



Technische
Universität
Braunschweig



Algorithmen und Datenstrukturen 2 – Übung #6

Wiederholung

Ramin Kosfeld & Chek-Manh Loi

16.07.2025

Inhalt

Wiederholung Stoff AuD2

Klausurinfos

Übung Reduktionen

Ausblick Branch & Bound

Zusammenfassung

Algorithmen und Datenstrukturen 2

Zusammenfassung

Algorithmen und Datenstrukturen 2



Knapsackvarianten

Problem

Typ

Komplexität

Algorithmen

Bemerkung

Laufzeit

Knapsackvarianten

Problem	0-1-KNAPSACK		MAXIMUM KNAPSACK				FRACTIONAL KNAPSACK
Typ	Entscheidungsproblem		Optimierungsproblem				Optimierungsproblem
Komplexität	NP-vollständig		← Zugehöriges Entscheidungsproblem ist NP-vollständig				P
Algorithmen	DP	B&B	Greedy ₀	Greedy _k ($k > 0$)	DP	B&B	Greedy
Bemerkung	-	-	Wert ist bel. schlecht	$1 - \frac{1}{k+1}$ -Approximation	Optimal	Optimal	Optimal
Laufzeit	$O(nZ)$	$O(n2^n)$	$O(n \log n)$	$O(n^{k+1})$	$O(nZ)$	$O(n2^n)$	$O(n \log n)$

Weitere Probleme:

- SUBSET SUM:
- PARTITION:
- MINIMUM VERTEX COVER:
- TRAVELING SALESMAN PROBLEM:

Knapsackvarianten

Problem	0-1-KNAPSACK		MAXIMUM KNAPSACK				FRACTIONAL KNAPSACK
Typ	Entscheidungsproblem		Optimierungsproblem				Optimierungsproblem
Komplexität	NP-vollständig		← Zugehöriges Entscheidungsproblem ist NP-vollständig				P
Algorithmen	DP	B&B	Greedy ₀	Greedy _k ($k > 0$)	DP	B&B	Greedy
Bemerkung	-	-	Wert ist bel. schlecht	$1 - \frac{1}{k+1}$ -Approximation	Optimal	Optimal	Optimal
Laufzeit	$O(nZ)$	$O(n2^n)$	$O(n \log n)$	$O(n^{k+1})$	$O(nZ)$	$O(n2^n)$	$O(n \log n)$

Weitere Probleme:

- SUBSET SUM: NP-vollständig, dynamisches Programm (Entscheidungsproblem)
- PARTITION: NP-vollständig, dynamisches Programm (Entscheidungsproblem)
- MINIMUM VERTEX COVER: NP-schwer (Optimierungsproblem)
- TRAVELING SALESMAN PROBLEM: NP-schwer (Optimierungsproblem)

Techniken

Nicht nur für Knapsack etc. relevant:

- Greedy-Algorithmen, Heuristiken
- Dynamic Programming
- Branch & Bound
- Approximationsalgorithmen

Hierzu sollte man nicht nur die spezifischen Algorithmen der Vorlesung gut kennen!

Auch ein gutes Verständnis der grundsätzlichen Ideen ist wichtig!

Algorithmenentwurf Aufgabe (i.d.R. zu Branch & Bound oder Dynamic Programming)!

Komplexität: Da draußen ist noch mehr

The following relations are known between PSPACE and the complexity classes **NL**, **P**, **NP**, **PH**, **EXPTIME** and **EXPSPACE** (note that $\subsetneq$, meaning strict containment, is not the same as $\not\subseteq$):

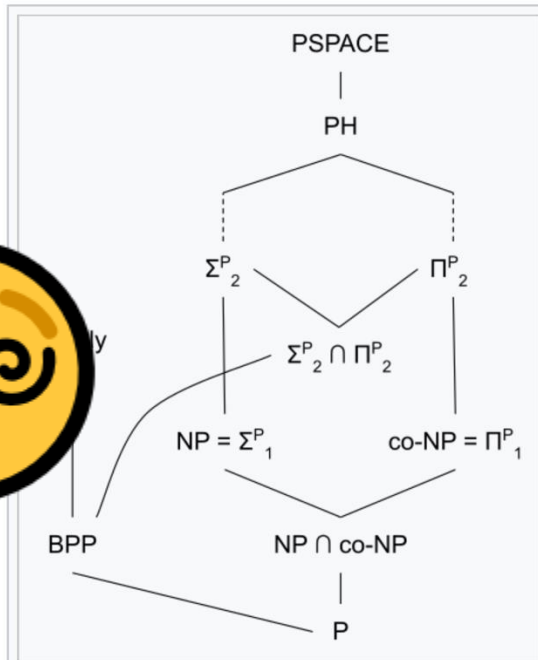
$$\text{NL} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PH} \subseteq \text{PSPACE}$$

$$\text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{EXPSPACE}$$

$$\text{NL} \subsetneq \text{PSPACE} \subsetneq \text{EXPSPACE}$$

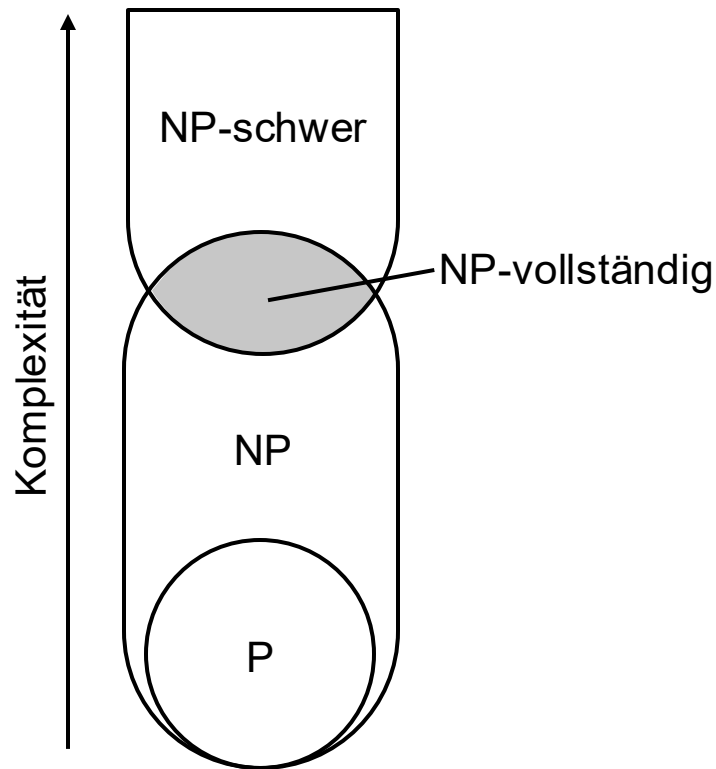
$$\text{P} \subsetneq \text{EXPTIME}$$

From the third line, it follows that both in the first and in the second line, at least one of the set containments must be strict, but it is not known which. **It is widely suspected that all are strict.**



Inclusions of complexity classes including **P**, **NP**, **co-NP**, **BPP**, **P/poly**, **PH**, and **PSPACE**

Komplexitätsklassen – für diese VL relevant



Ein Problem Π liegt in

- **P**, falls es einen Polynomialzeitalgorithmus gibt, der Π korrekt löst.
- **NP**, falls jede Lösung in polynomieller Zeit verifiziert werden kann.

Mindestens so schwer wie NP, denn man kann alle Probleme aus NP (per Reduktion) damit lösen

Ein Problem Π heißt

- **NP-schwer**, falls jedes Problem aus NP auf Π *reduziert* werden kann.
- **NP-vollständig**, falls Π NP-schwer ist und in NP liegt.

