}}(X, v)$, in dem v einen Grad ≥ 2 hat. T kann wie folgt zerlegt werden für jedes $X' \subseteq X$:



T_1 ist ein Steinerbaum für $X' \cup \{v\}$, T_2 ist ein Steinerbaum für $(X - X') \cup \{v\}$. Daraus folgt (1) unmittelbar.

(2): Sei T ein minimaler Steinerbaum für $X \cup \{v\}$. Für ein Blatt u in T sei P_u der längste Pfad in T von u ausgehend, in dem alle inneren Knoten Grad 2 haben und Nichtterminale sind.

Unterscheide drei Fälle.

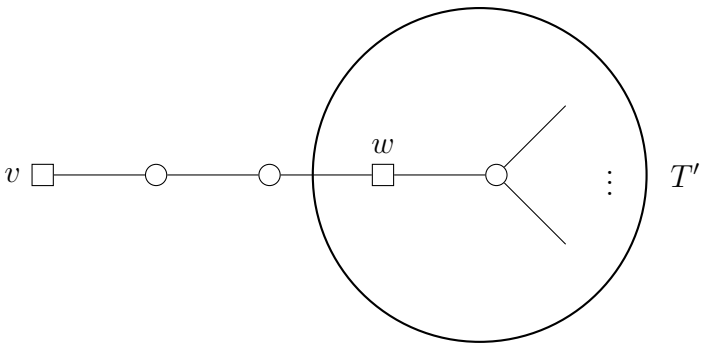
1. Fall: v hat Grad ≥ 2 in T.

$\implies$ $g(X \cup \{v\}) = g_{\text{in}}(X, v) = p(v, v) + g_{\text{in}}(X, v)$

$\implies$ dieser Fall wird durch das zweite Minimum in (2) abgedeckt.

2. Fall: v ist ein Blatt in T und P_v endet im Terminal w.

T:



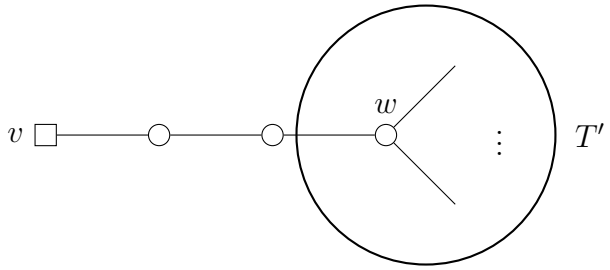
$\Longrightarrow$ $\text{cost}(T) = p(v,w) + \text{cost}(T')$, wobei T' ein minimaler Steinerbaum für X ist.

(T' muss minimal für X sein: Angenommen, es gäbe einen kleineren Steinerbaum $\tilde{T}$ für X, $\tilde{T}$ enthält ebenfalls w, also ist auch $P_v \cup \tilde{T}$ ein Steinerbaum für $X \cup \{v\}$, aber mit kleineren Kosten. Das ist ein Widerspruch zur Minimalität von T.)

$\Longrightarrow$ Dieser Fall ist durch das erste Minimum in (2) abgedeckt.

3. Fall: v ist ein Blatt in T und P_v endet im Nichtterminal w.

T:



$\Longrightarrow$ T' ist ein minimaler Steinerbaum für $X \cup \{w\}$, in dem w Grad ≥ 2 hat.

$\Longrightarrow$ Das zweite Minimum in (2) behandelt diesen Fall. □

Dreyfus-Wagner-Algorithmus für STP

Eingabe: Graph $G = (V, E)$, $c\colon E \to \mathbb{N}$, $S \subseteq V$.

Schritt 1: Berechne $p(v, w)$ für alle $v, w \in V$.

Schritt 2: Für alle $x, y \in S$, $x \neq y$, setze $g(\{x, y\}) \coloneqq p(x, y)$.

Schritt 3: (Rekursion)

Für alle $i = 2, \ldots, |S| - 1$:

Für alle $X \subseteq S$ mit $|X| = i$ und für alle $v \in V - X$:

$$g_{\text{in}}(X, v) \coloneqq \min_{\emptyset \neq X' \subsetneq X} \{g(X' \cup \{v\}) + g((X - X') \cup \{v\})\} \tag{1}$$

$$g(X \cup \{v\}) \coloneqq \min\left\{\min_{w \in X}\{p(v, w) + g(X)\}, \min_{w \in V - X}\{p(v, w) + g_{\text{in}}(X, w)\}\right\} \tag{2}$$

Ausgabe: $g(S)$.

