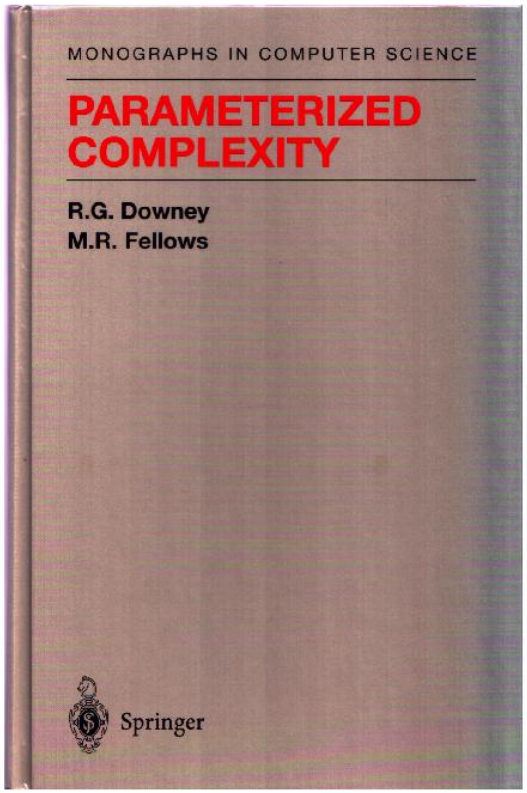
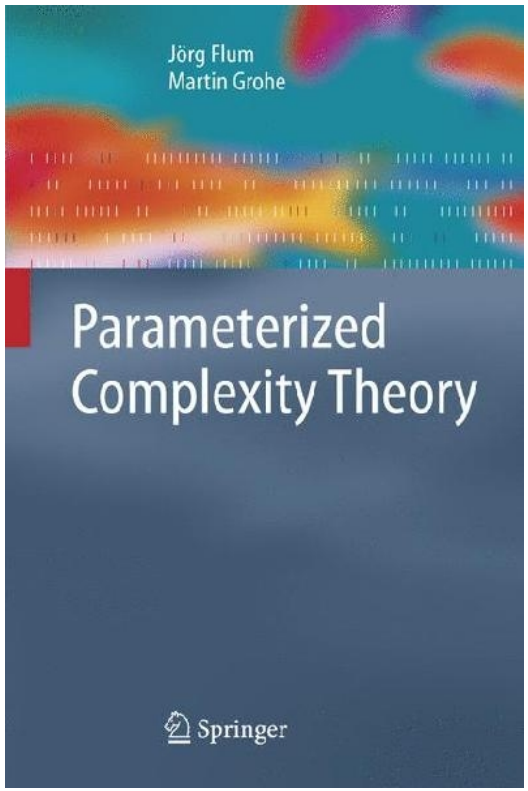
Parametrisierte Algorithmen

Montags 8:30–10:00 Uhr Raum AH II (Vorlesung)
Freitags 12:00–13:30 Uhr Raum 5052 (Vorlesung)
Montags 14:00–15:30 Uhr Seminarraum I1 (Übung)

Peter Rossmanith
Telefon 21131, Raum 6113, Erweiterungsbau E2
E-Mail: rossmani@informatik.rwth-aachen.de
Sprechstunde: Mittwochs 11–12 Uhr

Literatur





Im Handapparat vorhanden.

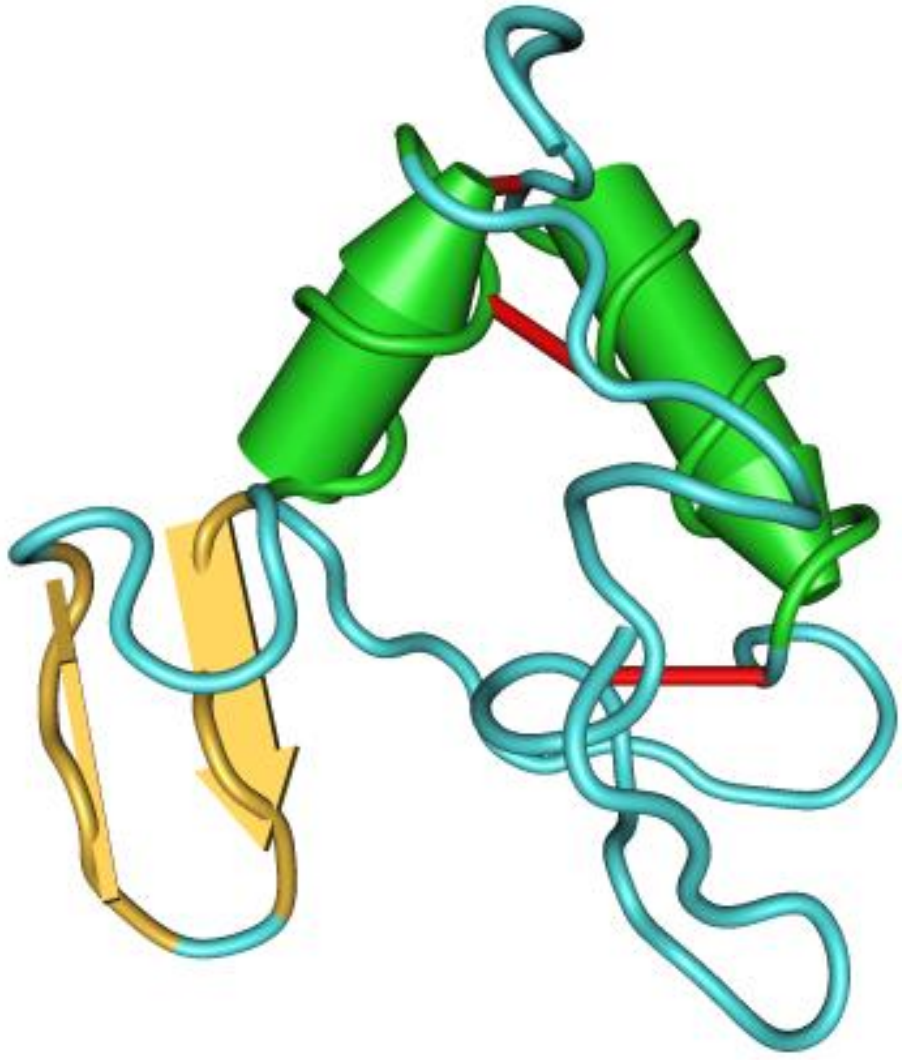
Übersicht

Einführung

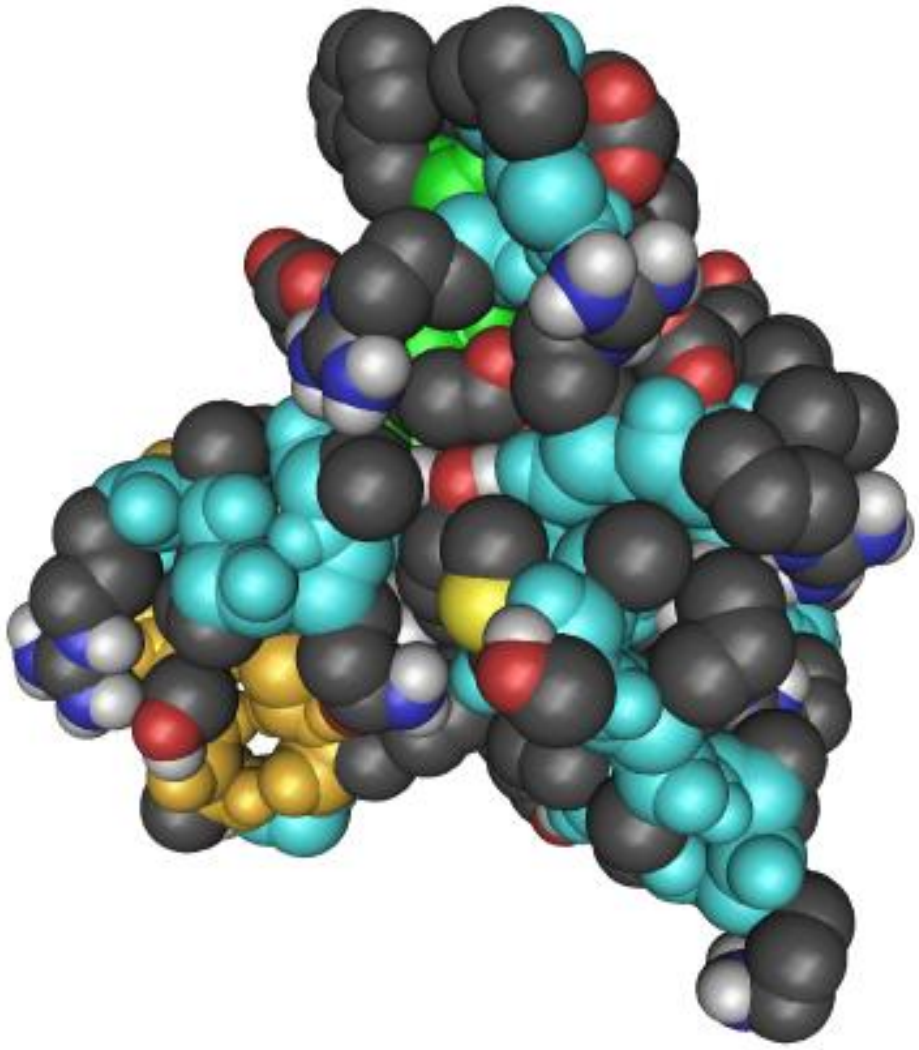
Parametrisierte Algorithmen dienen zur exakten Lösung schwerer Probleme. Andere Methoden:

- Heuristiken
- Simulated Annealing
- Approximationsalgorithmen
- Genetische Algorithmen
- Branch- and Bound
- Backtracking
- Testen aller Möglichkeiten

