

# Algorithmik für Schwere Probleme

thgoebel@ethz.ch

ETH Zürich, FS 2021

This documents is a **short** summary for the course *Algorithmik für Schwere Probleme* at ETH Zurich. It is intended as a document for quick lookup, e.g. during revision, and as such does not replace attending the lecture, reading the slides or reading a proper book.

We do not guarantee correctness or completeness, nor is this document endorsed by the lecturers. Feel free to point out any errata, either by mail or on Github.

# Contents

<b>1</b>	<b>Einführung</b>	<b>3</b>
1.1	Definitionen . . . . .	4
<b>2</b>	<b>Pseudopolynomielle Algorithmen</b>	<b>6</b>
<b>3</b>	<b>Parametrisierte Algorithmen</b>	<b>7</b>
3.1	Kernbildung . . . . .	8
3.1.1	Kronenzerlegung . . . . .	9
3.2	Tiefenbeschränkte Suchbäume . . . . .	10
3.3	Iterative Kompression . . . . .	10
3.4	Dynamische Programmierung für den Steinerbaum . . . . .	11
3.5	Baumzerlegung . . . . .	12
3.5.1	Beispiel: VCP parametrisiert mit Baumweite . . . . .	13

# 1 Einführung

## Konzepte

- NP-schwer vs. NP-vollständig
- Schwellwertsprache

**Polynomzeit-Reduzierbarkeit** Ein Entscheidungsproblem  $\Pi_1$  ist “polynomzeit-reduzierbar” auf ein anderes Entscheidungsproblem  $\Pi_2$ :

$$\iff \exists \text{ Algo } \mathcal{A} \text{ s.t. } \text{time}_{\mathcal{A}} \in \text{poly} \wedge \Pi_1(x) = \Pi_2(\mathcal{A}(x))$$

$$\iff \Pi_2 \text{ mindestens so schwer wie } \Pi_1$$

$$\iff \Pi_1 \text{ höchstens so schwer wie } \Pi_2$$

$$\iff \Pi_1 \preceq_P \Pi_2$$

**NP-schwer (NP-hard)** Ein Problem  $\Pi$  das “mindestens so schwer” ist wie alle Probleme in NP. D.h. alle Probleme in NP lassen sich auf  $\Pi$  reduzieren:

$$\forall \Pi' \in NP : \Pi' \preceq_P \Pi$$

$\Pi$  muss nicht notwendigerweise in NP liegen (d.h. kann schwerer sein)! Beispiel: das Halteproblem (nicht entscheidbar, daher  $\notin NP$ ).

**NP-vollständig (NP-complete)** Ein Problem  $\Pi$ , das in NP liegt und NP-schwer ist. “Repräsentativ” für die Menge NP, da sich alle Probleme aus NP darauf reduzieren lassen. Beispiel: Satisfiability-Problem SAT (Satz von Cook).

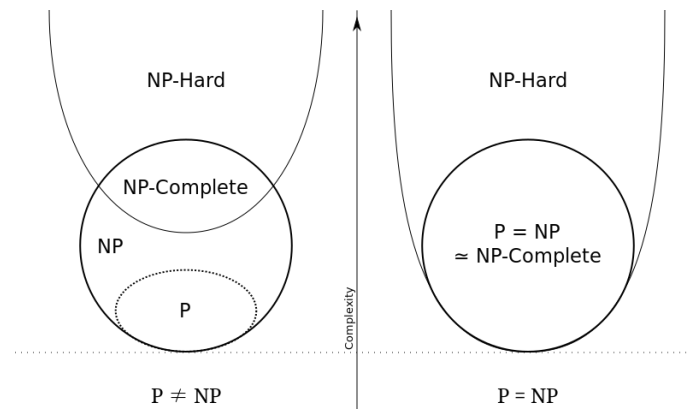


Figure 1: Mengendiagramm der Beziehungen (Quelle: Wikipedia)

**“Schwere Probleme”** NP-schwere Probleme, aber generell alle Probleme die sich nicht in Polynomzeit lösen lassen. **Sinnvollerweise gehen wir im Folgenden davon aus dass  $P \neq NP$ .**

Alle Instanzen unseres Problems sind deterministisch in Polynomzeit nicht lösbar. Mögliche Ansätze:

- a) nicht exakt sondern approximativ lösen (Approximationsalgorithmen)
- b) nicht deterministisch sondern nichtdeterministisch lösen (Randomisierte Algorithmen)

- c) nicht polynomiell sondern moderat exponentiell lösen<sup>1</sup>
- d) nicht alle sondern alle Instanzen mit einer bestimmten Struktur lösen (Parametrisierte Algorithmen)
- e) anderweitig zusätzliche Informationen über die Eingabe nutzen (Reoptimierung, Win-Win-Strategy)
- f) Heuristiken<sup>2</sup>

## 1.1 Definitionen

**Entscheidungsproblem**  $P = (L, U, \Sigma)$  wobei

- $\Sigma$  ein Alphabet
- $U \subseteq \Sigma^*$  die Menge der zulässigen Eingaben (als Wörter über dem Alphabet, als *Sprache*)
- $L \subseteq U$  die Menge der akzeptierten Eingaben (*JA-Instanzen*)

Ein Algorithmus  $\mathcal{A}$  *löst*  $P$  falls gilt:

$$\forall u \in U : A(x) = \begin{cases} 1 \text{ oder JA,} & \text{if } x \in L \\ 0 \text{ oder NEIN,} & \text{if } x \in U - L \end{cases}$$

**Vertex Cover Problem VC** “Der Hefepilz der parametrisierten Algorithmiker – ein Modellproblem.”

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

Ausgabe  $L$ : JA falls  $\exists C \subseteq V$  s.t.  $|C| \leq k$  mit  $\forall \{u, v\} \in E : u \in C \vee v \in V$ .