*Quasi die komplexesten Probleme in NP – denn mit ihnen kann man **auch alle anderen Probleme in NP** lösen!*

Probleme & Komplexitäten

Wichtige NP-vollständige Probleme:

- 3-SAT und ähnliche Probleme
- 0-1-Knapsack, Subset Sum
- Vertex Cover und ähnliche Probleme
- Hamiltonkreis, TSP (Entscheidungsvariante) und ähnliche Probleme

Auch hier: nicht nur Definitionen und Reduktionen kennen, auch Ideen verstanden haben.
Reduktion aus der Vorlesung kam schon in mehreren Klausuren vor (ausführen, anpassen).

Reduktionen

Gegeben:

Problem B unbekannter Komplexität

Problem A bekannter Komplexität

die wollen wir einordnen.

$\text{Problem } A \leq_p \text{ Problem } B$

„reduziere A auf B “



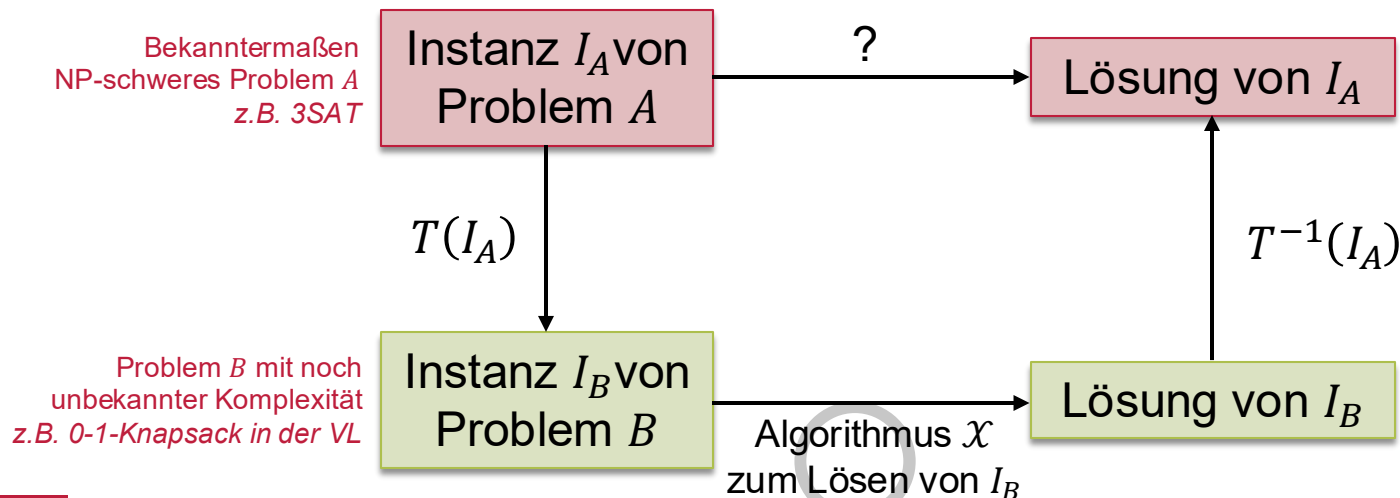
Mindset: Ich kann so bescheuerte Instanzen in Problem B bauen, dass ein Algorithmus, der smart genug ist, um die zu lösen, ganz nebenbei auch jede Instanz aus A lösen können muss.

→ Problem B ist mindestens so schwer wie Problem A

Reduktionen

$T(I_A)$ und $T^{-1}(I_A)$ besitzen polynomielle Laufzeit.
Daher: Besitzt $\mathcal{X}$ polynomielle Laufzeit, dann könnten wir die Lösung von I_A in polynomieller Zeit bestimmen.
Aber Problem A ist NP-schwer! Damit kann $\mathcal{X}$ nicht in P sein!

Wir sagen:
“Wir reduzieren (von)
3SAT auf 0-1-Knapsack”
 $3SAT \leq_p 0\text{-}1\text{-Knapsack}$



Was tun bei NP-Vollständigkeit?

Heuristik

Was tun bei NP-Vollständigkeit?

	Heuristik	Approximation	Exakt
Vorteil	Sehr schnelle und oft sehr gute Lösungen.	Gibt Garantien für den Wert der Lösung.	Geben den optimalen Wert aus.
Nachteil	Kann beliebig schlecht sein.	Oft nicht optimal, da der Worst-Case abgefangen werden muss.	Sie sind sehr langsam.
Beispiel	Greedy ₀	Greedy _k	B&B, DP

Hashing



Beliebige (Zahlen) Daten
beliebiger Länge

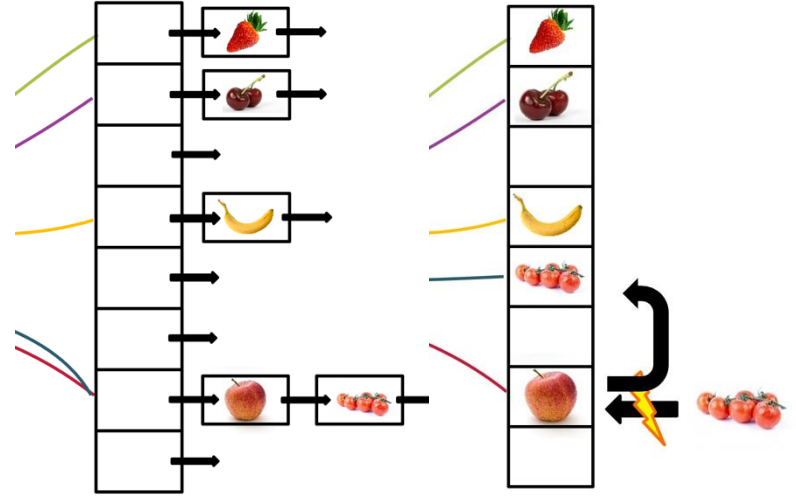
Hashfunktion

Zahl (Hash) aus endlicher
Menge von Zahlen

17

Speichere Eintrag an richtiger
Stelle in Hashtabelle

Kollisionen lösen?



...mit Listen

...mit offener
Adressierung

Das war bereits unser ganz grober
Überblick über das Semester :)

Gibt es noch Fragen?

Klausur

Klausurinfos

Steht auch auf der Webseite :)

Infos zur Klausur!

14. Juli 2025 | Allgemein

Hey zusammen!

Die Klausur für Algorithmen und Datenstrukturen 2 findet am **Freitag, den 08.08.2025**, von **15:15 bis 17:45** Uhr statt.

Die genaue Raumaufteilung wird vermutlich am Tag vor der Klausur bekannt gegeben.

Bitte beachtet die folgenden Punkte:

- Bitte findet Euch spätestens 15 Minuten vor Klausurbeginn im richtigen Prüfungsraum ein!
- Mitbringen:
 - Studierendenausweis (plus Lichtbildausweis, falls kein Foto auf dem Studiausweis ist),
 - dokumentenechter Stift (kein Bleistift, kein rot!)
 - Kein Tippex!
 - Wörterbuch, falls benötigt
- Weitere Unterlagen oder Hilfsmittel (z.B. Taschenrechner) sind *nicht* erlaubt!
- Eigenes Papier ist *nicht* erlaubt, wir stellen Papier.