Bemerkung. In dieser Form berechnet der Algorithmus nur die *Grösse* eines minimalen Steinerbaums. Es ist aber einfach, zusätzlich in jedem Schritt noch den entsprechenden Baum abzuspeichern.

Theorem 5. *Der Dreyfus-Wagner-Algorithmus berechnet die Grösse eines minimalen Steinerbaums in einer Zeit in* $O(n^2 \log n + n \cdot m + 3^k \cdot n + 2^k \cdot n^2)$, *wobei* $n = |V|$, $m = |E|$, $k = |S|$, *ist also ein* k*-parametrisierter Algorithmus für STP.*

Beweis. Wir beweisen die Korrektheit und Laufzeitschranke wieder einzeln.

1. Korrektheit: Folgt direkt aus dem Lemma.

2. Laufzeit: Wir analysieren die Schritte des Algorithmus einzeln.

   Schritt 1 ist in $O(n^2 \log n + n \cdot m)$ durchführbar (z. B. n-mal den Dijkstra-Algorithmus anwenden).

   Schritt 2 ist offensichtlich in $O(k^2)$ möglich.

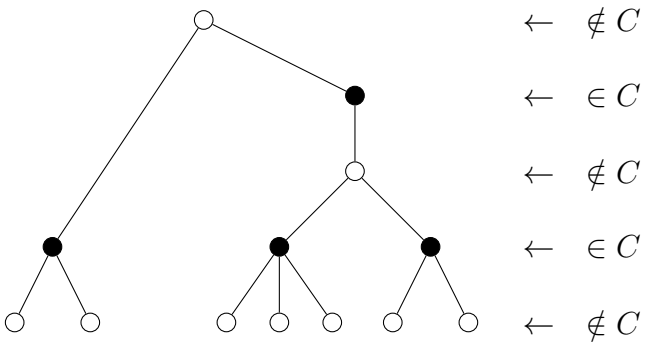
   Schritt 3 In der Rekursion (1) wird über alle Möglichkeiten iteriert, v, X' und X zu wählen. Da jedes Terminal zu genau einer der Mengen X', $X - X'$ oder $S - X$ gehört, und jede Aufteilung der Terminale auf diese drei Mengen genau eine Möglichkeit beschreibt, X' und X zu wählen, werden in (1) $O(n \cdot 3^k)$ Möglichkeiten berechnet, jeweils mit konstantem Aufwand.

   Bei der Berechnung von (2) werden $O(2^k \cdot n)$ Paare (X, v) betrachtet. Für jedes dieser Paare ist $O(n)$ Zeit nötig (Iteration über alle $w \in V$), also insgesamt Zeitaufwand $O(2^k \cdot n^2)$ für (2). □

4. Methode: Baumzerlegung von Graphen

Motivation. Viele schwere Graphprobleme werden einfach, wenn man die Eingaben auf Bäume einschränkt.

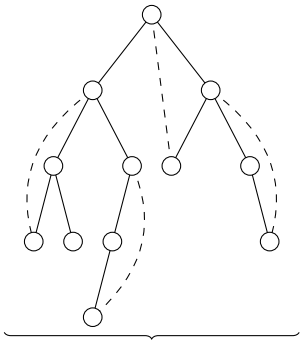
Beispiel 6. VC auf Bäumen: Von den Blättern aus einmal den Baum durchsuchen:



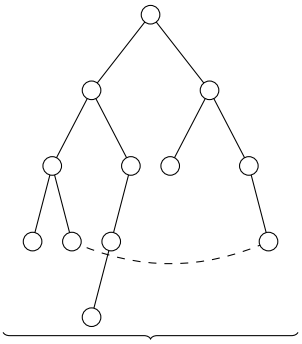
Idee. Definiere ein Mass für die „Baumähnlichkeit“ von Graphen und erweitere die Baum-Algorithmen auf „baumähnliche“ Graphen.