### Satisfiability-Problem SAT

Eingabe: CNF-Formel  $\Phi = C_1 \wedge \dots \wedge C_m$  mit Klauseln  $C_i$  über Variablen  $x_1, \dots, x_n$ .

Ausgabe: eine Variablen-Belegung die  $\Phi$  erfüllt.

Bei  $l$ -SAT enthält jede Klausel maximal  $l$  Literale.

**Optimierungsproblem**  $U = (L, M, cost, goal)$  wobei

- $L$  die Sprache der zulässigen Eingaben<sup>3</sup>
- $M : L \mapsto \Sigma^*$  so dass  $M(x)$  die Sprache der akzeptierten Lösungen für Eingabe  $x$
- $cost: \forall x \in L \forall y \in M(x) : cost(y, x) = \text{Kosten der Lösung } y \text{ für Eingabe } x$
- $goal \in \{\min, \max\}$  das Optimierungsziel
- $Opt_U(x) = goal\{cost(y, x) | y \in M(x)\}$  die Kosten einer optimalen Lösung für Eingabe  $x$

**Minimum Vertex Cover Problem MIN-VC** Wie VC, mit  $cost(C, G) = |C| = \text{Grösse des Vertex Covers}$  und  $goal = \min$ .

**MAX-SAT** Wie SAT, mit  $cost = \text{Anzahl belegte Variablen}$  und  $goal = \max$ .

<sup>1</sup>D.h. die Basis der Exponentiation ist klein, z.B.  $1.4^n$  statt  $2^n$ .

<sup>2</sup>Nachteil: Im Gegensatz zu den anderen Ansätzen ist hier die Qualität (Laufzeit, ...) nicht beweisbar.

<sup>3</sup>Oben noch  $U$ !

**Laufzeit** eines Algorithmus'  $\mathcal{A}$  auf Eingabe  $x$  ist  $\text{time}_{\mathcal{A}}(x)$  wobei  $\text{time}_{\mathcal{A}} : \mathbb{N} \mapsto \mathbb{N}$ . Die Laufzeit von  $\mathcal{A}$  in Abhängigkeit von der Grösse  $n$  der Eingabe ist:  $\text{time}_{\mathcal{A}}(n) = \max\{\text{time}_{\mathcal{A}}(x) \mid |x| = n, x \in L\}$ . Die Laufzeit wird in  $\mathcal{O}$ -Notation angegeben.

**Schwellwertsprache (threshold language)** definiert für ein Optimierungsproblem  $U$ :

$$Lang_U = \{(x, a) \in L \times \{0, 1\}^* \mid Opt_U(x) \leq Number(a)\}$$

wo  $Number(a)$  die Zahl mit der Binärdarstellung  $a$  (der *Schwellwert*) ist und  $goal = \min$ .

Beispiel:  $Lang_{MIN-VC} = VC$ . Aber  $Lang_{MAX-SAT} \neq SAT$  (da SAT leichter sein kann)!

$U$  heisst "NP-schwer" falls  $Lang_U$  NP-schwer ist (warum?).<sup>4</sup>

---

<sup>4</sup>Recall that NP-Schwere für Entscheidungsprobleme definiert ist. Über die Schwellwertsprache erweitern wir dies für Optimierungsprobleme.

## 2 Pseudopolynomielle Algorithmen

### Konzepte

- Pseudopolynomialität
- Stark NP-schwer

### Zahlproblem (integer value problem IVP)

Eingabe: darstellbar als Zahl  $x = x_1 \# \dots \# x_n$ ;  $x_i \in \{0, 1\}^*$  und interpretiert als Vektor  $Int(x) = (Number(x_1), \dots, Number(x_n))$ .

Beispiel: Travelling Salesman Problem TSP (via Adjazenzmatrix des Graphen).

Sei  $Max - Int(x) = \max\{Number(x_i)\}$  die grösste vorkommende Zahl (im Wert, nicht in der Darstellung).  $Max - Int(x)$  kann exponentiell in  $|x|$  sein.

**Pseudopolynomiell** Sei  $U$  ein Zahlproblem und  $\mathcal{A}$  ein Algorithmus der  $U$  löst.  $\mathcal{A}$  heisst *pseudopolynomiell* falls für alle Eingaben  $x$  ein Polynom  $p$  existiert, so dass

$$\text{time}_{\mathcal{A}}(x) \in \mathcal{O}\left(p(|x|, Max - Int(x))\right)$$

D.h. auf Eingaben mit “kleinen Zahlen” ist  $\mathcal{A}$  polynomiell.

### Rucksackproblem (Knapsack problem KP)

Eingabe  $I$ : Gewichte  $w_i \in \mathbb{N}^+$ , Kosten/Nutzen  $c_i \in \mathbb{N}^+$ , Limit/Kapazität  $b \in \mathbb{N}^+$ , wo  $i \in \{1, \dots, n\}$ .

Ausgabe: Indexmenge  $T \subseteq \{1, \dots, n\}$  s.t.  $\sum_{i \in T} w_i \leq b$

Kosten:  $\text{cost}(T, I) = \sum_{i \in T} c_i$

Ziel: max

Lösung mit DP: Iteration über alle Teilprobleme  $I_i$  und Speichern von Tripeln  $(k, W_{i,k}, T_{i,k}) = (\text{Nutzen, Gewicht, Indexmenge})$ , also Mengen  $T_{i,k}$  die exakt Nutzen  $k$  mit minimalen Gewicht  $W_{i,k}$  erreichen. In jeder Iteration behalte für jeden vorhandenen Nutzen ein Tripel mit minimalen Gewicht. Lese am Ende den maximal erreichten Nutzen  $k^*$  (und sein  $T_{n,k^*}$ ) aus.