Eckdaten:

- **Freitag, 08.08.2025, 15:30-17:30 Uhr**
(2h Bearbeitungszeit, spätestens 15:15 da sein!)
- Raumaufteilung wird kurz vorher bekannt gegeben (Webseite und Verteiler).
- Kurzfristige Infos werden über den Verteiler bekannt gegeben
→ *Mails regelmäßig checken!*
- Einsichtstermin wird nach der Klausur auf der Webseite und per Verteiler bekannt gegeben.

Was man wissen sollte:

- Alles aus der Vorlesung (außer explizite Exkurse, dazu höchstens grundlegende Fragestellungen)
- Übersicht bzw. einzelne größere Konzepte/Ideen aus Hausaufgaben *und* Übungen!
- Notation bei Rechenaufgaben i.d.R. wie in Hausaufgaben
- Schema wird grundsätzlich ähnlich wie alte Klausuren
- Altklausuren: Gut für die Vorbereitung!
 - Hausaufgabenzettel und GÜ auch
- Fragen: Mail an eure Tutoren :)

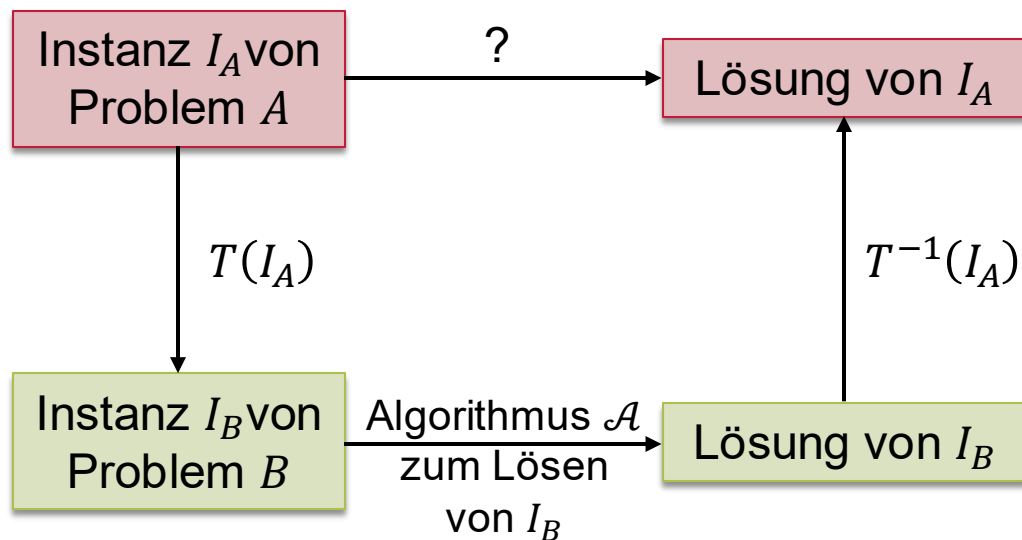
Wiederholung I

Reduktion

Reduktionen

$T(I_A)$ und $T^{-1}(I_A)$ besitzen polynomielle Laufzeit.

Daher: Besitzt $\mathcal{A}$ polynomielle Laufzeit, dann können wir die Lösung von I_A in polynomieller Zeit bestimmen.



3SAT-3

Gegeben: Formel wie bei 3SAT, aber jede Variable kommt in maximal drei Klauseln vor.

Frage: Lässt sich die Formel erfüllen?

... ist das nicht das Gleiche wie 3SAT?

→ Nein, nur *Teilmenge* aller 3SAT-Instanzen. Es könnte auch sein, dass die einfacher zu lösen sind.

... okay, *ist* das denn hier so? (Liegt 3SAT-3 in P?
Oder ist es NP-vollständig, so wie 3SAT selber? ...)

Unsere Optionen:

Polyzeit-Algorithmus finden (P) oder NP-Reduktion finden (NP-schwer).

3SAT-3

Gegeben: Formel wie bei 3SAT, aber jede Variable kommt in maximal drei Klauseln vor.

Frage: Lässt sich die Formel erfüllen?

Dieses Problem ist NP-schwer!

Wir zeigen: $3SAT \leq_p 3SAT-3$

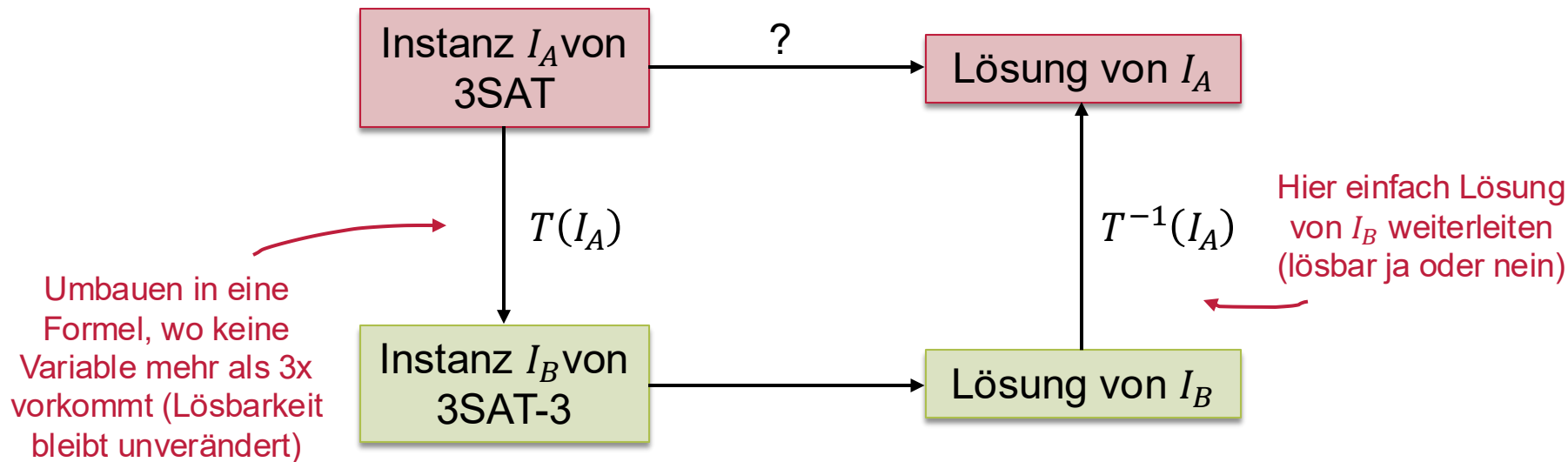
Also: Wenn wir ein Orakel (irgendeinen Solver etc.) haben, was 3SAT-3 lösen kann – dann zeige, dass wir damit auch alle Instanzen von 3SAT lösen können



Reduktionen

Die Instanzen sehen ja schon fast gleich aus.
(Manche vielleicht sogar komplett.)

