

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

1	Einführung	3
1.1	Definitionen	4
2	Pseudopolynomielle Algorithmen	6
3	Parametrisierte Algorithmen	7
3.1	Kernbildung	8
3.1.1	Kronenzerlegung	9
3.2	Tiefenbeschränkte Suchbäume	10
3.3	Iterative Kompression	10

1 Einführung

Konzepte

- NP-schwer vs. NP-vollständig
- Schwellwertsprache

Polynomzeit-Reduzierbarkeit Ein Entscheidungsproblem Π_1 ist “polynomzeit-reduzierbar” auf ein anderes Entscheidungsproblem Π_2 :

$$\begin{aligned} &\iff \exists \text{ Algo } \mathcal{A} \text{ s.t. } \text{time}_{\mathcal{A}} \in \text{poly} \wedge \Pi_1(x) = \Pi_2(\mathcal{A}(x)) \\ &\iff \Pi_2 \text{ mindestens so schwer wie } \Pi_1 \\ &\iff \Pi_1 \text{ höchstens so schwer wie } \Pi_2 \\ &\iff \Pi_1 \preceq_P \Pi_2 \end{aligned}$$

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

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

Π 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 Π , 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).

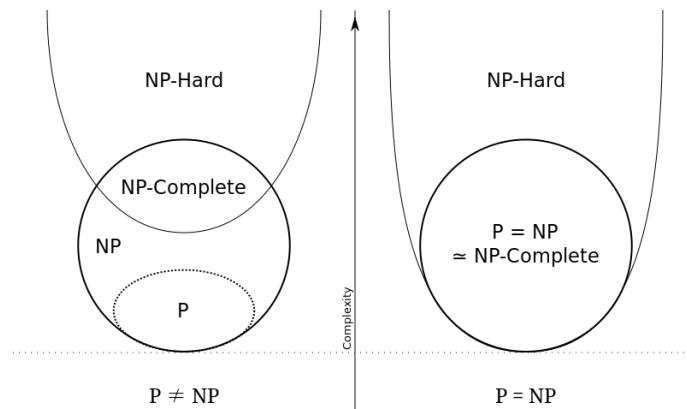


Figure 1: Mengendiagramm der Beziehungen (Quelle: Wikipedia)

“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 lösen¹
- 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) Heuristiken²

1.1 Definitionen

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

- Σ 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:

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

Vertex Cover Problem VC “Der Hefepilz der parametrisierten Algorithmer – 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 Φ erfüllt.

Bei l -SAT enthält jede Klausel maximal l Literale.

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

- L die Sprache der zulässigen Eingaben³
- $M : L \mapsto \Sigma^*$ 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 $goal = \max$.

¹D.h. die Basis der Exponentiation ist klein, z.B. 1.4^n statt 2^n .

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

³Oben 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?).⁴

⁴Recall 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 .⁵

⁵Wie 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
- Probleme: Edge Clique Cover, Cluster Editing

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) 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 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 par . \mathcal{A} heisst auch *fpt-Algorithmus* für U bezüglich 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 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, par) ein parametrisiertes Entscheidungsproblem, L die Sprache der JA-Instanzen von U . Ein *Kernbildungs-Algorithmus* für (U, 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?).⁶

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

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, par) ein parametrisiertes Entscheidungsproblem.

Ein fpt-Algorithmus für (U, par) existiert \iff Ein Kernbildungsalgorithmus für (U, par) existiert.

Edge Clique Cover Problem ECCP

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

Ausgabe: JA falls k Cliques⁷ 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) $(G, k) \xrightarrow{\text{isolierter Knoten } u} (G - \{u\}, k)$
- (ii) $(G, k) \xrightarrow{\text{isolierte Kante } e=\{u,v\}} (G - \{u, v\}, k - 1)$

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

⁷Clique: Knotenmenge so dass alle Knoten paarweise miteinander verbunden sind, d.h. einen vollständigen Teilgraphen bilden.

- (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 set⁸ 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).

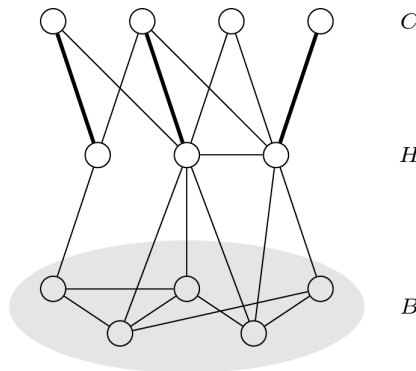


Figure 2: Kronenzerlegung: crown, head, body (Quelle: Vorlesung)

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.

⁸Eine 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 Knoten⁹

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

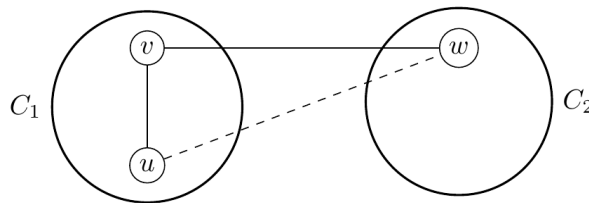


Figure 3: Cluster Editing Situation (Quelle: Vorlesung)

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 $G_1 = (V, E - \{\{u, v\}\})$, $G_2 = (V, E - \{\{v, w\}\})$, $G_3 = (V, E \cup \{\{u, w\}\})$. Laufzeit¹⁰: $\mathcal{O}(n^3 \cdot 3^k)$.

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.:

⁹In eine Clique bilden drei Knoten einen Kreis, keinen Pfad.

¹⁰Die Rekursionsformel $T(k) = 3 \cdot T(k - 1)$ gibt $\mathcal{O}(3^k)$ rekursive Aufrufe.

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

Beispiel: VCP Siehe Buch, Algorithmus 6.6, Seite 151ff.