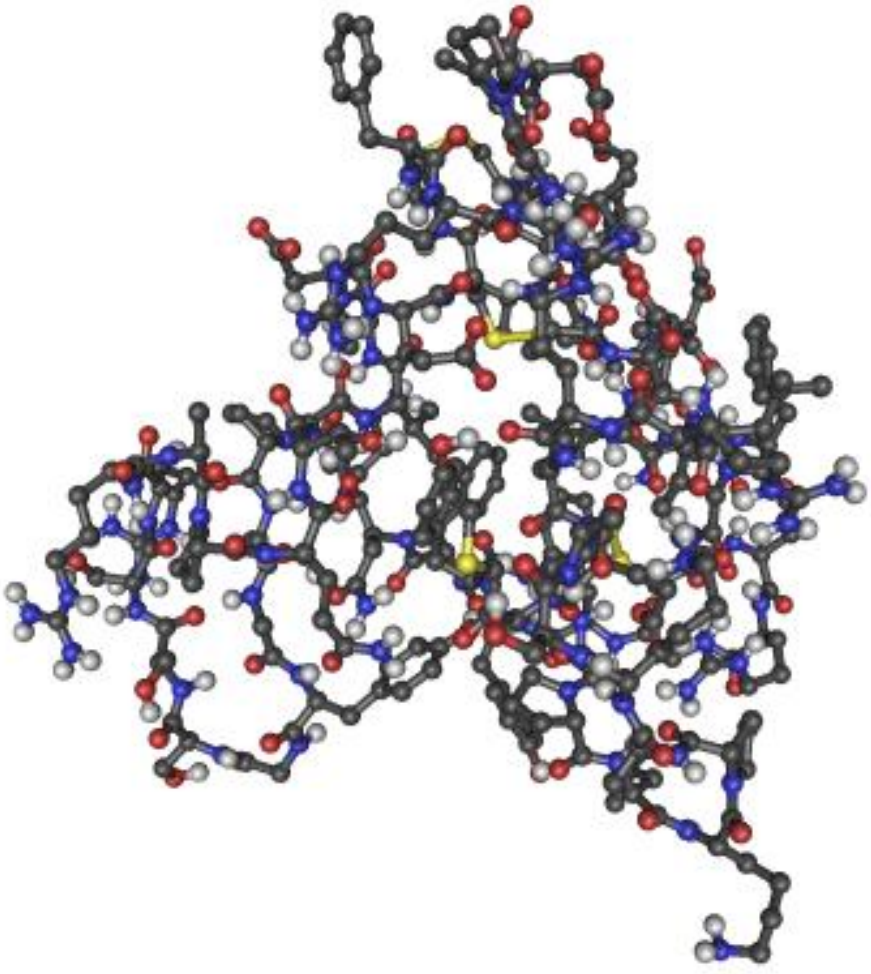
Proteinfaltung



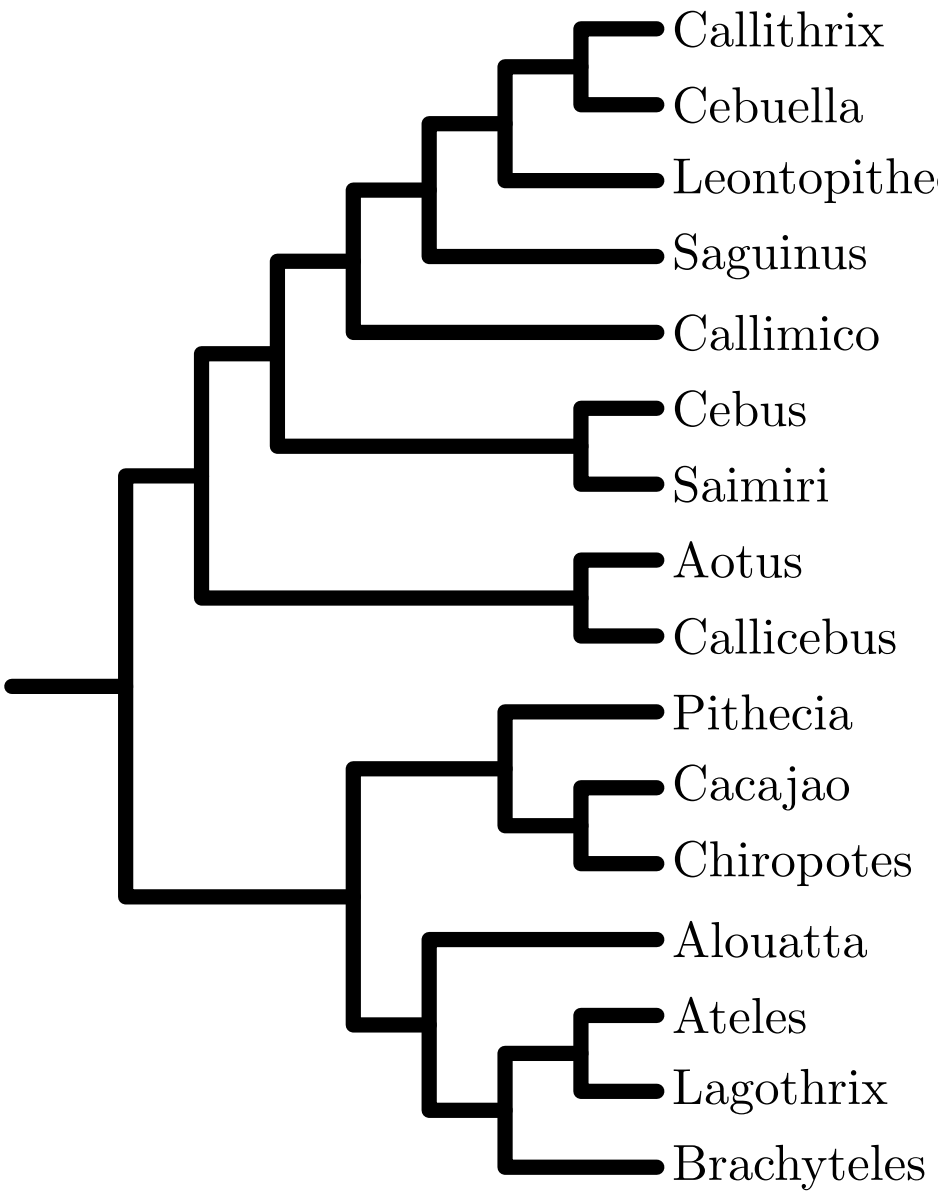
Proteinfaltung



Proteinfaltung



Taxonomie

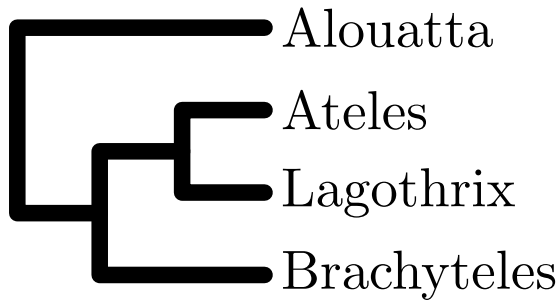

Callithrix
Cebuella
Leontopithecus
Saguinus
Callimico
Cebus
Saimiri
Aotus
Callicebus
Pithecia
Cacajao
Chiropotes
Alouatta
Ateles
Lagothrix
Brachyteles

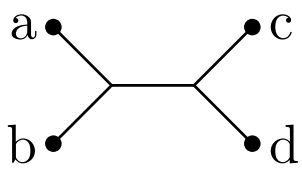
Taxonomie

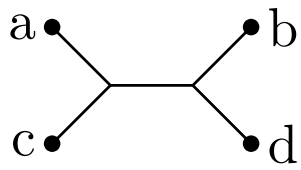
Quartettopologie

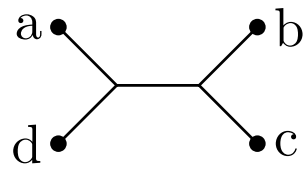
Eine Topologie, welche die Abstammung von vier Arten beschreibt.

- eindeutig
- leicht zu berechnen









Minimum Quartet Inconsistency

Minimum Quartet Inconsistency (MQI)

Eingabe: Eine Menge S von n Arten, eine Menge Q_S von Quartettopologien, sodaß zu jeder Menge $s \subset S$ mit $|s| = 4$ genau eine Quartettopologie in Q_S enthalten ist.

Frage: Existiert ein aus Q_S induzierter Evolutionsbaum, der höchstens k Topologien aus Q_S verletzt.

Laufzeit

Das MQI-Problem kann in $O(4^k \cdot n + n^4)$ Schritten gelöst werden.

Minimum Quartet Inconsistency

Minimum Quartet Inconsistency (MQI)

Eingabe: Eine Menge S von n Arten, eine Menge Q_S von Quartettopologien, sodaß zu jeder Menge $s \subset S$ mit $|s| = 4$ genau eine Quartettopologie in Q_S enthalten ist.

Frage: Existiert ein aus Q_S induzierter Evolutionsbaum, der höchstens k Topologien aus Q_S verletzt.

Laufzeit

Das MQI-Problem kann in $O(4^k \cdot n + n^4)$ Schritten gelöst werden.

NP-vollständige Probleme

Viele in der Praxis auftauchenden Probleme sind NP-vollständig.
Aus der Komplexitätstheorie ist bekannt:

Definition

Eine Sprache L ist NP-vollständig, wenn

- $L \in NP$
- Jedes Problem in NP läßt sich in polynomieller Zeit auf L reduzieren.

Theorem

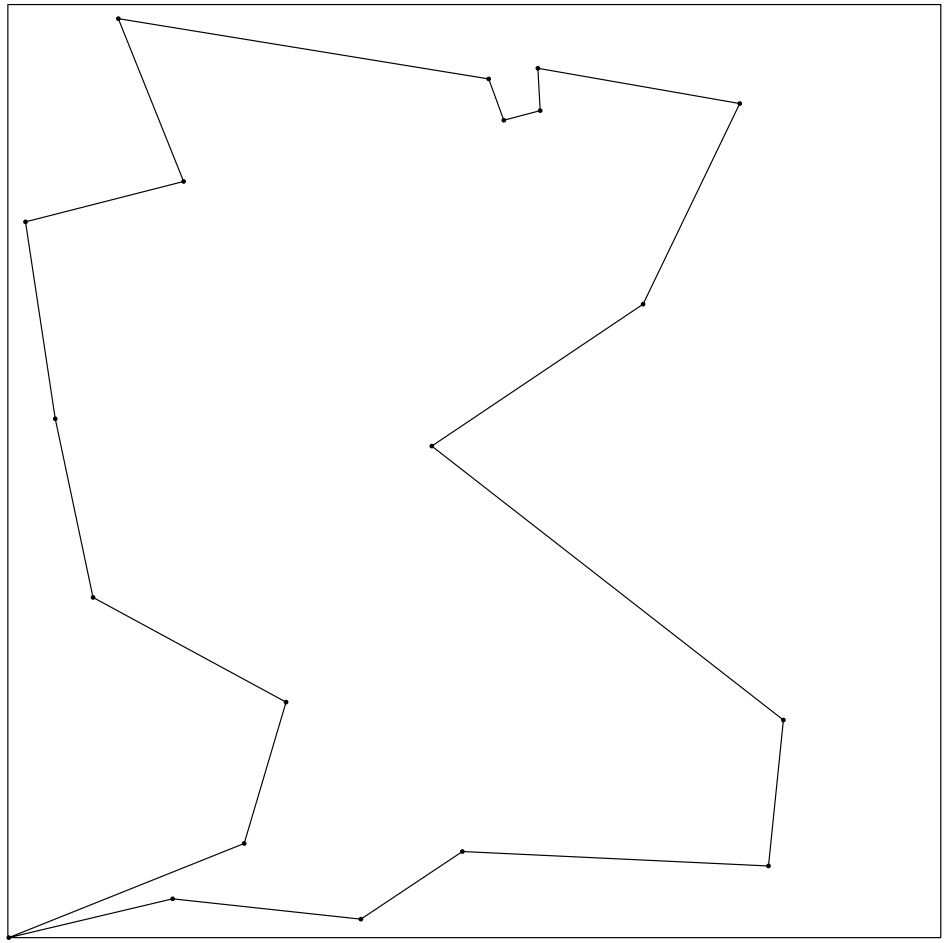
Gibt es einen polynomiellen Algorithmus für ein NP-vollständiges Problem, dann ist $P = NP$.