Laufzeit:  $\mathcal{O}(|I|^2 \cdot Max - Int(I))$  da  $|I|$  Iterationen und jeder Schritt in  $\leq \sum_j^n c_j = |I| \cdot Max - Int(I)$ .

**h-beschränktes Teilproblem** Sei  $U$  ein Zahlproblem,  $h : \mathbb{N} \mapsto \mathbb{N}$  monoton nicht-fallend. Das *h-beschränkte Teilproblem von U* (*h-value-bounded subproblem*)  $Value(h) - U$  ist das Teilproblem mit Eingaben  $I$  für die gilt:  $Max - Int(I) \leq h(|I|)$ .

**Stark NP-schwer** Ein Zahlproblem  $U$  heisst *stark NP-schwer* falls ein Polynom  $p$  existiert, so dass  $Value(p) - U$  NP-schwer ist.

In anderen Worten:  $U$  ist stark NP-schwer wenn es auch dann NP-schwer ist wenn alle darin vorkommenden Zahlen “klein” sind.

Beispiel: TSP. Generell jedes gewichtete Graph-Optimierungsproblem wenn das ungewichtete Pendant NP-schwer ist (hier: Hamiltonkreisproblem HCP).

Theorem: Sei  $U$  stark NP-schwer. Dann existiert kein pseudopolynomieller Algorithmus für  $U$ .<sup>5</sup>

<sup>5</sup>Wie immer unter der Annahme dass  $P \neq NP$ .

### 3 Parametrisierte Algorithmen

#### Konzepte

- Parametrisierter Algorithmus, fpt
- Standard-Parametrisierung
- Kernbildung
- (Sichere) Datenreduktionsregel
- Kronenzerlegung
- Tiefenbeschränkte Suchbäume
- Iterative Kompression
- Baumzerlegung, Baumweite; *bramble*, *bramble number*
- Probleme: Edge Clique Cover, Cluster Editing, Steinerbaum, VCP via Baumweite

**Idee** Verallgemeinerung von pseudopolynomiellen Algorithmen: Laufzeit hängt von der Eingabegrösse nur polynomiell ab, aber darf extrem gross werden in der Grösse eines Parameters. Partitionierung in Problemklassen entlang des Parameters.

**Parametrisierung** Sei  $U$  ein Entscheidungsproblem,  $L$  die Sprache der Eingaben.  $\text{par} : L \mapsto \mathbb{N}$  ist eine *Parametrisierung* von  $U$  falls gilt:

- (i)  $\text{par}$  ist in Polynomzeit berechenbar
- (ii) Für unendlich viele  $k \in \mathbb{N}$  ist die Parameter- $k$ -Menge für  $U$

$$\text{Set}_U(k) := \{I \in L \mid \text{par}(I) = k\}$$

unendlich. Dies stellt sich dass  $\text{par}$  nichttrivial ist, z.B. nicht etwa  $\text{par}(I) = |I|$ . In anderen Worten, für jeden Parameterwert soll es beliebig viele Probleminstanzen geben.

Ein Algorithmus  $\mathcal{A}$  heisst *par-parametrisierter Polynomzeit-Algorithmus* für  $U$  falls gilt:

- (i)  $\mathcal{A}$  löst  $U$
- (ii)  $\exists$  Polynom  $p \exists$  (berechenbare) Funktion  $f : \mathbb{N} \mapsto \mathbb{N}$  so dass  $\forall I \in L$ :

$$\text{time}_{\mathcal{A}}(I) \leq f(\text{par}(I)) \cdot p(|I|)$$

Dann heisst  $U$  *fixed-parameter-tractable* bezüglich  $\text{par}$ .  $\mathcal{A}$  heisst auch *fpt-Algorithmus* für  $U$  bezüglich  $\text{par}$  und läuft in fpt-Zeit.

Beispiel: Sei  $U$  ein Zahlproblem und sei  $\text{Val}(I) := \max\{|x_i|\}$ . Es gilt  $\text{Max} - \text{Int}(I) \leq 2^{\text{Val}(I)}$  und ist  $\text{Val}$  eine Parametrisierung von  $U$ .

**Ansätze** Die *Standard-Parametrisierung* wählt als Parameter die Grösse der gewünschten Lösung (z.B. Grösse des VCs). Die *Strukturelle Parametrisierung* wählt eine bestimmte Eigenschaft der Eingabe (z.B. maximaler Knotengrad). Andere Ansätze betrachten wir im Folgenden.

### 3.1 Kernbildung

**Idee** Polynomielles Preprocessing, Datenreduktion, um die Instanz auf einen *Kern* zu verkleinern, der in seiner Grösse nur noch vom Parameter abhängt. Diesen Kern dann mit bekannten Algorithmen (oder brute force) lösen.

**Kernbildung (kernelisation)** Sei  $(U, \text{par})$  ein parametrisiertes Entscheidungsproblem,  $L$  die Sprache der JA-Instanzen von  $U$ . Ein *Kernbildungs-Algorithmus* für  $(U, \text{par})$  ist ein Polynomzeit-Algorithmus  $\mathcal{A}$  der jede Eingabe  $(I, k)$  in eine neue Eingabe  $(I', k')$  (den *Kern (kernel)*) transformiert so dass:

- (i)  $I \in L \iff I' \in L$  (Korrektheit)
- (ii)  $|I'| + k' \leq g(k)$  für  $g : \mathbb{N} \mapsto \mathbb{N}$  (neue Grösse nur abhängig vom alten Parameterwert  $k$ )

Eine *Reduktionsregel* von  $\mathcal{A}$  heisst *sicher*, wenn sie (i) erfüllt.