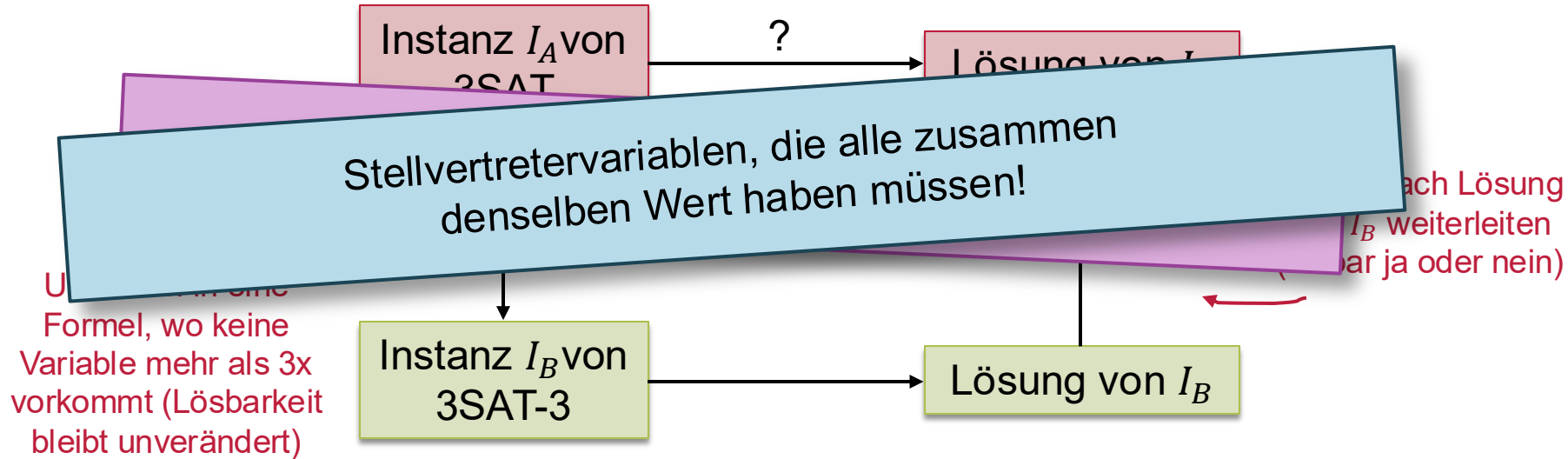
→ Problematisch nur Variablen, die öfter als drei Mal vorkommen.



Reduktionen

Die Instanzen sehen ja schon fast gleich aus.
(Manche vielleicht sogar komplett.)

→ Problematisch nur Variablen, die öfter als drei Mal vorkommen.



3SAT-3

$$(x_i \vee x_j \vee x_k)$$

$$(\bar{x}_i \vee \bar{x}_o \vee x_q)$$

$$(x_i \vee \bar{x}_p \vee x_j)$$

$$(x_i \vee x_p \vee \bar{x}_o)$$

Für jedes der n_i Literale von Variable x_i :

- Erzeuge Variablen $x_{1,i}, \dots, x_{n_i,i}$.
- Ersetze das c -te Literal von x_i mit einem Literal der Variablen $x_{c,i}$.
- Füge Klauseln der folgenden Form hinzu.

$$(x_{1,i} \vee \bar{x}_{2,i}) \wedge (x_{2,i} \vee \bar{x}_{3,i}) \wedge \dots \wedge (x_{n_i,i} \vee \bar{x}_{1,i})$$

$$\text{Äquivalent: } (x_{2,i} \rightarrow x_{1,i}) \wedge (x_{3,i} \rightarrow x_{2,i}) \wedge \dots \wedge (x_{n,i} \rightarrow x_{1,i})$$



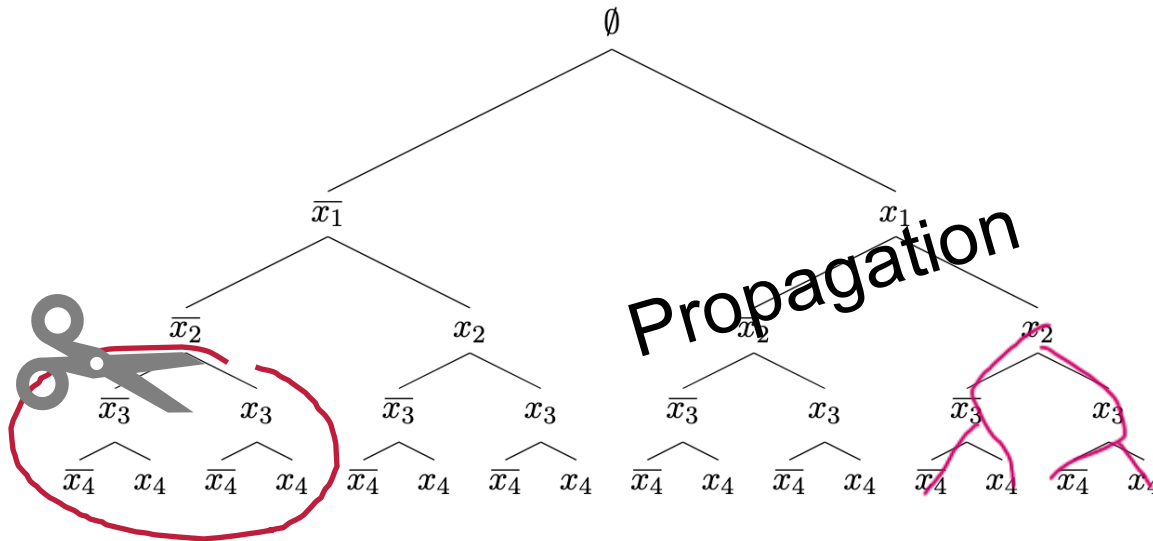
Beobachtungen:

- Die neuen Variablen tauchen genau drei Mal auf.
- Ist eine Variable auf *true* gesetzt, sind alle *true*. (Repräsentieren die gleiche Variable!)
- Die alte Variable taucht nicht mehr auf.

Ausblick: Branch & Bound



Branch & Bound



Grundidee B&B

Suche nach einer optimalen Lösung:

- Durchsuche strukturiert den gesamten Suchraum (wie vollständige Enumeration)
- D.h. wir haben Suchbaum (jeder Knoten hat einzigartige Teilbelegung)
- Wurzel: leere Teilbelegung, Blätter: vollständige Belegung
- Bei n binären Entscheidungen: 2^n Blätter im Worst Case
- Spare Arbeit:
 - Prüfe in jedem Knoten die Zulässigkeit
 - Berechne während der Suche Lösungen (z.B. an jedem Knoten Greedy₀ bei Knapsack)
 - Berechne für jeden Knoten Schranken (z.B. Upper Bound durch Fractional Greedy)
 - Die Schranken gelten für ganze Teilbäume unter ihrem Knoten
 - Falls die Schranke sagt, dass es keine bessere Lösung geben kann: Abschneiden!
- Oft kommt (in der Praxis) noch Propagation hinzu (was folgt aus der Teilbelegung?)

Vertex Cover

set of **vertices** that includes at least one

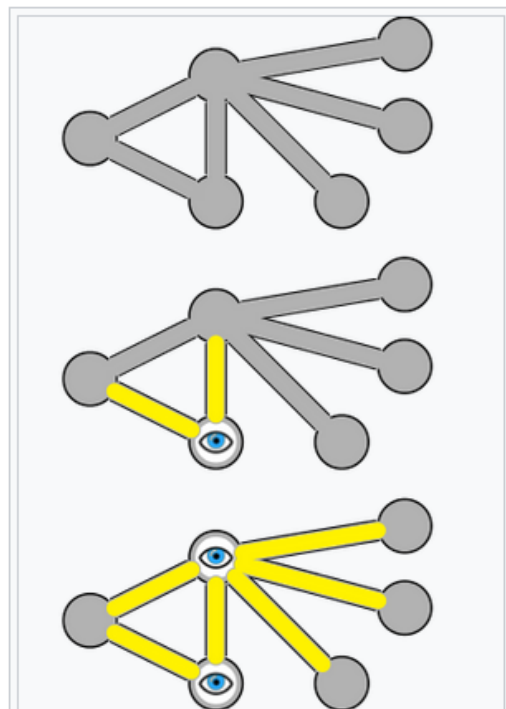
a classical **optimization problem**. It is
Moreover, it is **hard to approximate** – it
is **conjectured** is true. On the other hand, it
NP-hard optimization problem that has
em, was one of **Karp's 21 NP-complete**
onal **complexity theory**. Furthermore, the
n **parameterized complexity theory**.

linear program whose dual linear

x cover in hypergraphs.

set of V such that

every edge has at least

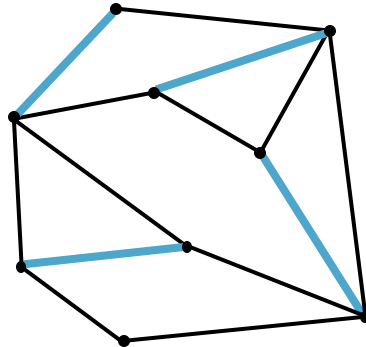


Example graph that has a vertex cover comprising 2 vertices (bottom), but none with fewer.