Frage. Was ist „Baumähnlichkeit“?

1. Ansatz. Anzahl der Kanten, die man entfernen muss, um einen Baum zu erhalten.
$\Longrightarrow$ Spiegelt nicht die Schwierigkeit wider, eine Lösung z. B. für VC zu berechnen:



vs.



immer noch baumähnlich,
für Bestimmung eines VC
müssen nur lokale Bereiche
(auf ≤ 3 Leveln) betrachtet
werden

keine Lokalität mehr,
VC-Algorithmus für Bäume
ist schwieriger anzupassen

$\Longrightarrow$ **Forderungen für „Baumähnlichkeit“**

- Fasse Knoten zu „Bags“ zusammen, die zusammen betrachtet werden müssen.
  $\Longrightarrow$ Jede Kante ist in einem Bag enthalten.
- Erhalte die Lokalität eines Knotens.
  $\Longrightarrow$ Alle Bags, die einen Knoten x enthalten, hängen zusammen.

Definition 8. Sei $G = (V, E)$ ein Graph, sei I eine beliebige Indexmenge. Eine *Baumzerlegung* von G ist ein Paar $(\{X_i \mid i \in I\}, T)$, wobei $X_i \subseteq V$ für alle $i \in I$ (die X_i werden *Bags* genannt) und T ein Baum ist mit den X_i als Knoten, so dass folgende Bedingungen gelten:

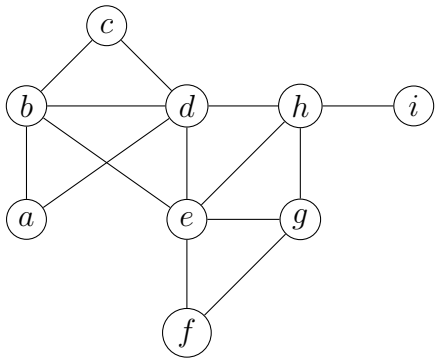
(1) $\bigcup_{i \in I} X_i = V$,

(2) für alle $\{u, v\} \in E$ existiert ein $i \in I$, so dass $u, v \in X_i$ sind,

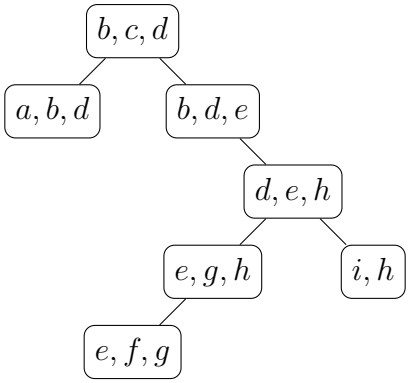
(3) für jeden Knoten $v \in V$ bilden die X_i, die v enthalten, einen Teilbaum von T.

Die *Weite* von $(\{X_i \mid i \in I\}, T)$ ist $\max\{|X_i| \mid i \in I\} - 1$.
Die *Baumweite* $\mathrm{tw}(G)$ von G ist die minimale Weite über alle Baumzerlegungen von G.

Beispiel 7. G:



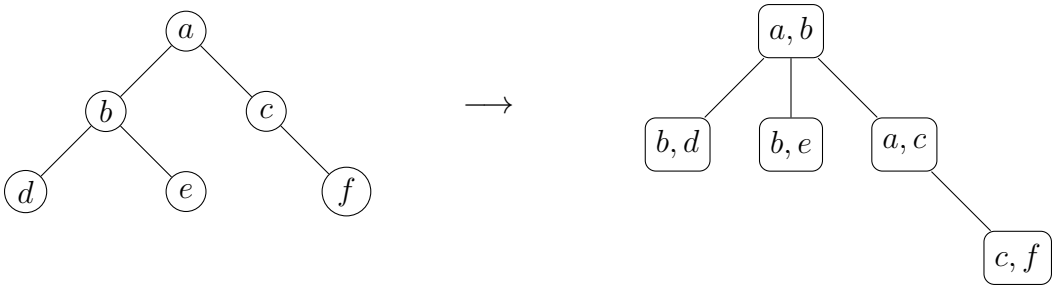
Baumzerlegung von G:



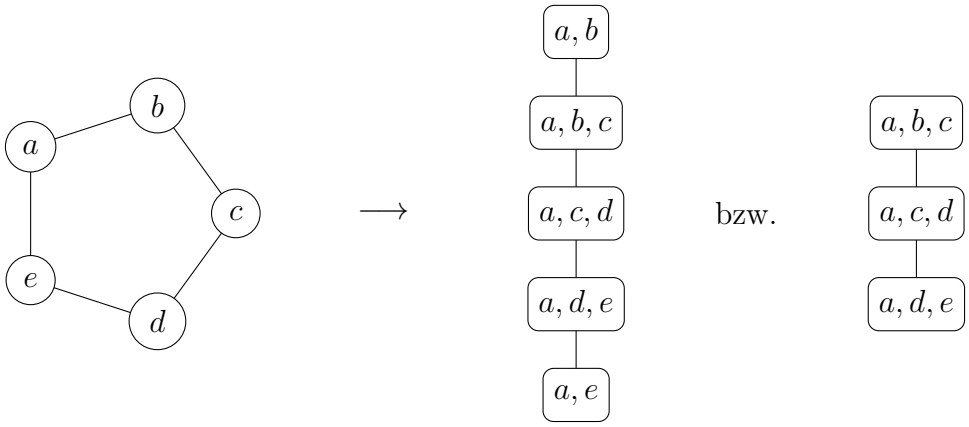
Die Weite dieser Baumzerlegung ist 2.

Beobachtung. *Bäume haben Baumweite 1.*

Beispiel 8.



Beispiel 9 (Baumweite eines Kreises).



$\Longrightarrow$ Ein Knoten (hier: a) muss in allen Bags vorkommen.

Anschaulich: Bei einer Berechnung, die einmal um den Kreis herumläuft, wird die Information über den Startknoten am Ende noch benötigt.

Beobachtung. *Sei $G = (V, E)$ ein Graph, $C \subseteq V$ eine Clique in G mit $|C| = k$. Dann ist die Baumweite von G mindestens $k - 1$.*

Beweis. Sei $V = \{1, 2, \ldots, n\}$, $C = \{1, 2, \ldots, k\}$. Wir zeigen mittels Induktion über k, dass alle Knoten aus C gemeinsam in einer Bag vorkommen müssen.

Induktionsanfang ($k = 2$):

Die Behauptung folgt direkt aus der Bedingung (2) für Baumzerlegungen.

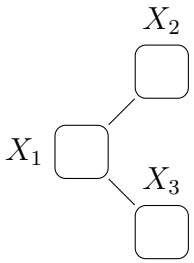
Induktionsschritt ($k \geq 3$):

Induktionsvoraussetzung: Die Behauptung gelte für alle Cliquen der Grösse $\leq k - 1$.

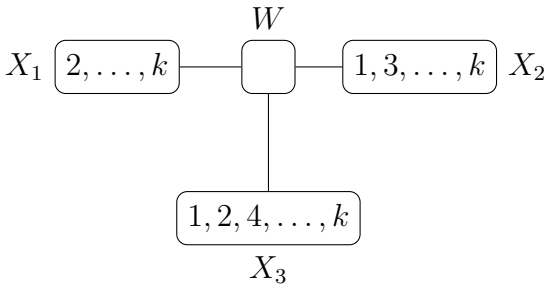
Betrachte eine Clique C der Grösse k. Diese enthält die Cliquen $C_i = \{1, \ldots, k\} - \{i\}$ der Grösse $k - 1$, die nach Induktionsvoraussetzung jeweils gemeinsam in einer Bag X_i vorkommen.

X_1 und X_2 sind in T durch einen Pfad verbunden (wegen der gemeinsamen Knoten $3, \ldots, k$), der X_3 nicht enthält (weil $3 \notin X_3$). Analog sind X_1 und X_3 durch einen Pfad verbunden, der X_2 nicht enthält und X_2 und X_3 durch einen Pfad, der X_1 nicht enthält.

Betrachte die Pfade $X_1 \to X_2$ und $X_1 \to X_3$. Wenn diese disjunkt sind, dann liegt X_1 auf dem Pfad von X_2 nach X_3. Das ist ein Widerspruch.