#### Beispiel: VCP

Reduktionsregel: Sei  $S \subseteq V$  ein VC von  $G$  so dass  $|S| \leq k$ . Dann enthält  $S$  alle Knoten mit  $\deg_G(v) > k$  (warum?). Dann gilt für ein solches  $v$ :

$$(G, k) \in VCP \iff (G - \{v\}, k - 1) \in VCP$$

Zusätzlich entferne alle isolierten Knoten mit  $\deg_G(v) = 0$  (sie decken keine Kanten ab).

Zu zeigen: wenn die Regel nicht mehr anwendbar ist, dann hat der verbleibende Graph  $G'$  entweder kein vertex cover, oder aber er ist "klein genug" (nur noch anhängig von  $k$ ), also ein Kern.

Beobachtung: Sei  $G$  ein Graph ohne isolierten Knoten, mit einem vertex cover der Grösse  $m$  und mit  $\max \deg_G(v) \leq k$ . Dann gilt  $|V| \leq m \cdot (k + 1)$  (warum?).<sup>6</sup>

Theorem: Diese Reduktionsregeln berechnen einen Kern der Grösse  $\mathcal{O}(k^2)$  in Zeit<sup>7</sup>  $\mathcal{O}(k \cdot n)$ .

Parametrisierter Algorithmus: Berechne einen Kern  $(G', k')$ . Wenn der Kern zu gross ist, geben NEIN aus. Andernfalls prüfe durch vollständige Suche ob ein VC mit  $|S'| \leq k'$  existiert.

Theorem: Dies ist ein fpt-Algorithmus bzgl. der Standard-Parametrisierung. Die Laufzeit beträgt  $\mathcal{O}(k \cdot n + k^{2k+2}) \subseteq \mathcal{O}(k^{2k+2} \cdot n)$ .

**Theorem** Sei  $(U, \text{par})$  ein parametrisiertes Entscheidungsproblem.

Ein fpt-Algorithmus für  $(U, \text{par})$  existiert  $\iff$  Ein Kernbildungsalgorithmus für  $(U, \text{par})$  existiert.

#### Edge Clique Cover Problem ECCP

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

Ausgabe: JA falls  $k$  Cliques<sup>8</sup>  $C_1, \dots, C_k$  existieren, so dass  $E = \bigcup_{i=1}^k E(C_i)$  (jede Kante ist in mind. einer Clique), sonst NEIN.

Reduktionsregeln:

- (i)  $(G, k) \xrightarrow{\text{isolierter Knoten } u} (G - \{u\}, k)$

<sup>6</sup>Falls  $|V| > k \cdot (k + 1)$ , dann wir wissen dass  $I \notin L$  und können NEIN ausgeben.

<sup>7</sup>Unter der Annahme dass die Knotengrade bereits gegeben sind, ansonsten  $\mathcal{O}(k \cdot n + m)$ .

<sup>8</sup>Clique: Knotenmenge so dass alle Knoten paarweise miteinander verbunden sind, d.h. einen vollständigen Teilgraphen bilden.



(ii)  $(G, k) \xrightarrow{\text{isolierte Kante } e=\{u,v\}} (G - \{u, v\}, k - 1)$

(iii)  $(G, k) \xrightarrow{\{u,v\} \in E \text{ s.t. } \{u,v\} \subsetneq N[u]=N[v]} (G - \{u\}, k)$

$N[u]$  ist die *geschlossene Nachbarschaft* von  $u$ , also alle benachbarten Knoten und  $u$  selbst.

Theorem: Das ECCP hat einen Kern mit  $\leq 2^k$  Knoten.

### 3.1.1 Kronenzerlegung

Ziel: statt einem quadratischen Kern suchen wir einen linearen Kern für das VCP.

Bisher genutzte Strukturen zur Reduktion: hoher Knotengrad, gleiche geschlossene Nachbarschaft.

**Kronenzerlegung (crown decomposition)** Die *Kronenzerlegung* von  $G = (V, E)$  ist eine Partitionierung  $V = C \cup H \cup B$  so dass:

(i)  $C \neq \emptyset$  ist ein independent set<sup>9</sup> in  $G$

(ii)  $N(C) = H$  (keine Kanten zwischen  $C$  und  $B$ )

(iii) Die Kanten zwischen  $C$  und  $H$  enthalten ein Matching  $M$  mit  $|M| = |H|$  ( $M$  *saturiert*  $H$ ).

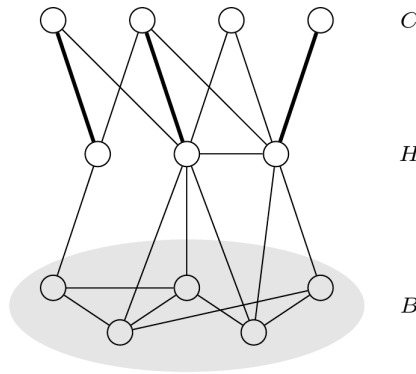


Figure 2: Kronenzerlegung: crown, head, body (Quelle: Vorlesung)

**Satz von König** Sei  $G$  ein ungerichteter bipartiter Graph, sei  $M$  ein Matching maximaler Kardinalität, sei  $S$  ein vertex cover minimaler Kardinalität. Dann gilt  $|M| = |S|$ .