Minimum Vertex Cover

Suche die kleinste Menge
 $VC \subseteq V$, sodass die Knoten in
VC jede Kante abdecken!

NP-schwer!

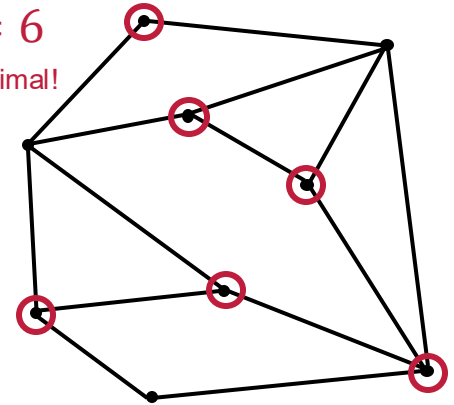


Matching:

Suche maximale Menge
von Kanten, die sich
nicht berühren.

Liegt in P.

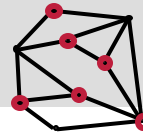
$|VC| = 6$
Hier aber *nicht* minimal!



Wähle 1 Knoten pro Kante:
Untere Schranke für Min VC

Wähle 2 Knoten pro Kante:
Obere Schranke für Min VC

Branch & Bound für Vertex Cover



Angenommen, wir wollten Vertex Cover mit Branch & Bound lösen

- Gegeben Graph $G = (V, E)$, gesucht Vertex Cover C (möglichst klein)
- Unsere Entscheidungsvariablen: $x_v \in \{0,1\}$ für jeden Knoten $v \in V$

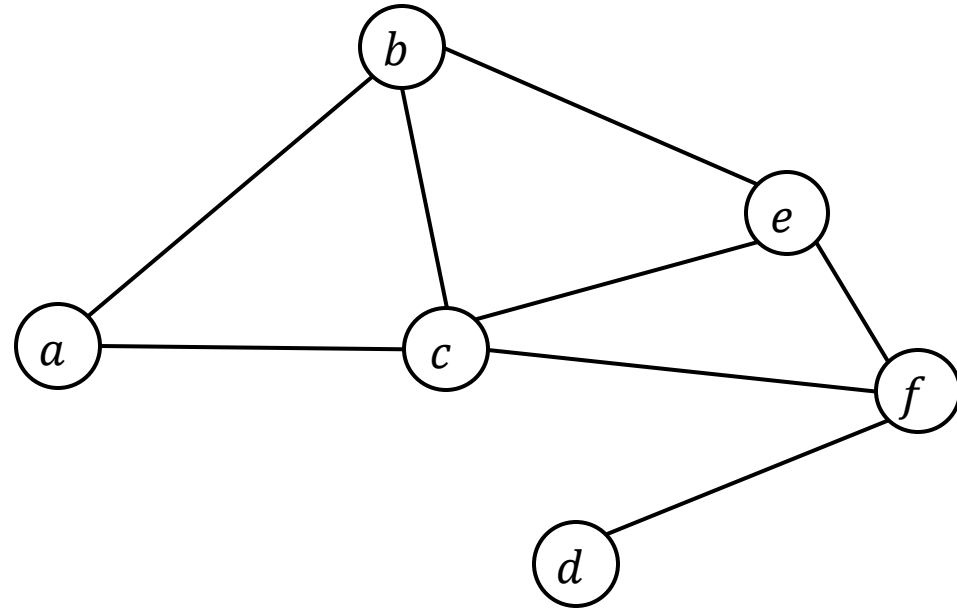
Was wollen wir haben?

- Untere Schranke
- Obere Schranke
- Propagation

Branch & Bound für Vertex Cover

Propagation:

- Aus Teilbelegungen, neue Belegungen herleiten
- Regeln:
 - Was, wenn $x_a = 0$?
 - Dann müssen $x_b = x_c = 1$ sein!
 - Was, wenn $x_b = x_c = 1$?
 - Dann können wir oBdA $x_a = 0$ setzen!
 - Was ist mit x_d ?
 - Können wir oBdA auf 0 setzen!

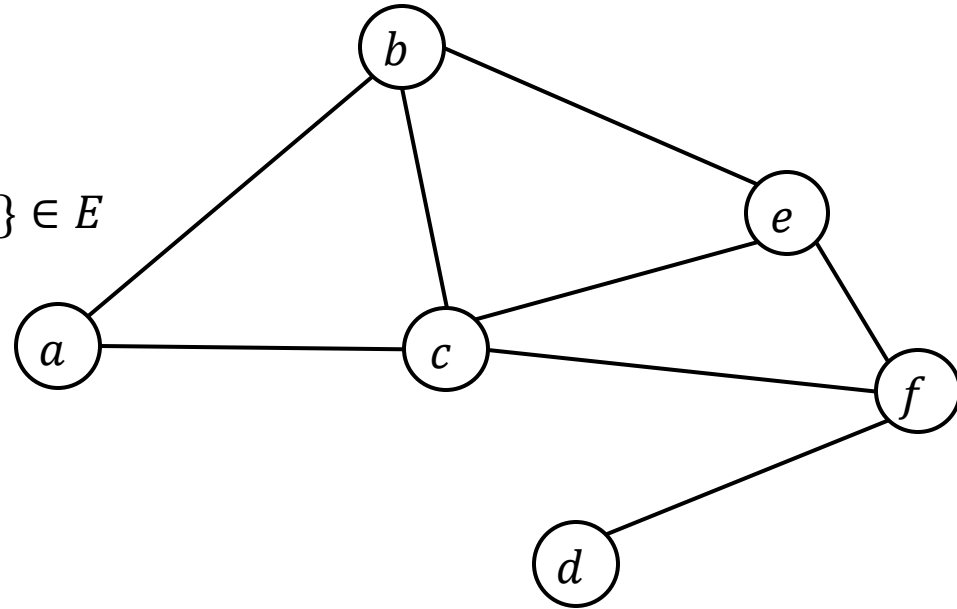


Branch & Bound für Vertex Cover

Propagation:

- Was, wenn $x_a = 0$?
- Dann müssen $x_b = x_c = 1$ sein!