Frage: Folgt daraus, daß NP-vollständige Probleme praktisch schwer zu lösen sind?

NP-vollständige Probleme

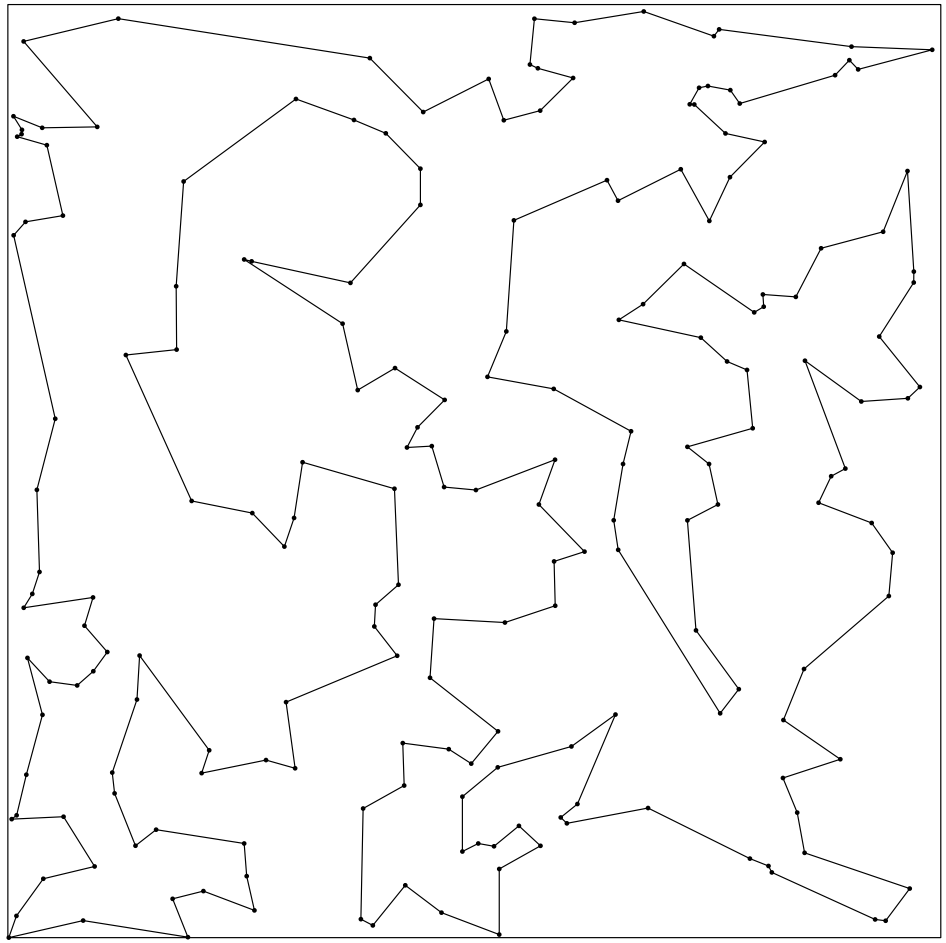
Warum ist SAT (satisfiability) NP-vollständig?
Weil sich eine Berechnung einer nichtdeterministischen Turingmaschine durch ein Schaltnetz simulieren läßt.
Daher kann die erfolgreiche Berechnung einer Turingmaschine auf die Existenz einer erfüllenden Belegung zurückgeführt werden.
Daher gibt es Formeln, deren Erfüllbarkeit so schwer zu testen sind, wie jedes beliebige Problem in NP zu lösen.
Betrachten wir die Menge aller Formeln, dann gibt es unter ihnen also sehr schwierige.
Die meisten Formeln sehen aber nicht so aus!

Beispiel: TSP



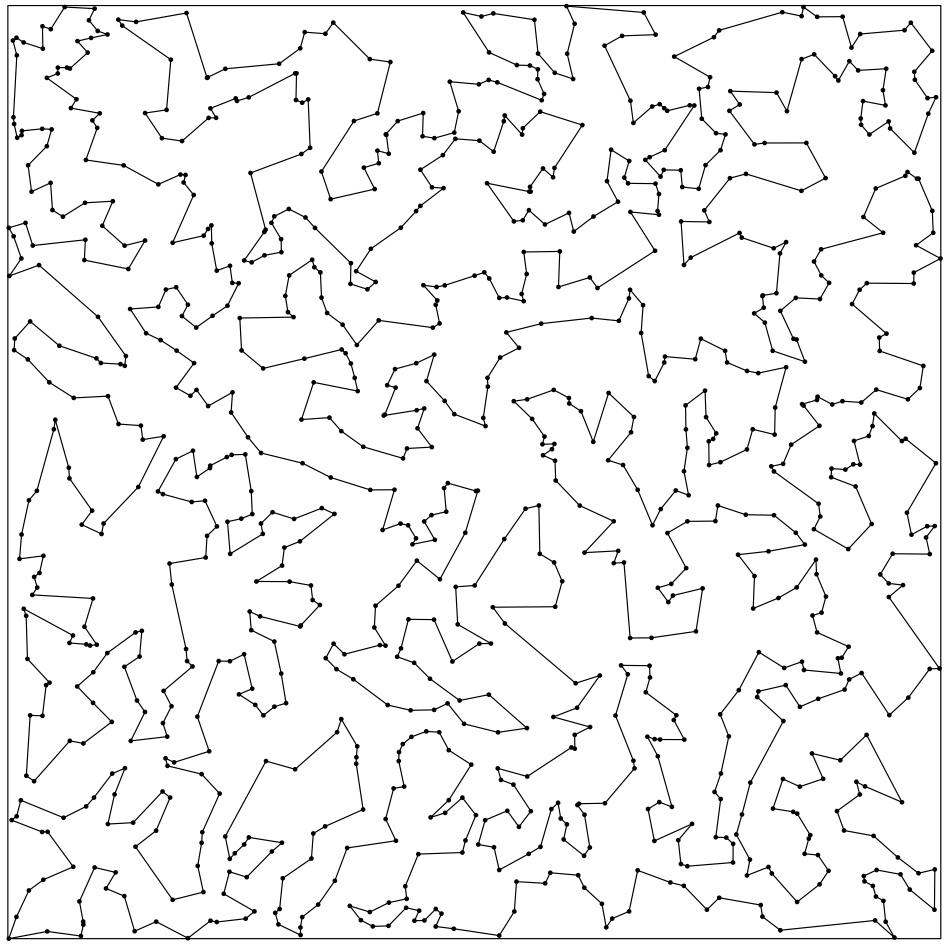
51.050611

Beispiel: TSP



435.489475

Beispiel: TSP



2107.739054

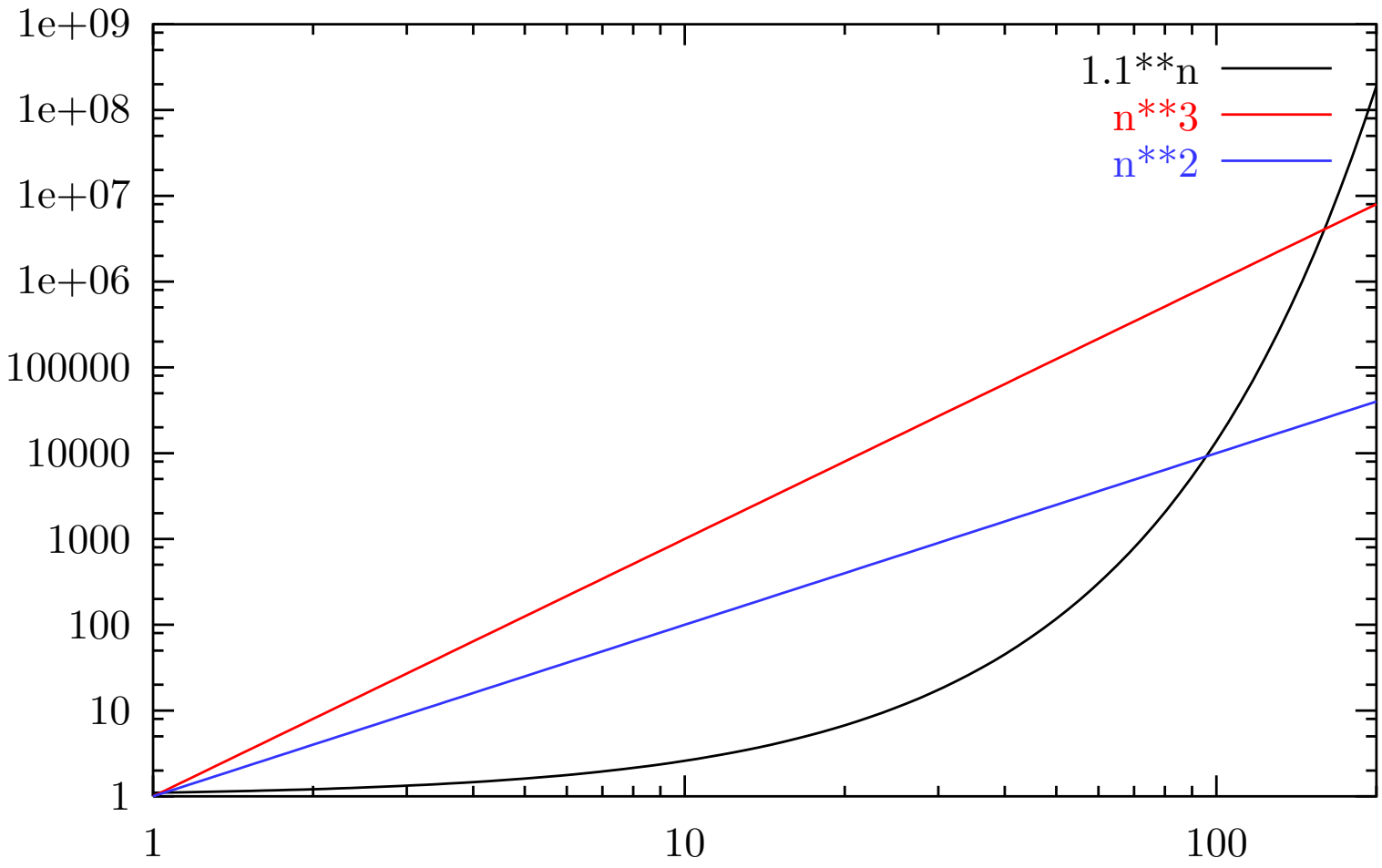
Laufzeiten

NP-vollständige Probleme sind praktisch schwer, weil es keine zielgerichteten Lösungsverfahren gibt.