Theorem:  $M$  und  $S$  können in Polynomzeit berechnet werden.

**Lemma** Sei  $G$  ungerichtet, ohne isolierte Knoten, mit  $|V| \geq 3k + 1$ . Dann existiert ein Polynomzeitalgorithmus der

(i) entweder eine Kronenzerlegung berechnet

(ii) oder ein Matching  $M$  mit  $|M| \geq k + 1$  findet.

Beweis siehe Buch, Kapitel 6.2, Seite 145f.

<sup>9</sup>Eine Menge von Knoten zwischen denen es keine Kante gibt.

## Reduktion für VCP

Lemma: Sei  $G$  ein Graph mit Kronenzerlegung  $V = C \cup H \cup B$  und sei  $k \in \mathbb{N}$ . Dann gilt:

$$(G, k) \in VCP \iff (G - (C \cup H), k - |H|) \in VCP$$

Theorem: Ein Kernbildungsalgorithmus mit obiger Reduktionsregel findet einen Kern mit  $\leq 3k$  Knoten.

## 3.2 Tiefenbeschränkte Suchbäume

**Idee** Vollständige Suche über alle möglichen Lösungen, aber mit Suchbaumtiefe beschränkt im Parameter.

**Beispiel: VCP** Beobachtung: in jedem vertex cover  $S$  gilt  $\forall e = \{v_1, v_2\} : v_1 \in S \vee v_2 \in S$ . Daher verzweige von  $(G, k)$  nach  $(G - \{v_1\}, k - 1)$  und  $(G - \{v_2\}, k - 1)$ . In den Blättern  $(G_j, 1)$  ist das VCP trivial. Dieser Suchbaum hat Tiefe  $k$  und  $2^{k-1}$  Blätter. Gesamtlaufzeit:  $\mathcal{O}(2^k \cdot n)$ .

## Cluster Editing Problem CEP

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

Ausgabe: JA falls durch Löschen/Einfügen von  $\leq k$  Kanten  $G$  in eine Vereinigung (union) von disjunkten Cliques (= jeder connected component ist eine Clique) transformiert werden kann, sonst NEIN. Bekannterweise NP-schwer.

Theorem: Graph  $G$  besteht aus disjunkten Cliques

$\iff G$  enthält *keinen* Pfad aus drei Knoten<sup>10</sup>

$\iff$  es gibt *keine* paarweise verschiedenen  $u, v, w \in V : \{u, v\}, \{v, w\} \in E \wedge \{u, w\} \notin E$

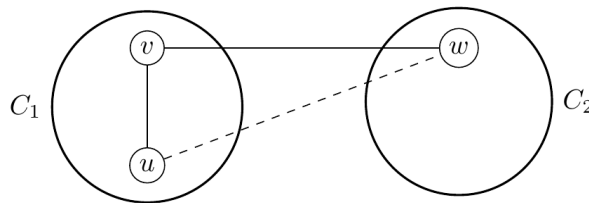


Figure 3: Cluster Editing Situation (Quelle: Vorlesung)

Algorithmus: Teste ob  $G$  bereits eine Vereinigung disjunkter Cliques ist, oder ob  $k = 0$ . Wenn nicht, finde  $u, v, w \in V$  mit obiger Bedingung. Rekursiere auf  $G_1 = (V, E - \{\{u, v\}\})$ ,  $G_2 = (V, E - \{\{v, w\}\})$ ,  $G_3 = (V, E \cup \{\{u, w\}\})$ . Laufzeit<sup>11</sup>:  $\mathcal{O}(n^3 \cdot 3^k)$ .

## 3.3 Iterative Kompression

**Idee** Gegeben ein Kompressionsalgorithmus der aus einer (bzgl. dem Schwellwert) etwas zu grossen Lösung eine kleinere, gültige Lösung in fpt-Zeit berechnet. Iteriere: Instanz vergrößern, komprimieren, usw.:

<sup>10</sup>In eine Clique bilden drei Knoten einen Kreis, keinen Pfad.

<sup>11</sup>Die Rekursionsformel  $T(k) = 3 \cdot T(k - 1)$  gibt  $\mathcal{O}(3^k)$  rekursive Aufrufe.

$$\begin{array}{l}
(I_0, k_0) \text{ mit Lösung } S_0^*, |S_0^*| \leq k_0 \\
\begin{array}{c} \text{Instanz vergrößern} \\ \longrightarrow \end{array} (I_1, k_1) \text{ mit Lösung } S_1, |S_1| > k_1 \\
\begin{array}{c} \text{komprimieren} \\ \longrightarrow \end{array} (I_1, k_1) \text{ mit Lösung } S_1^*, |S_1^*| \leq k_1 \\
\vdots
\end{array}$$

### Disjoint Vertex Cover Problem DVCP

Eingabe:  $G = (V, E)$ , vertex cover  $W$ ,  $k \in \mathbb{N}$

Ausgabe: JA falls es ein vertex cover  $S$  gibt mit  $|S| \leq k \wedge S \cap W = \emptyset$  sonst NEIN.

Lemma: Jede DVCP-Instanz  $(G, W, k)$  kann in Zeit  $\mathcal{O}(|V|^2)$  gelöst werden (warum?<sup>12</sup>).

**Beispiel: VCP** Idee: füge einen Knoten nach dem anderen hinzu. Starte mit  $S_1^* = \emptyset$ . In jeder Iteration setze  $S_i \leftarrow S_{i-1}^* \cup \{v_i\}$  und benutze  $\mathcal{A}_{compress}(G_i, S_i, k)$  um entweder  $S_i^*$  zu finden oder NEIN. Details siehe Buch, Algorithmen 6.6+6.7, Seite 151ff.