Wenn $x_v = 0$, dann $x_u = 1$ für alle $\{u, v\} \in E$

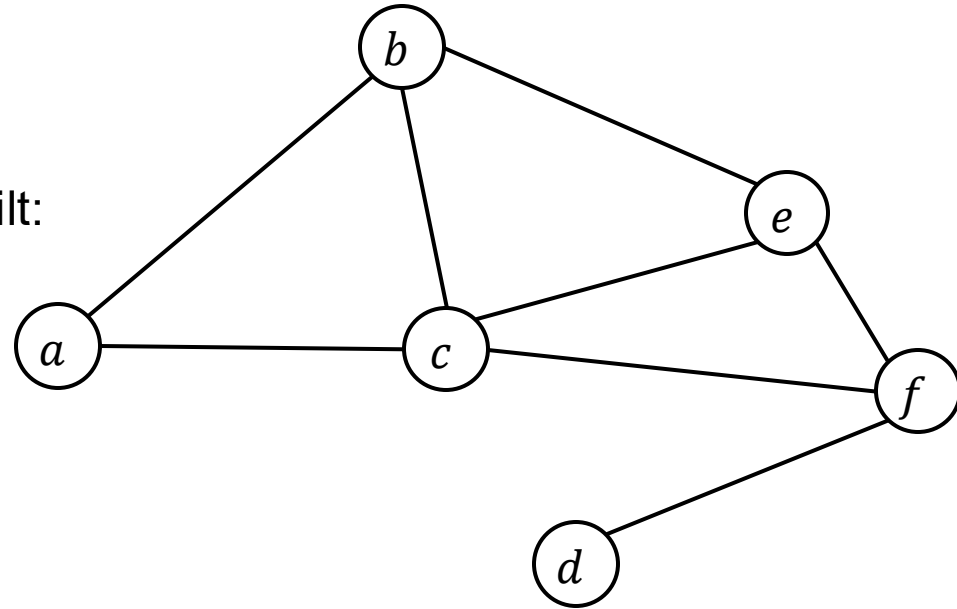


Branch & Bound für Vertex Cover

Propagation:

- Was, wenn $x_b = x_c = 1$?
- Dann können wir oBdA $x_a = 0$ setzen!

Wenn für einen unbelegten Knoten v gilt:
 $x_u = 1$ für alle $\{u, v\} \in E$, setze $x_v = 0$



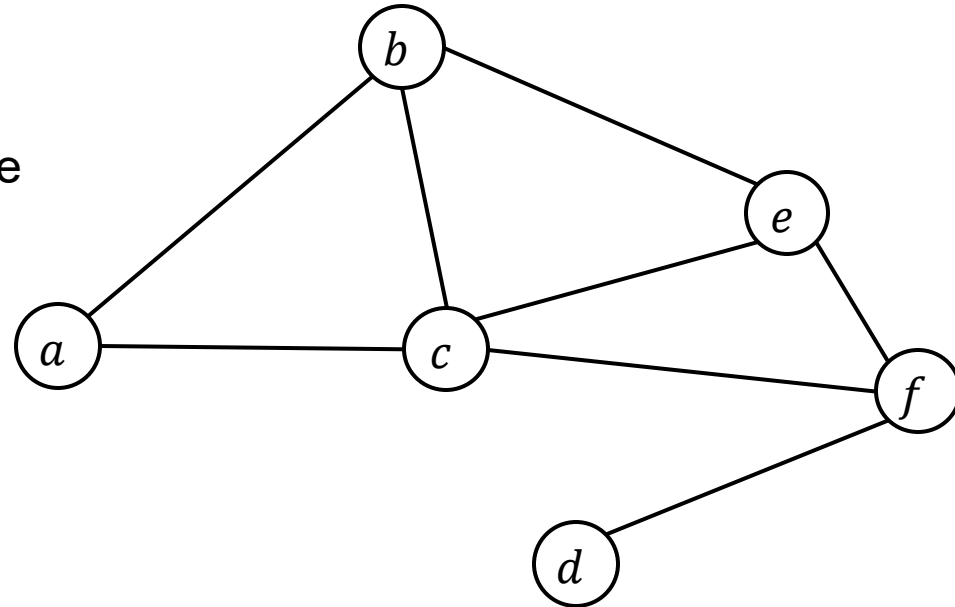
Branch & Bound für Vertex Cover

Propagation:

- Was ist mit x_d ?
- Können wir oBdA auf 0 setzen!

Wenn ein unbelegter Knoten v nur eine nicht-überdeckte Kante $\{u, v\} \in E$ hat:

- Setze $x_u = 1, x_v = 0$



Propagation für Vertex Cover: Allgemeine Regeln

Regeln: Gegeben Teilbelegung

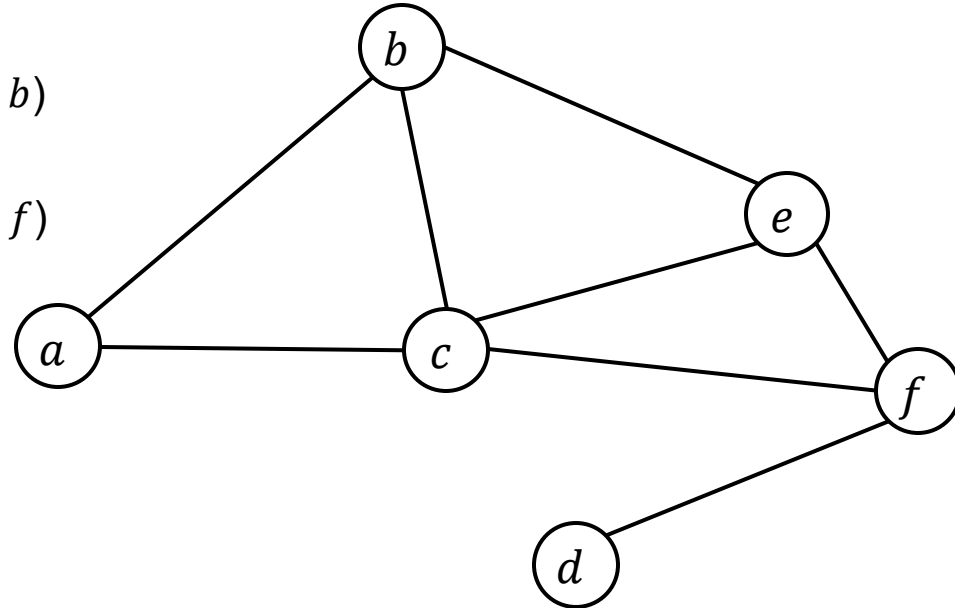
- Wenn $x_v = 0$, dann $x_u = 1$ für alle $uv \in E$
- Wenn für einen unbelegten Knoten v gilt: $x_u = 1$ für alle $uv \in E$, setze $x_v = 0$
- Wenn ein unbelegter Knoten v nur eine nicht-überdeckte Kante $uv \in E$ hat:
 - Setze $x_u = 1, x_v = 0$
- Anwendung dieser Regeln liefert erweiterte Teilbelegung
- Wir können belegte Knoten und überdeckte Kanten entfernen und erhalten einen kleineren Graphen G' , auf dem wir wieder Vertex Cover lösen können
- Die Lösung für Vertex Cover auf dem kleinen Graphen G' können wir mit der erweiterten Teilbelegung auf den gesamten Graphen erweitern

Beispiel

Angenommen, Teilbelegung $x_c = 1$

Regeln anwenden:

- a hat nur eine nicht-überdeckte Kante (zu b)
 - Setze $x_a = 0, x_b = 1$
- d hat nur eine nicht-überdeckte Kante (zu f)
 - Setze $x_d = 0, x_f = 1$
- e hat nur überdeckte Kanten
 - Setze $x_e = 0$
- Resultierender 'kleiner Graph' G' ist leer!
- Optimales VC auf leerem Graph: $C' = \emptyset$
- Ergänzung aus Teilbelegung: $C = \{b, c, f\}$