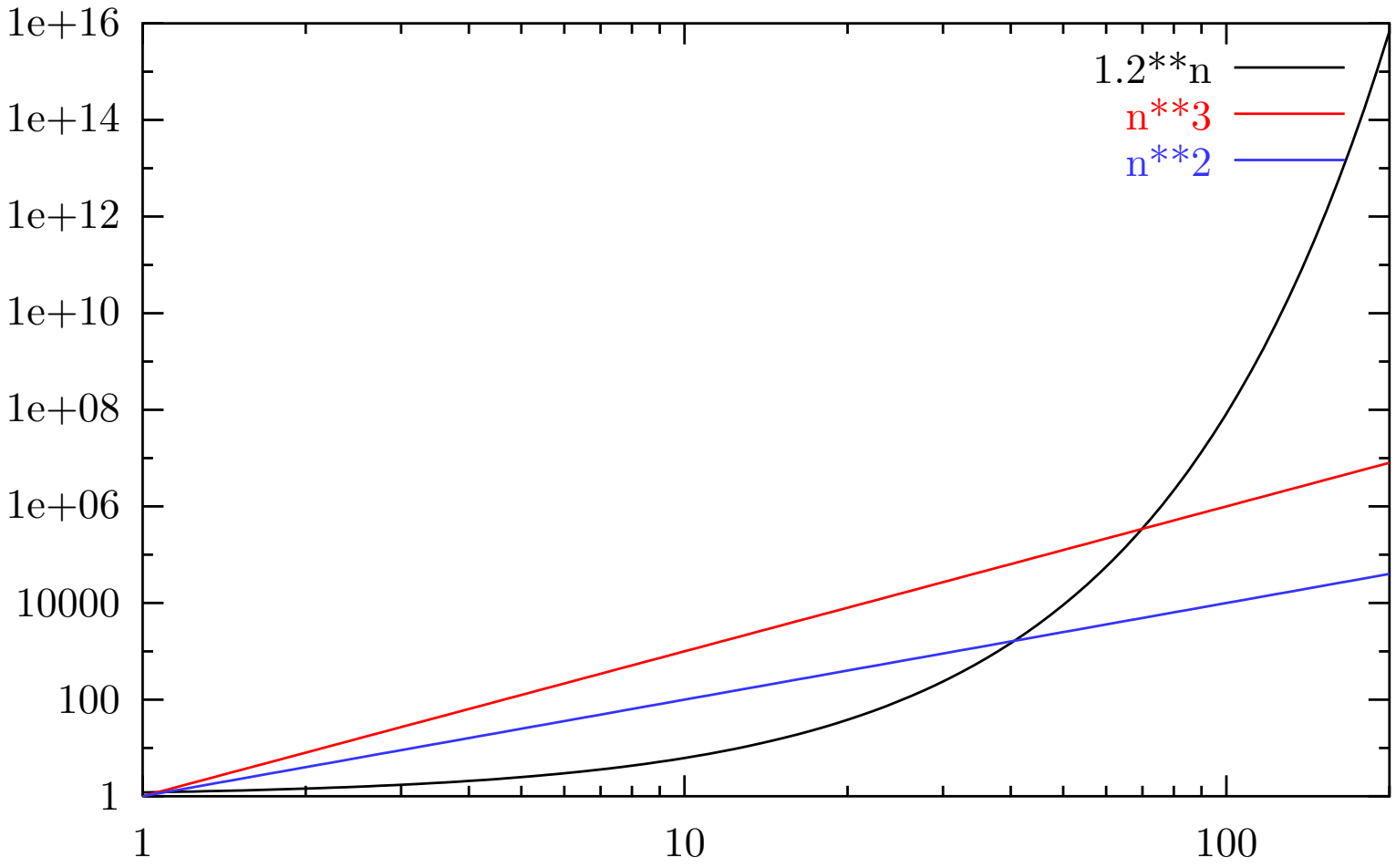
- Greedy-Algorithmen
- Divide-and-Conquer
- Dynamisches Programmieren

Praktisch müssen auch viele falsche Teillösungen probiert werden, was zu exponentiellen Laufzeiten führt.