Wenn nicht, dann existiert eine Bag W, die zu allen drei Bags X_1, X_2, X_3 verbunden ist:



□

Da W auf dem Pfad von X_1 nach X_2 liegt, enthält W alle Knoten aus $X_1 \cap X_2$, also $3, \ldots, k$. Da W auf dem Pfad von X_1 nach X_3 liegt, enthält W alle Knoten aus $X_1 \cap X_3$, also $2, 4, \ldots, k$. Und da W auf dem Pfad von X_2 nach X_3 liegt, enthält W alle Knoten aus $X_2 \cap X_3$, also $1, 4, \ldots, k$.

$\implies$ W enthält alle Knoten $1, \ldots, k$.

Theorem 6 (Hier ohne Beweis). *Die Bestimmung der Baumweite eines gegebenen Graphen ist NP-schwer.*

Bemerkung. Damit erfüllt die Baumweite nicht die erste Bedingung (Polynomzeit-Berechenbarkeit), die wir an eine Parametrisierung gestellt haben. Allerdings lässt sich die Baumweite approximieren, d. h. für einen Graphen G mit $\mathrm{tw}(G) = k$ lässt sich in Polynomzeit eine Baumzerlegung der Weite $O(k \cdot \log(k))$ finden.

Ziel. $\mathrm{tw}(G)$-parametrisierter Algorithmus für VC.

Theorem 7. *Für einen Graphen $G = (V, E)$ mit gegebener Baumzerlegung $(\{X_i \mid i \in I\}, T)$ der Weite w kann man einen optimalen VC in Zeit $O(2^w \cdot w \cdot |I|)$ berechnen.*

Beweis-Idee. Dynamische Programmierung: Betrachte für jede Bag alle möglichen VC für die Knoten dieser Bag. Füge diesen Teil-VC in den Baum T bottom-up von den Blättern aus zusammen. □

Beweis. Wir wenden diese Idee nun formal an.

Schritt 1: Für jede Bag $X_i = \{x_{i_1}, \ldots, x_{i_{n_i}}\}$ der Grösse n_i berechne eine Tabelle A_i der Grösse $(n_i + 1) \times 2^{n_i}$:

x_{i_1}	x_{i_2}	$\ldots$	$x_{i_{n_i}}$	m_i
0	0	$\ldots$	0	
0	0	$\ldots$	1	
		$\vdots$		
1	1	$\ldots$	0	
1	1	$\ldots$	1	

Jede Zeile entspricht einem potentiellen VC der Knoten aus X_i: Eine 1 entspricht „im VC enthalten“ und eine 0 ist „nicht im VC enthalten“.

Die Spalte m_i soll am Ende die minimale Grösse eines VC für G enthalten, der auf X_i mit der ausgewählten Knotenmenge übereinstimmt.

Formal: Jede Zeile lässt sich darstellen als $C_i \colon X_i \to \{0, 1\}$ und

$$m_i(C_i) \coloneqq \min\{|V'| \mid V' \subseteq V \text{ ist VC für } G, \text{ so dass für alle } x \in X_i \text{ gilt } x \in V' \iff C_i(x) = 1\}.$$

Wir nennen C_i auch *Färbung* von X_i, die Knoten im VC werden mit 1 gefärbt, alle anderen mit 0. Nicht jede mögliche Auswahl von Knoten aus X_i lässt sich zu einem gültigen VC ergänzen. Wir nennen eine solche Färbung *ungültig*.

Insbesondere ist eine Färbung C_i ungültig, wenn es $x, y \in X_i$ gibt mit $\{x, y\} \in E$ und $C_i(x) = C_i(y) = 0$.

Initialisierung der Tabellen: Für alle X_i und alle Zeilen C_i in X_i:

$$m_i(C_i) \coloneqq \begin{cases} |\{x \in X_i \mid C_i(x) = 1\}|, & \text{falls } C_i \text{ gültig ist,} \\ \infty, & \text{sonst.} \end{cases}$$

Dynamische Programmierung: Sei X_i der Vorgängerknoten von X_j in T. Berechne die Werte m_i neu wie folgt:

Sei $X_i = \{z_1, \ldots, z_s, v_1, \ldots, v_{t_i}\}$ und $X_j = \{z_1, \ldots, z_s, u_1, \ldots, u_{t_j}\}$ mit $X_i \cap X_j = \{z_1, \ldots, z_s\}$.