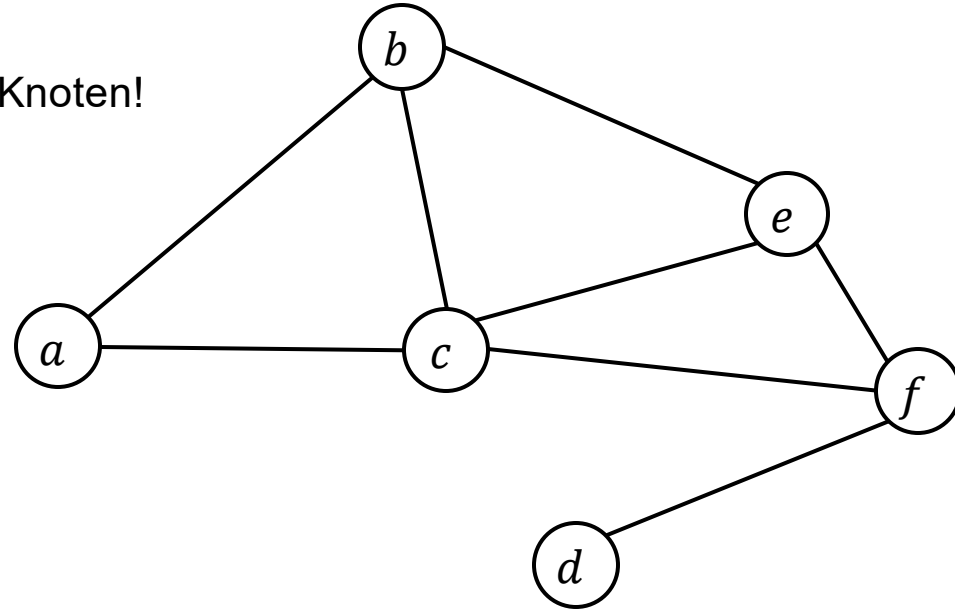
Untere Schranke

Wie viele Knoten brauchen wir mindestens?

- Maximum Matching!
- Für jedes Matching M : wir brauchen $\geq M$ Knoten!

Beispiel hier:

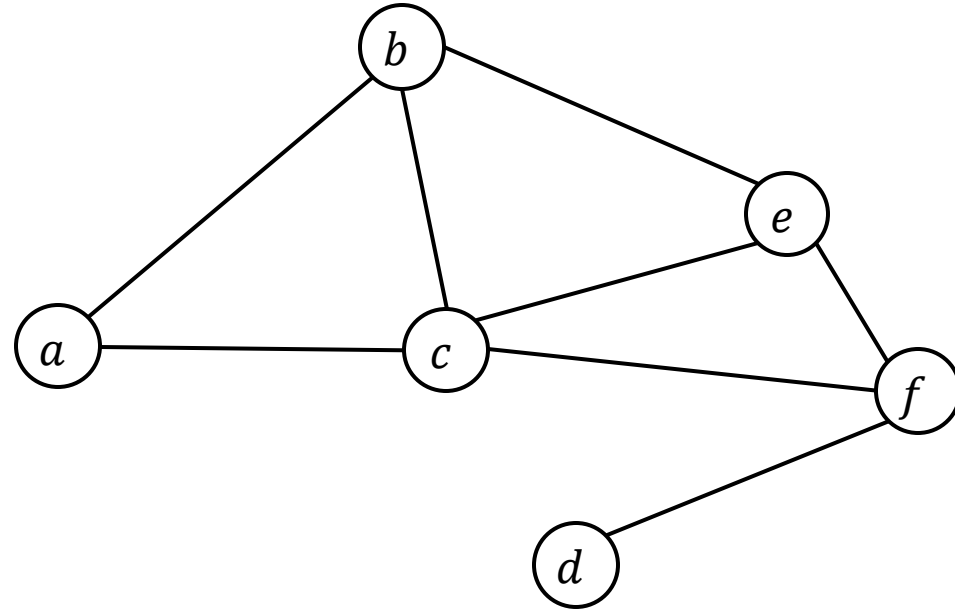
- $M = \{ab, ce, df\}$, untere Schranke 3



Obere Schranke

Wie viele Knoten brauchen wir höchstens?

- Finde heuristisch Lösungen
- Z.B. durch **maximales Matching***
 - Solange es unabgedeckte Kanten gibt:
 - Wähle beliebige unabgedeckte Kante
 - Füge beide Endpunkte ein
 - Liefert 2-Approximation



Gesamtalgorithmus

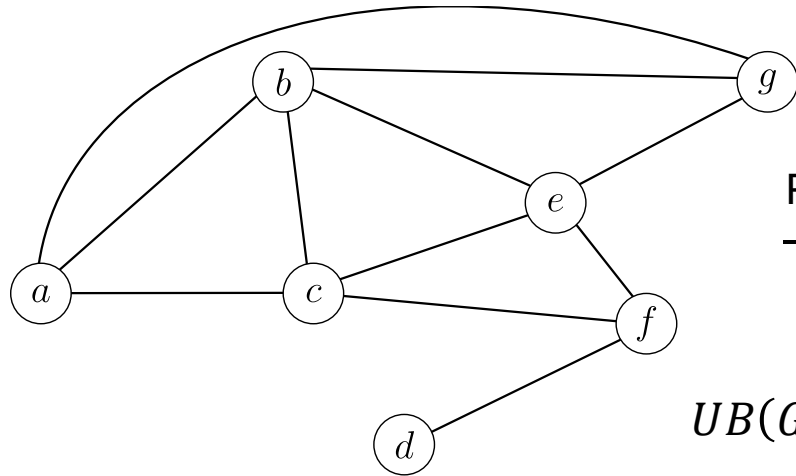
Gegeben Graph $G = (V, E)$

Globale Variable: obere Schranke $P = n$, beste Lösung $B = V$

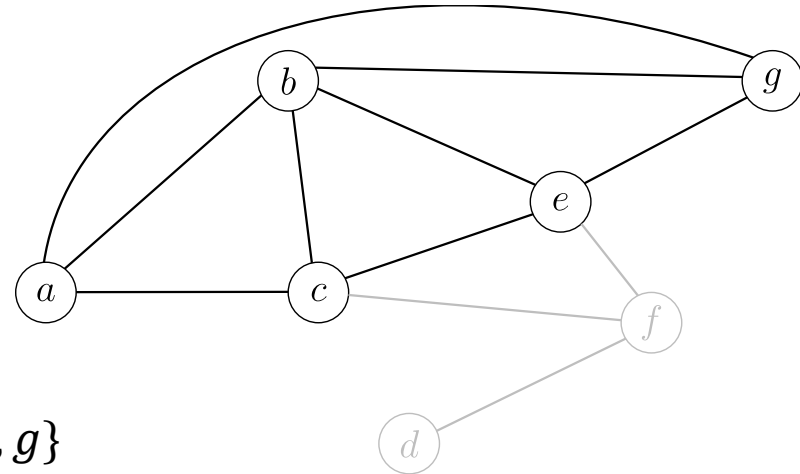
Routine VertexCoverBB(Graph G, Teilbelegung T):

- Wende Propagation an, erhalte G', T'
- Sei $C_f = \{v \in V \mid T'(x_v) = 1\}$
- $U = UB(G') \cup C_f$
- Falls $|U| < P$: $P = |U|$, $B = U$
- Falls G' leer: **return**
- $L = LB(G') + |C_f|$
- Falls $L \geq P$: **return**
- Wähle Knoten $v \in G'$
- VertexCoverBB($G', T' \cup \{x_v = 0\}$)
- VertexCoverBB($G', T' \cup \{x_v = 1\}$)