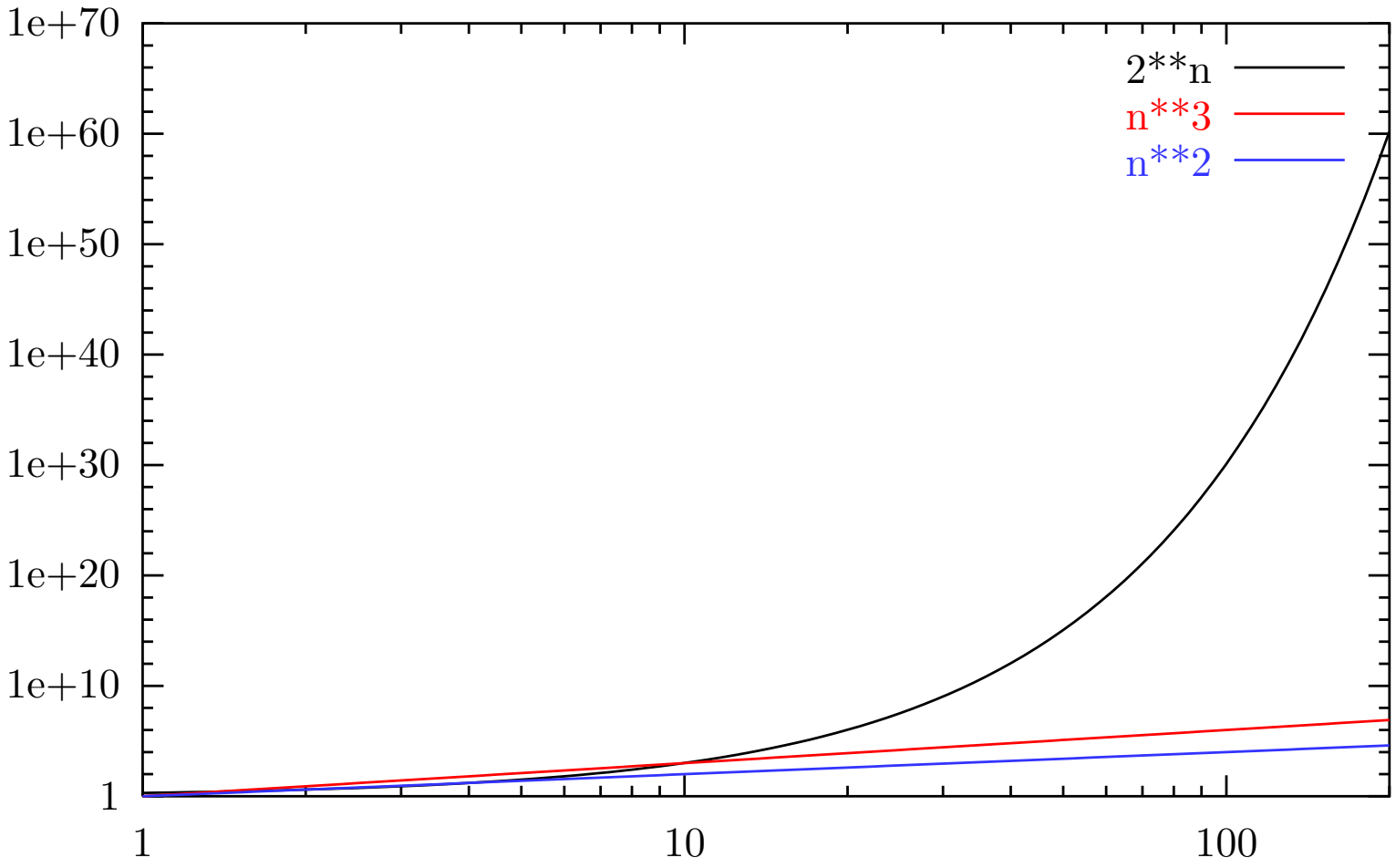
Laufzeitvergleich



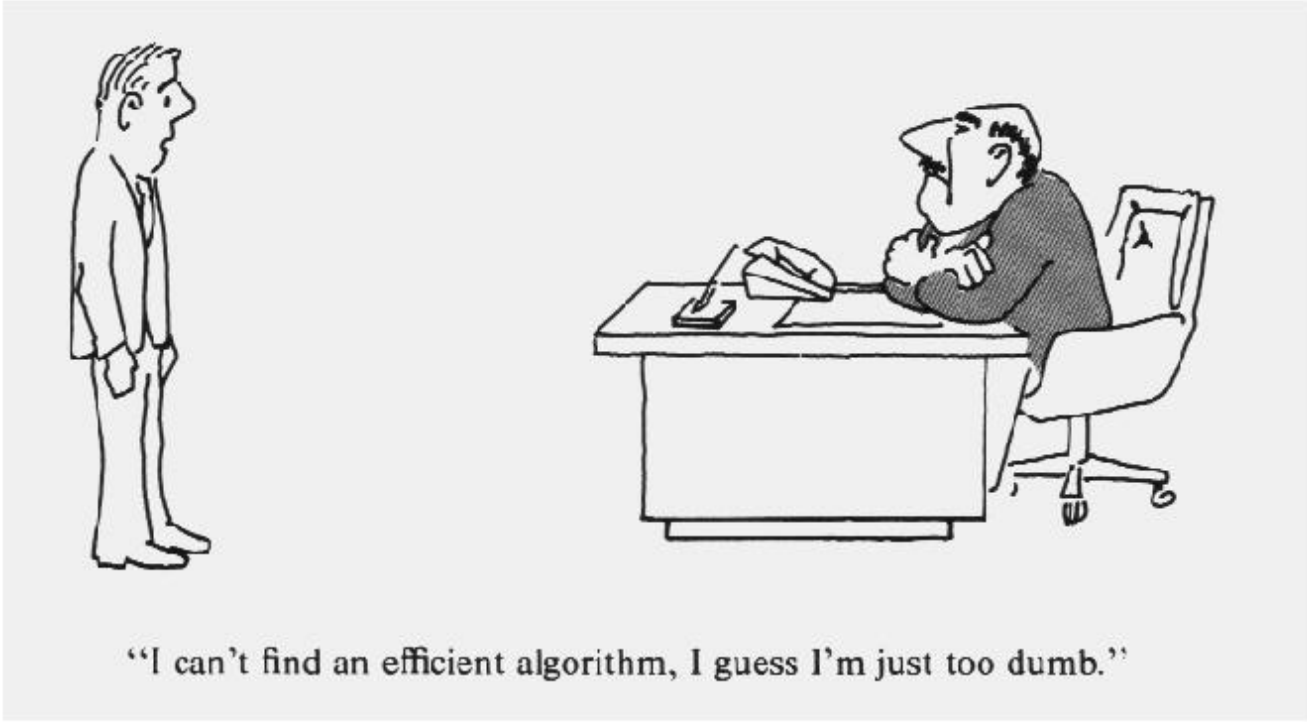
Laufzeitvergleich



Laufzeitvergleich



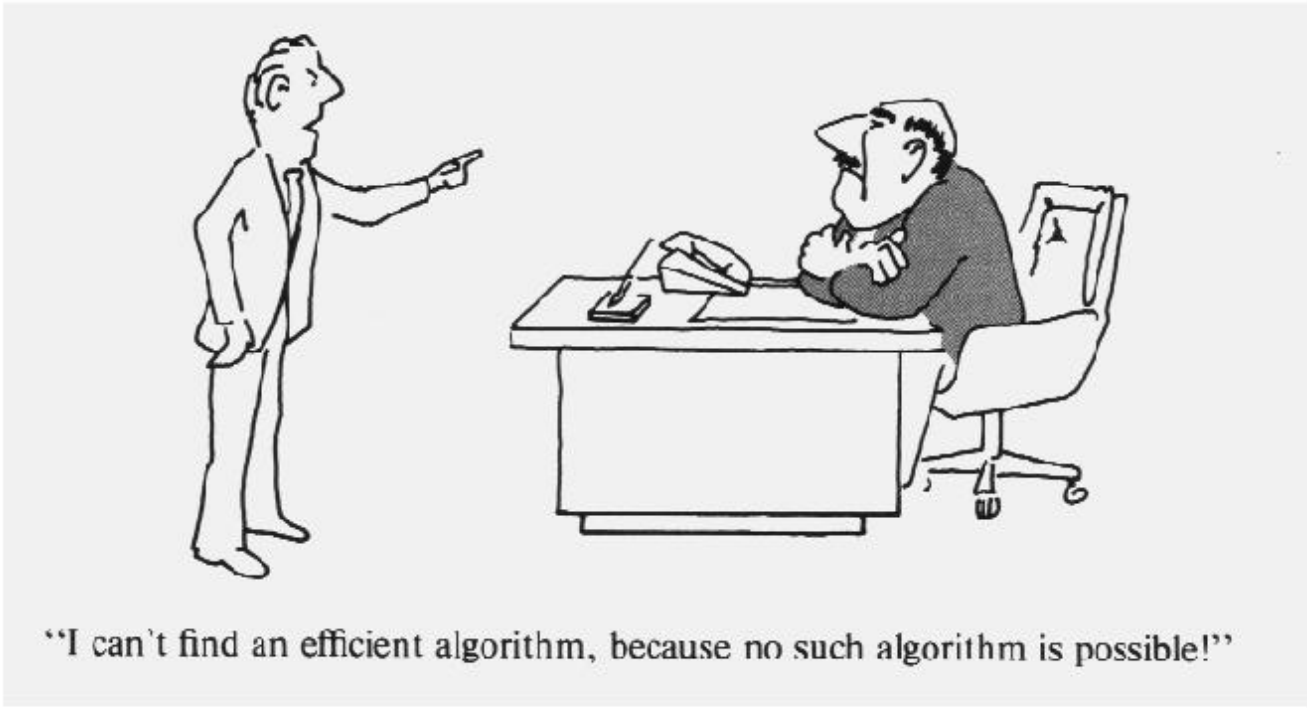
NP-Vollständigkeit als Ausrede



"I can't find an efficient algorithm, I guess I'm just too dumb."

Garey and Johnson. Computers and Intractability.

NP-Vollständigkeit als Ausrede



"I can't find an efficient algorithm, because no such algorithm is possible!"

Garey and Johnson. Computers and Intractability.

NP-Vollständigkeit als Ausrede

"I can't find an efficient algorithm, but neither can all these famous people."

Garey and Johnson. Computers and Intractability.

Übersicht

Einfache und schwere Instanzen

- Exponentielle Laufzeit im Worst Case
- Sie muß nur für wenige Instanzen groß sein
- Praktisch vorkommende Instanzen können einfach sein
- Wie können wir schwere und einfache Instanzen unterscheiden?

Parameter

Wir ordnen jeder Instanz eine Zahl, den Parameter zu.
Hoffnung:

- Bei kleinem Parameter schnelle Laufzeit
- Praktische Instanzen haben kleinen Parameter

Kein Widerspruch zur NP-Vollständigkeit des Problems!

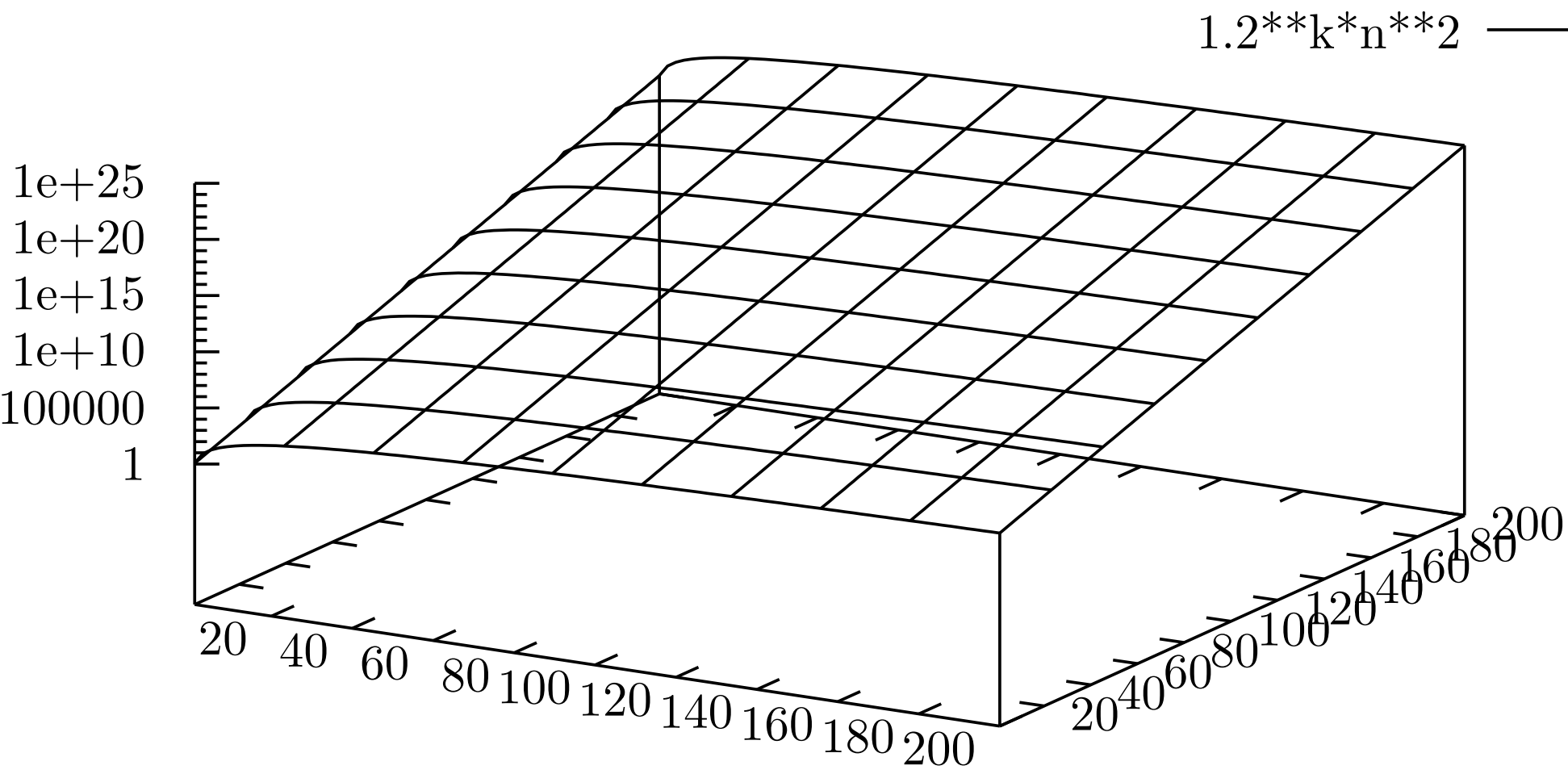
Zentrale Definition

Gegeben ist ein Problem.
Sei n die Größe einer Instanz und k der zugehörige Parameter.
Das Problem ist fixed parameter tractable, wenn es einen Algorithmus gibt, der das Problem löst und dessen Laufzeit