Sei C_Z eine Färbung von $Z = \{z_1, \ldots, z_s\}$. Eine *Erweiterung* von C_Z auf X_i (bzw. X_j) ist eine Färbung $C_i \colon X_i \to \{0, 1\}$, so dass $C_i|_Z = C_Z$ (analog für C_j).

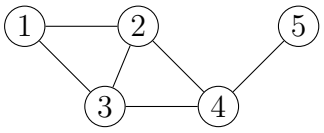
Für jedes C_Z und jede Erweiterung C_i setze

$$m_i(C_i) \coloneqq m_i(C_i) + \min\{m_j(C_j) \mid C_j \colon X_j \to \{0,1\} \text{ ist Erweiterung von } C_Z\} - |\{x \in Z \mid C(x) = 1\}|$$

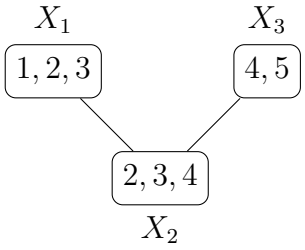
$\Longrightarrow$ minimale Ergänzung des VC C_i um die Knoten aus X_j, letzter Term der Formel zieht doppelt gezählte Knoten wieder ab.

Beispiel 10.

G:



Baumzerlegung:



Initialisierung:

A_1:

1	2	3	m_1
0	0	0	∞
0	0	1	∞
0	1	0	∞
0	1	1	2
1	0	0	∞
1	0	1	2
1	1	0	2
1	1	1	3

A_2:

2	3	4	m_2
0	0	0	∞
0	0	1	∞
0	1	0	∞
0	1	1	2
1	0	0	∞
1	0	1	2
1	1	0	2
1	1	1	3

A_3:

4	5	m_3
0	0	∞
0	1	1
1	0	1
1	1	2

Aktualisierung der Tabellen: Aus A_1 und A_2 folgt $Z = \{2, 3\}$.

A_2:

2	3	4	m_2	VC
0	1	1	$2 + 2 - 1 = 3$	$\{1, 3, 4\}$
1	0	1	$2 + 2 - 1 = 3$	$\{1, 2, 4\}$
1	1	0	$2 + 2 - 2 = 2$	$\{2, 3\}$
1	1	1	$3 + 2 - 2 = 3$	$\{2, 3, 4\}$

Aus A_2 und A_3 folgt $Z = \{4\}$.

A_2:

2	3	4	m_2	VC
0	1	1	$3 + 1 - 1 = 3$	$\{1, 3, 4\}$
1	0	1	$3 + 1 - 1 = 3$	$\{1, 2, 4\}$
1	1	0	$2 + 1 - 0 = 3$	$\{2, 3, 5\}$
1	1	1	$3 + 1 - 1 = 3$	$\{2, 3, 4\}$

1. Korrektheit des Algorithmus:

   - Wegen der Bedingung $V = \bigcup_{i \in I} X_i$ wird jeder Knoten betrachtet.
   - Wegen der Bedingung, dass jede Kante in einer Bag enthalten ist, ergeben sich nach dem Aussortieren der ungültigen Färbungen nur noch gültige VC für die jeweils betrachteten Teilbäume von T.
   - Die dritte Eigenschaft der Baumzerlegung (alle Bags mit Knoten x sind miteinander verbunden) stellt sicher, dass die berechneten VC konsistent sind (jeder Knoten bekommt eine eindeutige Farbe).

2. Laufzeit:

   - Maximale Grösse einer Tabelle: $O(2^w \cdot w)$
   - $|I|$ Updates
   - Das Update einer Tabelle kann durch die Verwendung geeigneter Datenstruktur linear in der Grösse der grösseren Tabelle erfolgen.