Kompression: Idee: probiere alle möglichen intersections  $X$  des gesuchten VCs mit  $S$  aus:

If  $|S| \leq k$  return  $S$ . For all  $X \subseteq S$  with  $|X| \leq k$  do: solve DVCP on  $(G - X, S - X, k - |X|)$  – if YES return  $X \cup N(S - X)$ .

Laufzeit:  $\mathcal{O}(2^k \cdot n^2)$  also gesamt:  $\mathcal{O}(2^k \cdot n^3)$ .

## 3.4 Dynamische Programmierung für den Steinerbaum

### Steinerbaum-Problem STP

Eingabe:  $G = (V, E)$  mit Kantenkosten  $c : E \mapsto \mathbb{N}$ , *Terminalen*  $S \subseteq V$ , *Nicht-Terminalen/Steiner-Knoten*  $N = V - S$

Lösungen: Teilbaum  $T$  von  $G$  (“Steinerbaum”) der alle Knoten aus  $S$  enthält (und einige aus  $N$ )

Kosten:  $\text{cost}(T, (G, c, S)) = c(T) = \sum_{e \in E(T)} c(e) = \text{Summe der Kanten} = \text{“Grösse” des Baums}$

Ziel: min

**DP-Ansatz** Annahme: alle Blätter sind Terminale (warum?).

Beobachtung: falls  $Y \subseteq N$  gegeben ist dann ist das STP einfach: berechne MST von  $S \cup Y$ .

$\implies$  k-parametrisierter Algorithmus mit  $k = |N|$ .

Uninteressant, wir schauen im Folgenden ein DP mit dem Parameter  $k = |S|$  an. Für alle  $X \subseteq S$  und alle  $v \in V - X$  berechne zwei Steinerbäume:

1. Auf Instanz  $(G, c, X)$  einen Steinerbaum mit minimalen Kosten  $g(X)$ .
2. Auf Instanz  $(G, c, X \cup \{v\})$  einen Steinerbaum mit minimalen Kosten  $g_{in}(X, v)$  so dass  $v$  ein innerer Knoten ist (d.h.  $\deg(v) \geq 2$ ).

<sup>12</sup>Es reicht zu prüfen ob  $N(W)$  ein vertex cover ist

**Lemma** Es gilt:

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

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

wobei  $p(v, w)$  die minimalen Kosten eines Pfades von  $v$  nach  $w$  in  $G$  sind. Beweis siehe Buch Kapitel 6.5, Seite 159ff.

**Dreyfus-Wagner Algorithmus** Eingabe  $(G, c, S)$ . Berechne  $p(v, w)$  für alle  $v, w \in V$ . Initialisiere  $g(\{x, y\}) := p(x, y)$  für alle  $x, y \in S$ . Für alle  $X \subseteq S$  mit  $|X| \in [2, |S| - 1]$  und alle  $v \in V - X$  berechne  $g_{in}(X, v)$  und  $g(X \cup \{v\})$ . Gebe  $g(S)$  aus.

Laufzeit:  $\mathcal{O}(n^2 \log n + n \cdot m + (k^2 +) 3^k \cdot n + 2^k \cdot n^2)$

### 3.5 Baumzerlegung

**Motivation** Viele schwere Graphprobleme sind auf Bäumen einfach. Z.B. Reduktionsregel für VCP: entferne Blätter und ihre Nachbarn, füge Nachbarn ins VC hinzu.

Ziel: “Baumähnlichkeit” von Graphen beschreiben und “Baum-Algorithmenvariante” anwenden.

**Definition (Baumzerlegung)** Eine *Baumzerlegung* (tree decomposition) von  $G$  ist ein Paar  $D = (T, B)$  wobei  $T = (V_T, E_T)$  ein Baum ist. Sei  $I$  eine Indexmenge, die die Knoten aus  $V_T$  aufzählt.<sup>13</sup> Sei  $B$  eine Label-Funktion  $B : I \mapsto 2^V$  die jedem Index  $i$  von  $V_T$  eine Knotenmenge  $X_i \subseteq V$  (einen *bag*) zuordnet, so dass:

- (i)  $\bigcup_{i \in I} X_i = V$  (alle Knoten kommen vor)
- (ii)  $\forall \{u, v\} \in E \exists i \in I : u, v \in X_i$  (alle Kanten in einem bag)
- (iii)  $\forall v \in V$  bilden die bags  $X_i$  mit  $v \in X_i$  einen Teilbaum von  $T$  (Zusammenhang, Lokalität)

Die *Weite* von  $D$  ist  $\max\{|X_i|\} - 1$ .<sup>14</sup>

Die *Baumweite*  $\text{tw}(G)$  von  $G$  ist die minimale Weite über alle möglichen Baumzerlegungen.

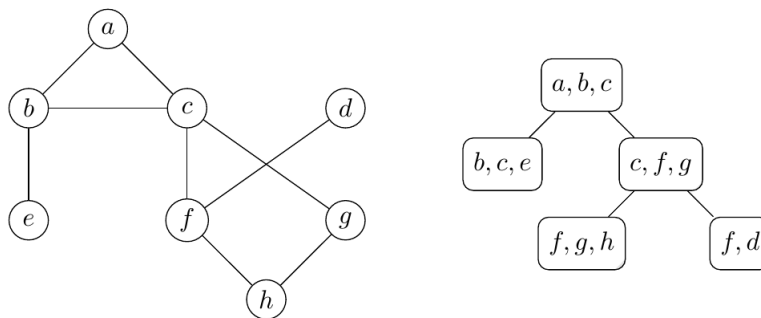


Figure 4: Graph  $G$  (links) und seine Baumzerlegung  $D$  (rechts) (Quelle: Vorlesung)

<sup>13</sup>Dies erlaubt es uns getrennt über die Knotenmenge  $V_T$  des Baums  $T$  und über die bags zu sprechen.