$$O(f(k)n^c)$$

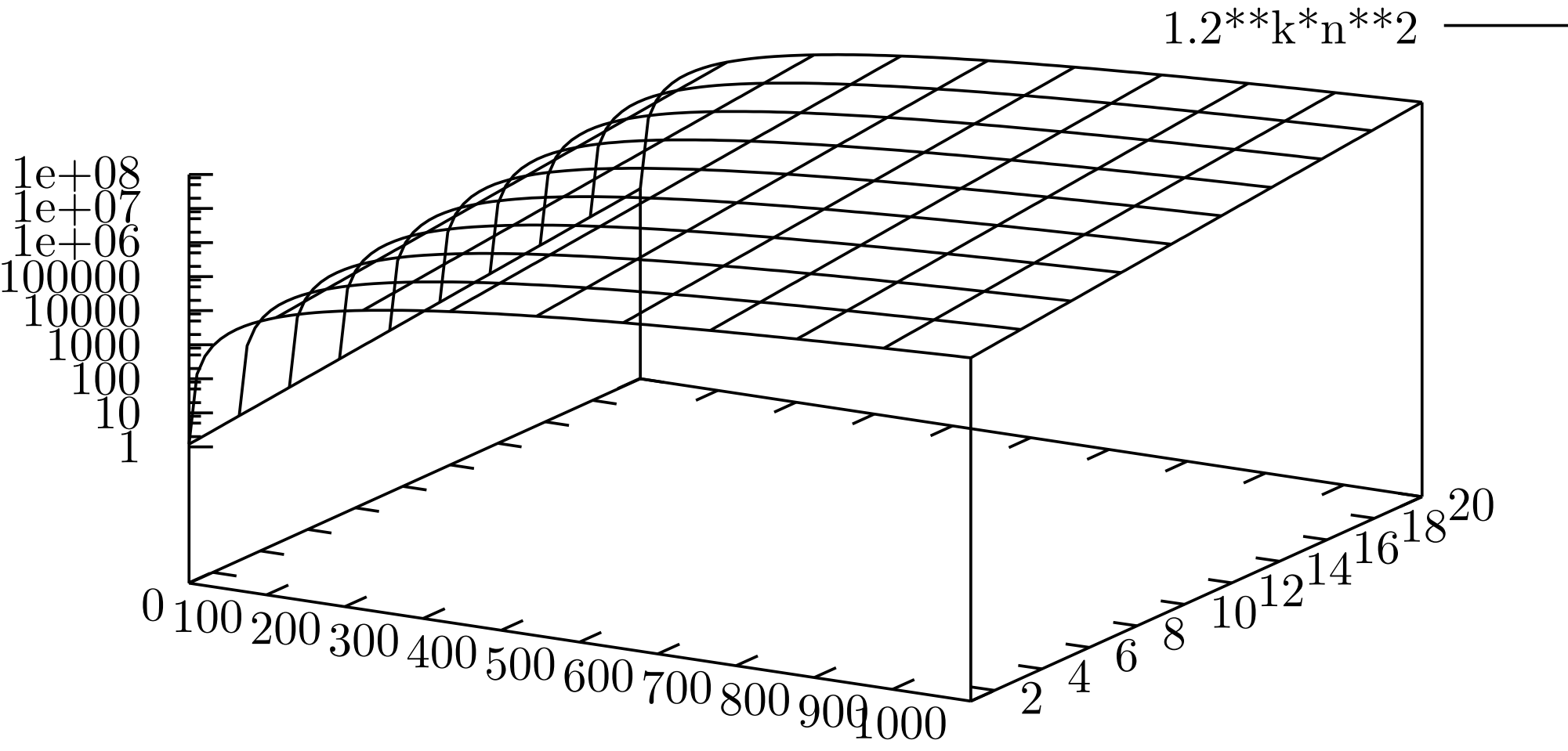
ist.
Dabei ist c eine Konstante und f eine beliebige Funktion.

Laufzeit eines parametrisierten Algorithmus



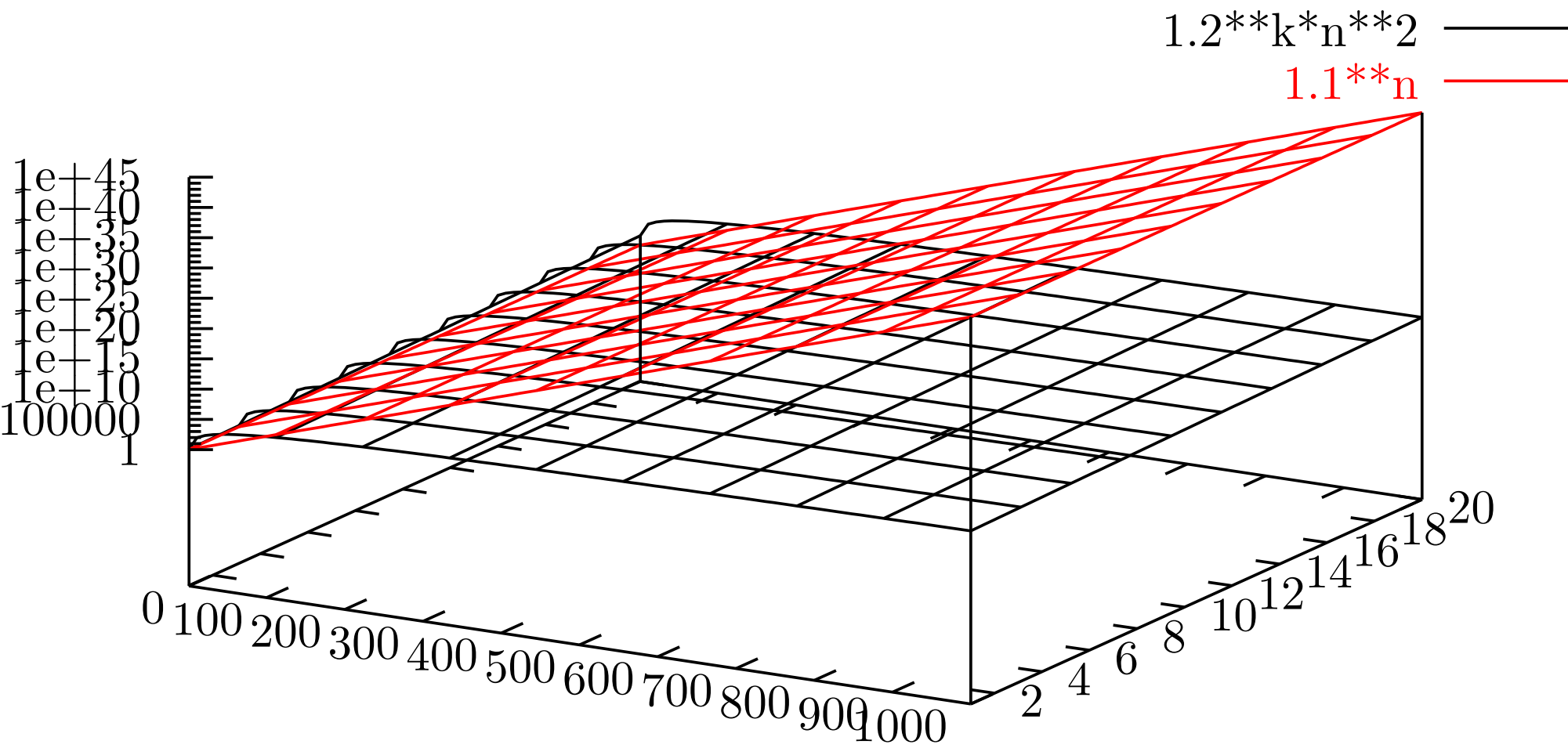
Laufzeit ist $1.2^k n^2$. Der Parameter liegt zwischen 1 und n.

Laufzeit eines parametrisierten Algorithmus



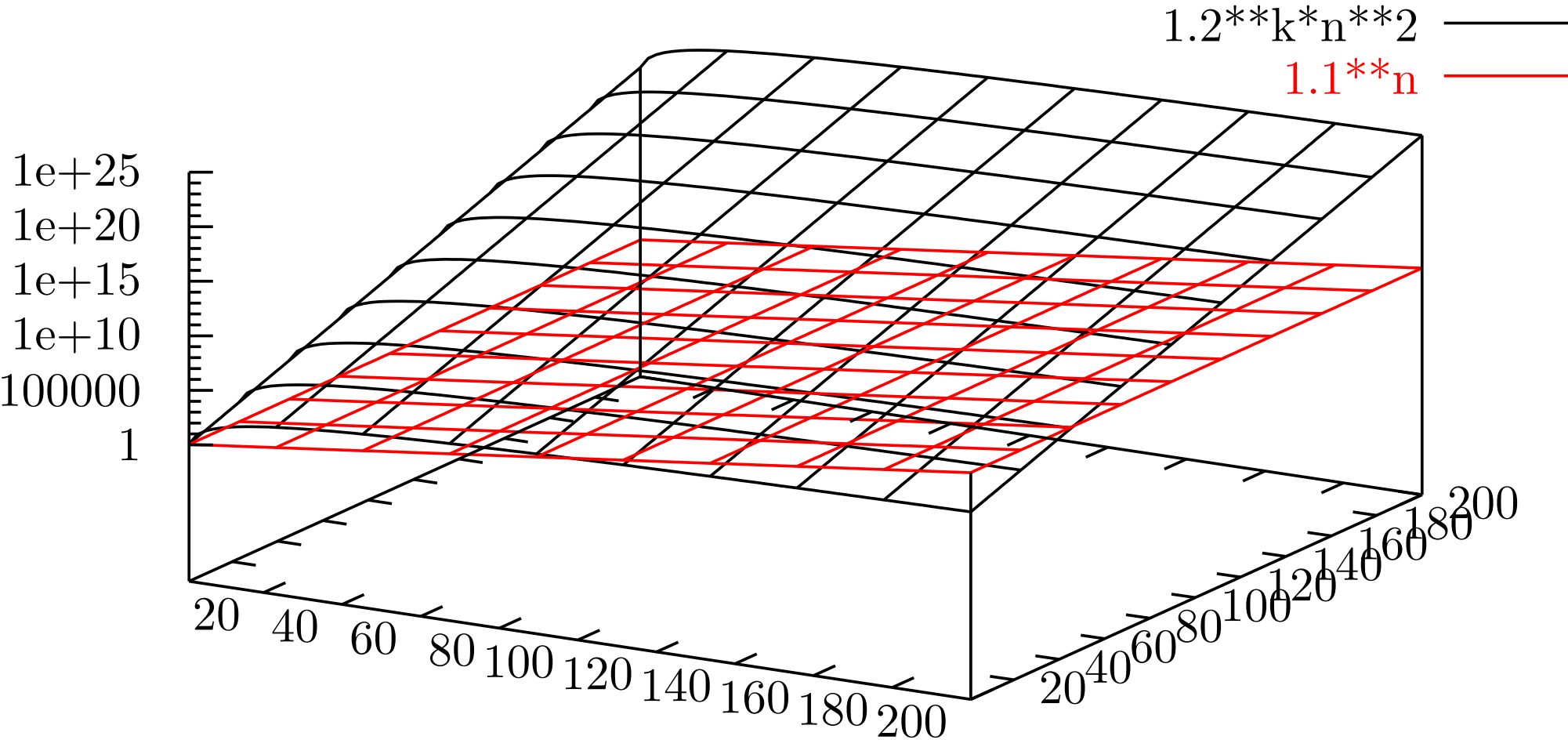
Laufzeit ist $1.2^k n^2$. Der Parameter ist klein.

Laufzeit eines parametrisierten Algorithmus



Laufzeit ist $1.2^k n^2$. Der Parameter ist klein. Der nicht-parametrisierte Algorithmus hat Laufzeit 1.1^n.