$\implies$ Die Gesamtlaufzeit ist in $O(2^w \cdot w \cdot |I|)$. □

Korollar 1. *Das Vertex-Cover-Problem ist* $\mathrm{tw}(G)$*-parametrisierbar.*

Beweis. Sei $G = (G, k)$ eine Eingabe für VC.

1) Approximiere $\mathrm{tw}(G)$, erhalte Baumzerlegung der Weite $w \in O\Big(\mathrm{tw}(G) \cdot \log(\mathrm{tw}(G))\Big)$

2) Verwende den Algorithmus von oben

$\implies$ Laufzeit in $2^{O(\mathrm{tw}(G) \cdot \log(\mathrm{tw}(G))} \cdot O\Big(\mathrm{tw}(G) \cdot \log(\mathrm{tw}(G)) \cdot |I|\Big)$. □

Ausblick 1. Subexponentieller Algorithmus für VC in planaren Graphen.

Theorem 8. *VC kann in planaren Graphen in Zeit* $2^{O(\sqrt{k})} \cdot n$ *gelöst werden, wobei* $n = |V|$.

Beweis-Idee.

1) VC (in allgemeinen Graphen) hat Kern der Grösse $O(k)$.
   $\implies$ Wende den entsprechenden Algorithmus an, der Graph bleibt dabei planar.

2) Ein planarer Graph kann mit Hilfe von Separatoren der Grösse $O(\sqrt{k})$ in Teile der Grösse $O(\sqrt{k})$ zerlegt werden.
   $\implies$ hat Baumweite $\in O(\sqrt{k})$.

3) Berechne $O(\sqrt{k})$-Baumzerlegung.

4) Wende $\mathrm{tw}(G)$-parametrisierten Algorithmus an. □

Grenzen der Parametrisierbarkeit

Ziel. Zeigen, dass es Probleme gibt, die keinen fpt-Algorithmus erlauben (unter gewissen komplexitätstheoretischen Annahmen).

Definition 9. Sei U ein Entscheidungsproblem, beschrieben durch das Eingabealphabet Σ und die Sprache L_U der JA-Instanzen. Sei $\mathrm{Par}\colon \Sigma^* \to \mathbb{N}$ eine Parametrisierung von U. Dann nennen wir (U, Par) ein *parametrisiertes Problem*.
Ein parametrisiertes Problem (U, Par) lässt sich eindeutig beschreiben durch die Menge $L_{U,\mathrm{Par}} \subseteq \Sigma^* \times \mathbb{N}$, so dass $(x,k) \in L_{U,\mathrm{Par}} \iff x \in L_U$ und $k = \mathrm{Par}(x)$.

Definition 10. Seien $L, L' \subseteq \Sigma^* \times \mathbb{N}$ zwei parametrisierte Probleme. Wir sagen, dass sich L auf L' reduzieren lässt mit Hilfe einer *parametrisierten Standard-Reduktion*, wenn es Funktionen

$$f\colon \mathbb{N} \to \mathbb{N}, \quad k \mapsto k',$$
$$g\colon \mathbb{N} \to \mathbb{N}, \quad k \mapsto k'',$$
$$h\colon \Sigma^* \times \mathbb{N} \to \Sigma^*, \quad (x,k) \mapsto x'$$

gibt, so dass

(1) $h(x,k) = x'$ ist berechenbar in Zeit $k'' \cdot |(x,k)|^c$ für ein $c \geq 0$,

(2) $(x,k) \in L \iff (x',k') \in L'$.

Anschaulich:

- Reduktion lässt sich in parametrisierter Zeit durchführen.
- Parameter der neuen Instanz hängt nur vom Parameter der alten Instanz ab.

Definition 11. Das *gewichtete 2-SAT-Problem* (*W2SAT*) ist das folgende parametrisierte Problem:

Eingabe: Eine Boolesche Formel F in 2-KNF und ein $k \in \mathbb{N}$.

Frage: Gibt es eine erfüllende Belegung für F mit Gewicht genau k?

Parametrisierung: $\mathrm{Par}(F,k) = k$.

Dabei ist das *Gewicht* einer Belegung die Anzahl der auf 1 gesetzten Variablen.

Definition 12.