<sup>14</sup>Das  $-1$  ist kosmetisch damit echte Bäume eine Baumweite von 1 haben.

## Beobachtungen

- Sei  $G$  ein Baum. Dann ist  $\text{tw}(G) = 1$ .
- Sei  $G$  ein Kreis. Dann ist  $\text{tw}(G) = 2$ .
- Sei  $G$  ein Graph,  $v \in V, e \in E$ . Dann ist jede Baumzerlegung  $D$  für  $G$  auch eine Baumzerlegung für  $G - e$ . Auch kann  $D$  in eine Baumzerlegung  $D'$  für  $G - v$  transformiert werden mit gleicher oder kleinerer Weite (z.B. durch Entfernen von  $v$  aus  $D$ ).  
 $\implies$  Die Baumweite ist monoton: zusätzliche Knoten/Kanten verringern nicht die Baumweite.
- Sei  $G$  ein Graph, sei  $C \subseteq V$  eine Clique der Grösse  $|C| = k$ . Dann gilt  $\text{tw}(G) \geq k - 1$  (und es existiert ein bag, der alle Knoten der Clique enthält).  
 $\implies$  grosse Cliques  $\implies$  grosse Baumweite. ABER kleine Cliques  $\nRightarrow$  kleine Baumweite (z.B.  $l \times l$ -Gitter-Graph).  
 $\implies$  Konzept der Clique kein genügendes Modell für den Zusammenhang (und damit der Baumähnlichkeit) eines Graphen. Verallgemeinerung: statt Knoten die sich gegenseitig berühren betrachten wir Mengen von Knoten die sich gegenseitig berühren.

**Definition (bramble)** Sei  $G$  ein Graph, sei  $\mathcal{B} = \{B_1, \dots, B_k\}$  mit  $B_i \subseteq V$ . Wir sagen dass  $B_i, B_j$  sich *berühren* (*touch*), falls  $B_i \cap B_j \neq \emptyset$  oder  $\exists \{x_i, x_j\} \in E : x_i \in B_i \wedge x_j \in B_j$ .

Wenn alle  $B_i$  sich paarweise berühren, dann heisst  $\mathcal{B}$  *bramble* ("Dornen-/Brombeergestrüpp") von  $G$ .

$C \subseteq V$  heisst *Überdeckung* (*cover*) von  $\mathcal{B}$ , falls  $C \cap B_i \neq \emptyset$  für alle  $i$ .

Die *Ordnung* (*order*) von  $\mathcal{B}$  ist die Grösse einer minimalen Überdeckung von  $\mathcal{B}$ .

Die *bramble number*  $\text{bn}(G)$  von  $G$  ist die maximale Ordnung eines bramble von  $G$ .

Beispiel: In einem  $k \times k$ -Gitter-Graph sei ein "Kreuz"  $C_{i,j}$  die Vereinigung von Reihe  $i$  und Spalte  $j$ . Dann ist  $\mathcal{B} = \{C_{i,j} \mid 1 \leq i, j \leq k\}$  ein bramble von  $G$  der Ordnung  $k$  und jede Reihe und jede Spalte ist eine Überdeckung von  $\mathcal{B}$ .

## Theoreme

- Sei  $G$  ein Graph. Dann gilt  $\text{bn}(G) = \text{tw}(G) + 1$ .  
 $\longrightarrow$  Die bramble number beschreibt die Grösse des grössten bags in einer optimalen Baumzerlegung. D.h. sie bietet eine untere Schranke für die Weite einer beliebigen Baumzerlegung.
- Die Konstruktion einer Baumzerlegung und Bestimmung der Baumweite sind NP-schwer.  
 $\longrightarrow$  Problem: das Parameter eines parametrisierten Algorithmus' sollte in Polynomzeit berechenbar sein!
- Sei  $G$  ein Graph mit  $\text{tw}(G) = k$ . Dann kann man eine Baumzerlegung  $D$  der Grösse/Weite  $\leq 5k$  in Zeit  $2^{\mathcal{O}(k)} \cdot n$  berechnen.

### 3.5.1 Beispiel: VCP parametrisiert mit Baumweite

**Idee** Löse VCP in den bags und setze die Gesamtlösung aus den Teillösungen anhand der Baumstruktur bottom-up zusammen.

**Definition** Eine Baumzerlegung heisst *einfach* (*simple*), falls  $\forall i, j \in I : X_i \not\subseteq X_j$ .

**Lemma** Jede Baumzerlegung lässt sich umwandeln in eine einfache Baumzerlegung.

Beweis: falls solche  $X_i, X_j$  existieren, dann müssen sie auf einem Pfad liegen. O.B.d.A. ist  $X_i$  Nachbar von  $X_j$ . Kontrahiere beide. Die Weite bleibt erhalten. Machbar in Zeit  $\mathcal{O}(\text{tw}(G) \cdot n)$ .<sup>15</sup>

**Lemma** Sei  $G$  ein Graph. Dann existiert eine Baumzerlegung  $D = (T, B)$  von  $G$  mit (optimaler) Weite  $\text{tw}(G)$  so dass  $|V_T| \leq |V|$ .

—> Es gibt wenige bags, und diese sind klein.