Laufzeit eines parametrisierten Algorithmus



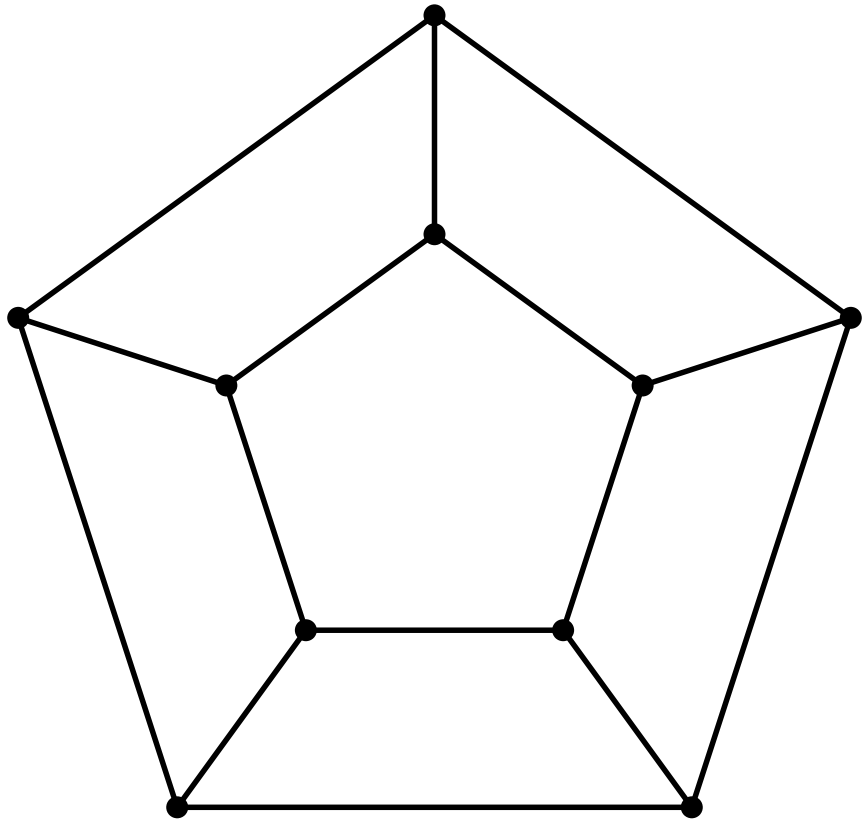
Beispiel: Vertex Cover

Gegeben: Ein Graph $G = (V, E)$.
Gesucht: Ein minimales Vertex Cover $C \subseteq E$.

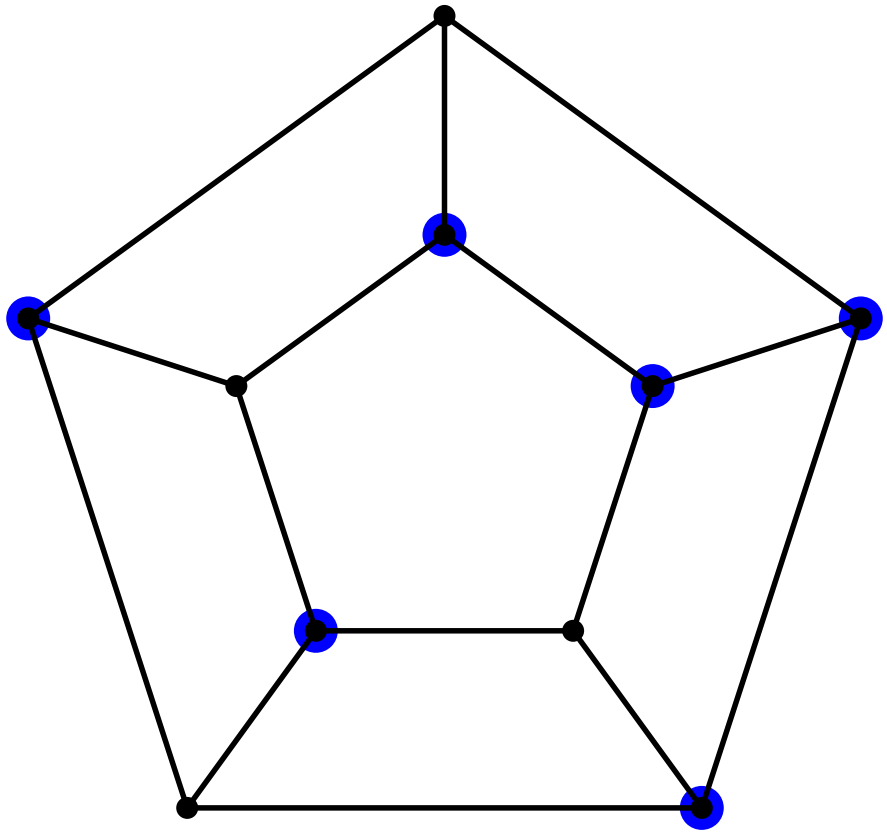
Definition

Ein Menge $C \subseteq V$ ist ein Vertex Cover von $G = (V, E)$, falls mindestens ein Endpunkt jeder Kante in E in C enthalten ist.

Beispiel



Beispiel



Modellierung als ILP

Gegeben ist $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}$.

$$
\begin{aligned}
&\text{Minimiere } v_1 + \ldots + v_n \\
&\quad \text{unter } 0 \leq v_i \leq 1 \text{ für } i = 1, \ldots, n \\
&\qquad v_i + v_j \geq 1 \text{ für } \{v_i, v_j\} \in E \\
&\qquad v_i \in \mathbf{Z} \text{ für } i = 1, \ldots, n
\end{aligned}
$$

Jedes NP-vollständige Problem läßt sich auf ein ILP zurückführen (aber meist ist das keine gute Idee).

British Museum Method

Wichtige NP-vollständige Probleme sind oft Suchprobleme. In einem (sehr großen) Suchraum verbergen sich die Lösungen.
Eine mögliche Lösungsstrategie ist es daher, den Lösungsraum vollständig zu durchsuchen.
Für Vertex Cover bedeutet dies, alle $C \subseteq V$ durchzuprobieren.
Das sind $2^{|V|}$ Möglichkeiten.
Die Laufzeit beträgt $O(|E|2^{|V|})$.

Backtracking

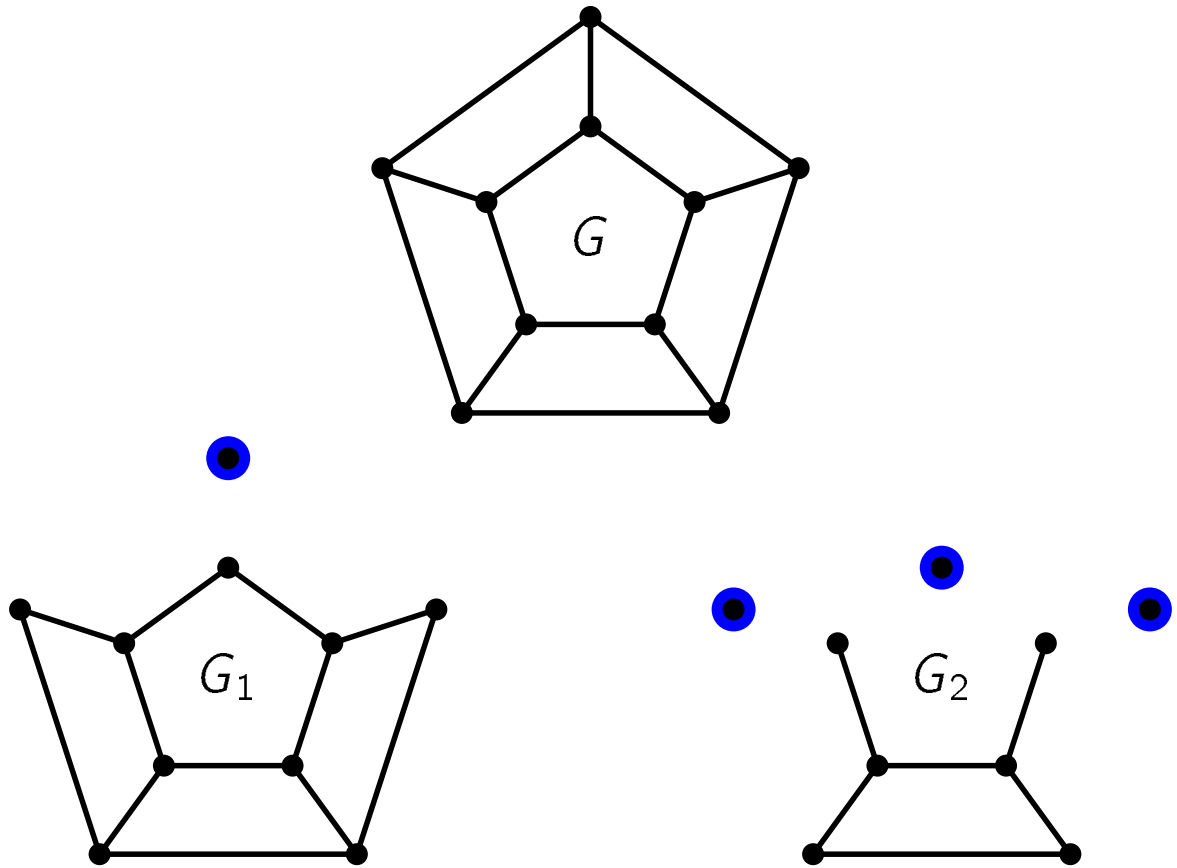
Wir betrachten einen Knoten $v \in V$.
Es gibt zwei Möglichkeiten $v \in C$ oder $v \notin C$.
Falls $v \notin C$, dann $N(v) \subseteq C$, denn die zu inzidenten Kanten müssen ja abgedeckt werden.
(Mit $N(v)$ bezeichnen wir die zu v adjazenten Knoten.)
Aus diesen Beobachtungen ergibt sich ein einfacher Algorithmus.

Backtracking

Eingabe: $G = (V, E)$
Ausgabe: Optimales Vertex Cover $VC(G)$

if $V = \emptyset$ **then return** $\emptyset$
Wähle einen beliebigen Knoten $v \in G$
$G_1 := (V - \{v\}, \{ e \in E \mid v \notin e \})$
$G_2 := (V - \{v\} - N(v), \{ e \in E \mid e \cap N(v) = \emptyset \})$
if $|\{v\} \cup VC(G_1)| \leq |N(v) \cup VC(G_2)|$
then return $\{v\} \cup VC(G_1)$
else return $N(v) \cup VC(G_2)$

Backtracking



Backtracking (andere Möglichkeit)

Jede Kante $e = \{v_1, v_2\}$ muß von v_1 oder v_2 überdeckt werden. Wir betrachten also eine beliebige Kante $\{v_1, v_2\}$ und testen rekursiv beide Möglichkeiten:

- $v_1 \in C$
- $v_2 \in C$

Wieder ergibt sich ein einfacher Algorithmus.