- Die Klasse $W[1]$ enthält alle parametrisierten Probleme, die sich auf W2SAT reduzieren lassen mit Hilfe einer paramerisierten Standard-Reduktion.
- Ein parametrisiertes Problem P heisst *W[1]-schwer*, wenn sich W2SAT auf P reduzieren lässt mit Hilfe einer parametrisierten Standard-Reduktion.
- $W[1]$-*vollständig*: in $W[1]$ und $W[1]$-schwer

- Die Klasse *FPT* enthält alle parametrisierten Probleme, für die ein fpt-Algorithmus existiert.

Zusammenhang zur klassischen Komplexitätstheorie:

Theorem 9 (Hier ohne Beweis). *Falls $W[1] = FPT$, dann kann 3-SAT auf einer Eingabe F mit n Variablen in Zeit $2^{o(n)} \cdot |F|^{O(1)}$ gelöst werden.*

Bemerkung. Dies ist eine stärkere Bedingung als 3SAT $\notin P$. So gesehen ist die $W[1]$-Vollständigkeit eines Problems eine schwächere Aussage als die NP-Vollständigkeit.
Aber die meisten Forscher nehmen an, dass ein solcher Algorithmus für 3SAT nicht möglich ist.
$\Longrightarrow$ bekannt unter dem Namen *Exponentialzeit-Hypothese* (exponential-time hypothesis, *ETH*).

Beispiel 11. Beispiel eines $W[1]$-schweren Problems: Independent Set

Definition 13. Parametrisierte Version des *Independent-Set-Problems* (IS):

Eingabe: Graph $G = (V, E)$ und $k \in \mathbb{N}$.

Frage: Gibt es ein IS $X \subseteq V$ (d. h. $\{v, w\} \notin E$ für alle $v, w \in X$) der Grösse exakt k?

Parametrisierung: $\mathrm{Par}(G, k) = k$.

Definition 14. *Gewichtetes antimonotones 2SAT-Problem* (WAM2SAT):
Einschränkung von W2SAT auf Formeln, in denen nur Klauseln der Form $\overline{x} \vee \overline{y}$ für Variablen x, y vorkommen.

Theorem 10 (Hier ohne Beweis). *WAM2SAT ist $W[1]$-vollständig.*

Theorem 11. *IS ist $W[1]$-vollständig.*

Beweis-Idee. Es müssen zwei Eigenschaften von IS gezeigt werden.

(1) IS $\in W[1]$:

Idee: Stelle den Graphen als Formel dar, die Knoten als Variablen, die Kanten als Klauseln: Sei $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}$, $E = \{e_1, \ldots, e_m\}$. Definiere $\Phi_G = F_1 \wedge \ldots \wedge F_m$ durch $F_i = (\overline{v_{i_1}} \vee \overline{v_{i_2}})$, falls $e_i = \{v_{i_1}, v_{i_2}\}$. Dann beschreibt Φ_G genau die IS-Bedingung: Für jede Kante darf höchstens einer der beiden Endpunkte in das IS aufgenommen werden.

$\Longrightarrow$ Belegung mit Gewicht k entspricht IS der Grösse k.

(2) IS ist $W[1]$-schwer:

Reduktion aus (1) funktioniert auch in der umgekehrten Richtung: Aus der antimonotonen Formel kann der Graph konstruiert werden. □

Literaturhinweise

Dieses Skript orientiert sich im Wesentlichen an dem folgenden Buch:

☞ Rolf Niedermeier: *Invitation to Fixed-Parameter Algorithms*, Oxford University Press, 2006.

Die Darstellung des Dreyfus-Wagner-Algorithmus wurde dem folgenden Buch entnommen:

☞ Hans Jürgen Prömel, Angelika Steger: *The Steiner Tree Problem*, Vieweg, 2002.

Weitere Darstellungen der parametrisierten Komplexität finden sich in:

☞ Marek Cygan et al.: *Parameterized Algorithms*, Springer, 2015.

☞ Rod Downey, Mike Fellows: *Fundamentals of Parameterized Complexity*, Springer, 2013.

☞ Rod Downey, Mike Fellows: *Parameterized Complexity*, Springer, 1999.

☞ Jörg Flum, Martin Grohe: *Parameterized Complexity Theory*, Springer, 2006.