Beweis: O.B.d.A. hat  $D$  Baumweite  $\text{tw}(G)$  und ist einfach. Wähle ein  $r \in T$  als Wurzel.  $\forall v \in V$  sei  $\text{high}(v)$  der höchste Knoten in  $V_T$  dessen bag  $v$  enthält.  $\text{high}(v)$  ist eindeutig (warum?). Behauptung:  $\forall X_i \exists v \in V : \text{high}(v) = X_i$ . Fall nicht, gilt entweder  $i = r$ , aber dann ist  $X_r = \emptyset$ . Oder aber  $X_i$  hat einen parent  $X_j$ , und da es kein eindeutiges  $v$  gibt für  $X_i$  muss  $X_i \subseteq X_j$  gelten. Beide Fälle sind ein Widerspruch zur Annahme dass  $D$  einfach ist. Also hat jeder bag mind. ein  $v$  an höchster Stelle, ergo muss  $|V_T| \leq |V|$  gelten.

**Notation** Wurzel  $R$ , Orientierung von  $R$  zu den Blättern. Teilbaum  $T_X$  mit Wurzel  $X$ . Induzierter Teilgraph  $G[T_X]$  von  $G$  mit all den Knoten die in einem bag aus  $T_X$  sind.

**Definition** Sei  $G$  ein Graph, sei  $X \subseteq Y \subseteq V$ , seien  $C_X, C_Y$  vertex cover für  $G[X], G[Y]$ .  $C_Y$  *erweitert* (*extends*)  $C_X$ , falls  $C_X \subseteq C_Y$  und  $C_Y \cap X = C_X$  (d.h.  $C_Y$  stimmt auf  $X$  mit  $C_X$  überein).

**Theorem** Sei  $G$  ein Graph und sei  $D$  eine einfache Baumzerlegung von  $G$  der Weite  $k$ .<sup>16</sup> Dann lässt sich ein optimaler VC für  $G$  berechnen in Zeit  $\mathcal{O}(2^k \cdot k \cdot n)$ .

Beweis: Konstruktiv. Ziel: verwalte eine Tabelle, die für jeden bag  $X$  alle optimalen VCs für  $G[T_X]$  enthält – diese lassen sich zu einem optimalen VC für  $G$  erweitern.

- 1) Initialisierung: Für alle bags  $X$  berechne alle zulässigen VCs für  $G[X]$  und definiere  $w_X : 2^X \mapsto \mathbb{N} \cup \{\infty\}$  für alle  $C \subseteq X$  wie folgt:

$$w_X(C) = \begin{cases} |C| & \text{if } C \text{ is a VC for } G[X] \\ \infty & \text{otherwise} \end{cases}$$

D.h.  $w_X$  speichert die Grösse aller VC-Kandidaten für  $G[X]$ .

- 2) Bottom-up-Durchlauf von  $T$ : Für jeden Knoten  $Y$  merge die Information  $w_X$  aus jedem seiner Kinder  $X$  in  $w_Y$  hinein. Sobald alle Kinder abgearbeitet sind enthält  $w_Y$  die Grösse aller VC-Kandidaten für  $G[T_Y]$ .

Merge-Schritt: Für alle  $C_Y \subseteq Y$ , sei  $Z = X \cap Y$  und sei  $C_Z := (C_Y \cap Z) \subseteq Z$  (also  $C_Y$  erweitert  $C_Z$ ). Dann berechne:

$$w_Y(C_Y) = w_Y(C_Y) + \min\{w_X(C_X) \mid C_X \subseteq X \text{ extends } C_Z\} - |C_Z|$$

Beispiel siehe Buch Kapitel 6.6, Seite 173.

Beobachtung: Jeder Knoten der in  $T_X$  aber nicht in  $X$  vorkommt, kommt nicht ausserhalb von  $T_X$  vor (wegen Eigenschaft 3 der Baumzerlegung). Dadurch bleibt die Tabelle klein, da wir für alle abgehandelten “unteren” Knoten den optimalen VC bereits kennen.

Laufzeit:

<sup>15</sup>Wenn die Ausgangszerlegung  $D$  wie im Theorem oben gebildet wurde, impliziert dies dass  $D$   $\mathcal{O}(n)$  viele bags hat.

<sup>16</sup> $k = \text{tw}(G)$  muss nicht gelten!

- 1) Initialisierung:  $2^{k+1}$  Teilmengen pro bag, jede in amortisiert<sup>17</sup>  $\mathcal{O}(k)$  testbar ob sie ein VC ist.  $\leq n$  bags (da einfache Zerlegung).  $\implies \mathcal{O}(2^k \cdot k \cdot n)$
- 2) Mergen von  $X$  in Vorgänger  $Y$ : Sortiere  $w_x$ -Tabelle bzgl. Ordnung auf  $Z = X \cap Y$  in  $\mathcal{O}(2^k \cdot \log 2^k)$ . Finde Minimum für alle  $C_Z$  in  $\mathcal{O}(2^k)$  (da Tabelle sortiert). Für jedes  $C_Y \subseteq Y$  finde passendes  $C_Z$  in  $w_x$ -Tabelle in  $\mathcal{O}(\log 2^k)$  dank binärer Suche.  $\leq n$  merges.  $\implies \mathcal{O}(2^k \cdot k \cdot n)$

**Korollar** Das VCP ist fpt bzgl.  $\text{tw}(G)$ .

---

<sup>17</sup>Naiv  $\mathcal{O}(k^2)$  da wir alle Kanten in  $G[X]$  auf coverage testen müssen. Besser: iteriere über Teilmengen via Gray-Code, so dass sich immer nur ein Knoten ändert, d.h. wir nur  $\leq k$  Kanten testen müssen.