Beispiel: Wurzel des Suchbaums



Propagation



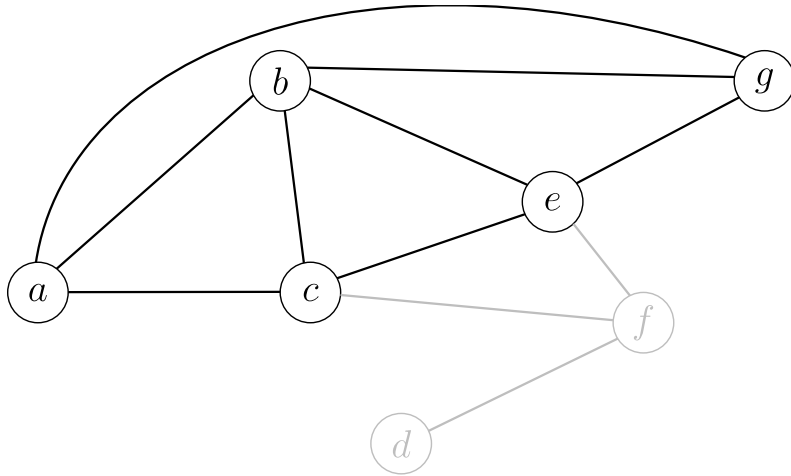
$$UB(G') = \{b, c, a, g\}$$

$$B = \{a, b, c, f, g\}, P = 5$$

$$LB(G') = |\{ab, ce\}|, L = 3$$

$$x_f = 1, x_d = 0$$

Beispiel: Wurzel des Suchbaums - Entscheidung

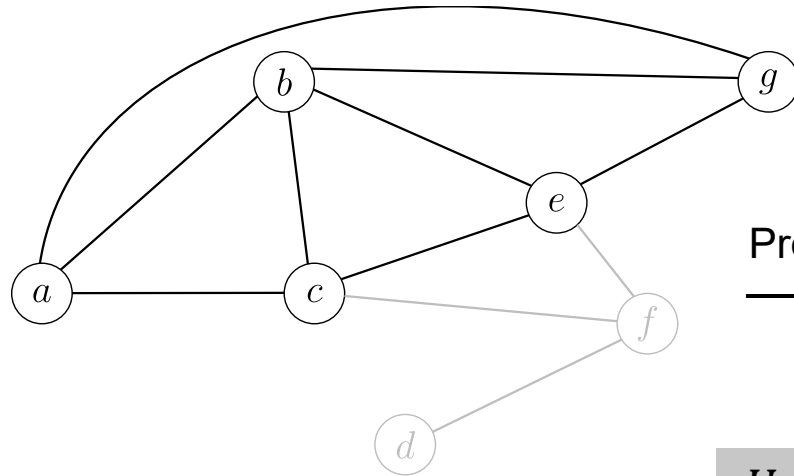


$$x_f = 1, x_d = 0$$

Fall 1: Setze $x_c = 0$

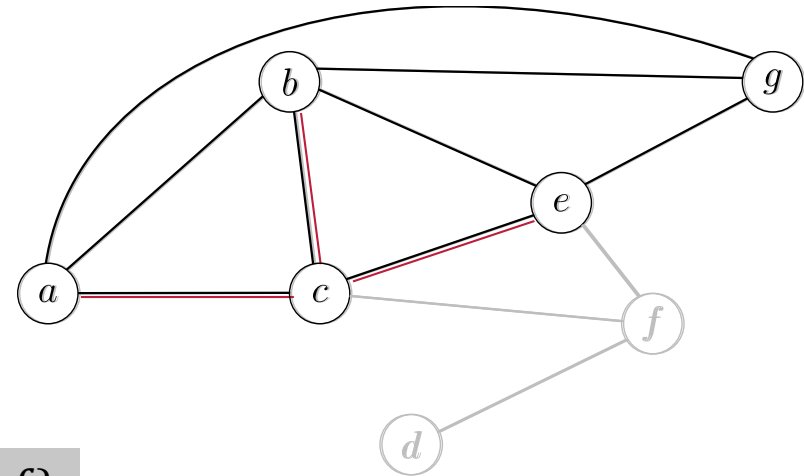
Fall 2: Setze $x_c = 1$

Beispiel: Fall $x_c = 0$



$$x_f = 1, x_d = 0, x_c = 0$$

Propagation



$$x_f = 1, x_d = 0, x_c = 0$$

$$x_a = 1, x_b = 1, x_e = 1$$

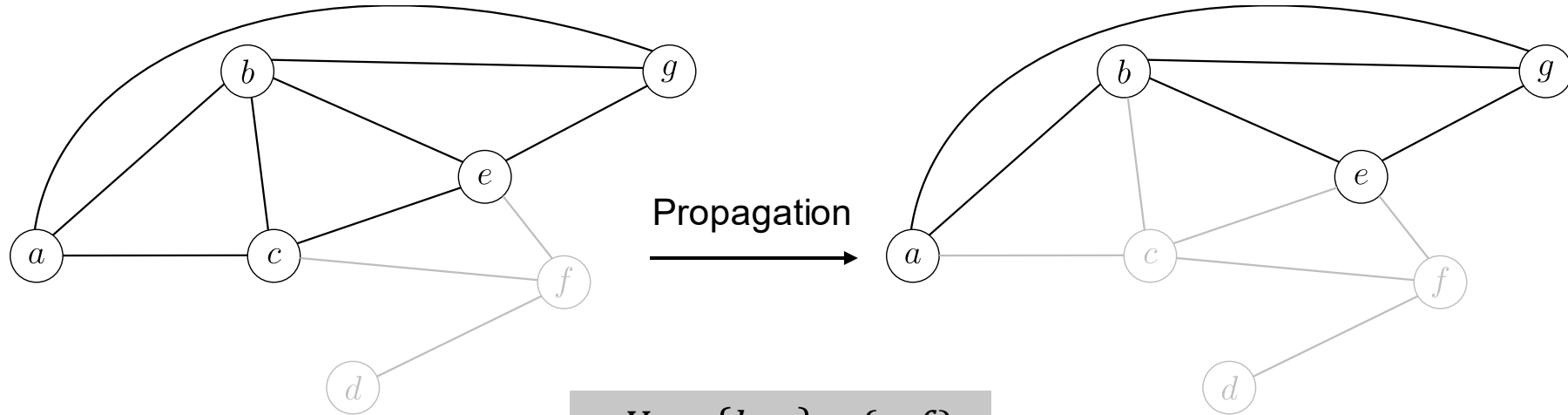
$$x_g = 0$$

$$U = \{a, b, e, f\}$$

$$B = U, P = 4$$

return (da leer)

Beispiel: Fall $x_c = 1$



$$x_f = 1, x_d = 0, x_c = 1$$

$U = \{b, g\} \cup \{c, f\}$
 $L = |\{ab, eg\}| + 2 = 4$
return (da $L \geq P$)

$$x_f = 1, x_d = 0, x_c = 1$$

B&B für Vertex Cover

Takeaway hier:

- B&B kann sehr gut zur Lösung von verschiedensten Arten von NP-schweren Problemen genutzt werden.
- Der Algorithmus muss stets *an das individuelle Problem angepasst werden*.
- Worüber wird gebranched? Hier z.B. Knoten (oder Kanten)
- Was sind geeignete obere und untere Schranken, die zum Problem passen?
- ... weitere Optimierungen und Ideen, wie Propagation, geschicktes Wählen und Treffen der nächsten Entscheidung, Bewegung durch den Suchbaum, ...



... das wars!

Schöne vorlesungsfreie Zeit
und viel Erfolg bei der Klausur :)

kosfeld@ibr.cs.tu-bs.de

loi@ibr.cs.tu-bs.de