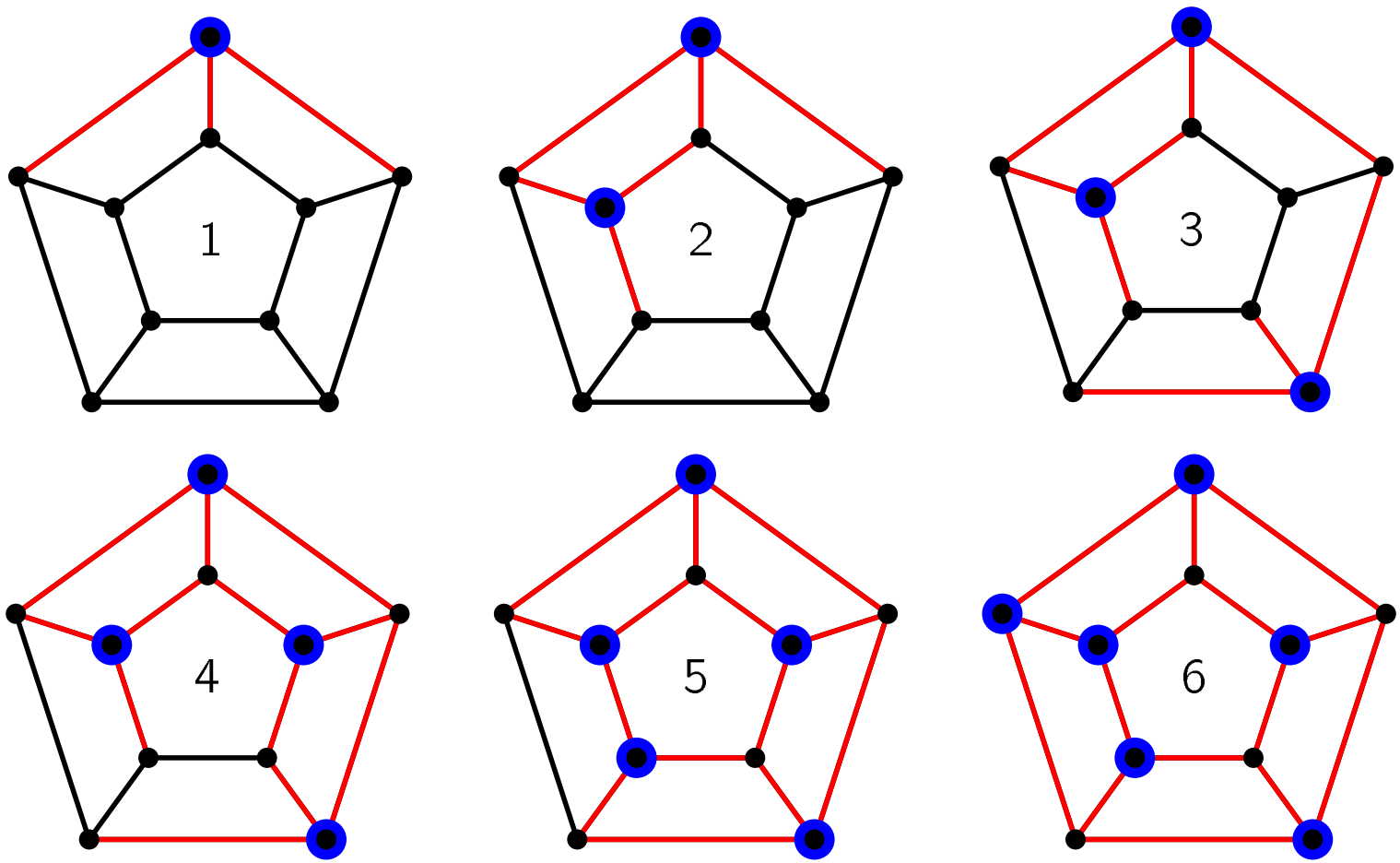
Backtracking (andere Möglichkeit)

Eingabe: $G = (V, E)$
Ausgabe: Optimales Vertex Cover $VC(G)$

if $E = \emptyset$ **then return** $\emptyset$
Wähle beliebige Kante $\{v_1, v_2\} \in E$
$G_1 := (V - \{v_1\}, \{ e \in E \mid v_1 \notin e \})$
$G_2 := (V - \{v_2\}, \{ e \in E \mid v_2 \notin e \})$
if $|\{v_1\} \cup VC(G_1)| \leq |\{v_2\} \cup VC(G_2)|$
then return $\{v_1\} \cup VC(G_1)$
else return $\{v_2\} \cup VC(G_2)$

Der rekursive Algorithmus berechnet ein optimales Vertex Cover.

Heuristiken



Wähle immer einen Knoten mit maximalem Grad (greedy).

Approximationsalgorithmen

Jede Kante muß von mindestens einem ihrer Knoten abgedeckt werden.
Problem: Von welchem?
Lösung: Wir nehmen beide.

- Natürlich gibt es so keine Garantie, daß wir eine optimale Lösung finden.
- Das so gefundene Vertex Cover kann aber höchstens doppelt so groß sein wie ein optimales.

Approximationsalgorithmen

Der Algorithmus könnte so aussehen:

$C := \emptyset$;
while $E \neq \emptyset$ **do**
  Wähle ein $e \in E$;
  $V := V - e$;
  $C := C \cup e$;
  $E := \{ e' \in E \mid e \cap e' = \emptyset \}$
od;
return C

Parametrisierter Algorithmus

Eingabe: $G = (V, E), k$
Ausgabe: Vertex Cover $VC(G, k)$ der Größe k oder kleiner, falls eines existiert.

if $E = \emptyset$ **then return** $\emptyset$
if $k = 0$ **then return** „gibt es nicht“
Wähle beliebige Kante $\{v_1, v_2\} \in E$
$G_1 := (V - \{v_1\}, \{ e \in E \mid v_1 \notin e \})$
$G_2 := (V - \{v_2\}, \{ e \in E \mid v_2 \notin e \})$
if $|\{v_1\} \cup VC(G_1, k-1)| \leq |\{v_2\} \cup VC(G_2, k-1)|$
then return $\{v_1\} \cup VC(G_1, k-1)$
else return $\{v_2\} \cup VC(G_2, k-1)$

Parametrisierter Algorithmus

Eingabe: $G = (V, E), k$
Ausgabe: Vertex Cover $VC(G, k)$ der Größe k oder kleiner, falls eines existiert.

if $E = \emptyset$ **th**
if $k = 0$ **the**
Wähle belie
$G_1 := (V -$
$G_2 := (V -$
if $|\{v_1\} \cup V$
then retur
else return

Fragen:

1. Was ist |„gibt es nicht“|?
2. Warum ist die Laufzeit $O(f(k)n^c)$?
3. Was ist hier $f(k)$?
4. Findet der Algorithmus immer ein optimales Vertex Cover?
5. Lassen sich die letzten Zeilen vereinfachen?

Parametrisierter Algorithmus

Eingabe: $G = (V, E), k$
Ausgabe: Vertex Cover $VC(G, k)$ der Größe k oder kleiner, falls eines existiert.

if $E = \emptyset$ **then return** $\emptyset$
if $k = 0$ **then return** „gibt es nicht“
Wähle beliebige Kante $\{v_1, v_2\} \in E$
$G_1 := (V - \{v_1\}, \{ e \in E \mid v_1 \notin e \})$
$G_2 := (V - \{v_2\}, \{ e \in E \mid v_2 \notin e \})$
if $VC(G_1, k-1) \neq$ „gibt es nicht“
then return $\{v_1\} \cup VC(G_1, k-1)$
else return $\{v_2\} \cup VC(G_2, k-1)$

Parametrisierter Algorithmus — Laufzeit

Jeder rekursive Aufruf benötigt nur polynomielle Zeit.
Wieviele solche Aufrufe gibt es?
Jede Inkarnation ist ein Blatt im Rekursionsbaum oder hat zwei Kinder.

- Die Wurzel hat Parameter k
- Der Parameter der Kinder ist um eins kleiner als der des Elternknotens
- Der Parameter wird nicht negativ

Also ist die Höhe des Baums höchstens k
Seine Größe ist daher höchstens 2^k.

Der lange Weg zu Vertex Cover

- Fellows & Langston (1986): $O(f(k)n^3)$
- Robson (1986): $O(1.211^n)$
- Johnson (1987): $O(f(k)n^2)$
- Fellows (1988): $O(2^k n)$
- Buss (1989): $O(kn + 2^k k^{2k+2})$
- Downey, Fellows, & Raman (1992): $O(kn + 2^k k^2)$
- Balasubramanian, Fellows, & Raman (1996): $O(kn + 1.3333^k k^2)$

Der lange Weg zu Vertex Cover

- Balasubramanian, Fellows, & Raman (1998): $O(kn + 1.32472^k k^2)$
- Downey, Fellows, Stege (1998): $O(kn + 1.31951^k k^2)$
- Niedermeier & R. (1998): $O(kn + 1.292^k)$
- Chen, Kanj, & Jia (1999): $O(kn + 1.271^k k^2)$
- Chen, Kanj, & Jia (2001): $O(kn + 1.285^k)$
- Niedermeier & R. (2001): $O(kn + 1.283^k)$

Beschränkte Suchbäume

Ein Beschränkter Suchbaum–Algorithmus muß diese Bedingungen für den Baum der rekursiven Aufrufe erfüllen:

- Jeder Knoten ist durch eine natürliche Zahl markiert
- Die Markierung der Wurzel ist eine Funktion des Parameters
- Die Anzahl der Kinder eines Knotens ist durch eine Funktion der Markierung begrenzt
- Die Markierungen der Kinder sind kleiner als die Markierung des Elternknotens

Korrektheit

Theorem

Gegeben sei ein Algorithmus, der auf beschränkten Suchbäumen basiert.
Dann gibt es eine Funktion f, so daß alle Suchbäume für Eingaben mit Parameter k höchstens $f(k)$ viele Knoten haben.

Beweis der Korrektheit

Beweis.

Wir definieren eine Funktion $S(k)$, die eine obere Schranke für die Anzahl der Blätter ist, die ein Unterbaum des Suchbaums hat, dessen Wurzel mit k markiert ist.

- Die Wurzel sei mit höchstens $w(k)$ markiert
- Jeder Knoten mit Markierung k habe höchstens $b(k)$ viele Kinder
- Die Existenz von w und b ist durch die Definition eines beschränkten Suchbaums garantiert

□

Beweis der Korrektheit (Fortsetzung)

Beweis.

Es gilt

$$S(k) \leq b(k)S(k-1),$$

denn es gibt höchstens $b(k)$ Kinder, deren Unterbäume höchstens je $S(k-1)$ viele Blätter haben.
Mit der Anfangsbedingung $S(0) = 1$ ist die Lösung der Rekursionsungleichung

$$S(k) \leq \prod_{i=1}^{k} b(i).$$

Die Gesamtzahl der Blätter ist höchstens $S(w(k))$, eine Funktion von k. $\square$

Beispiel Center String

Seien u und v zwei Strings der Länge n.
Wir definieren $h(u, v)$, genannt Hamming-Abstand von u und v, als die Anzahl der Positionen, in welchen sich u und v unterscheiden.
Beispiel:

$$h(agctcagtaccc, agctcataacgc) = 3$$

Beispiel Center String

Das Center-String-Problem ist so definiert:

Eingabe: k Strings $s_1, \ldots, s_k$ der Länge n, eine Zahl m

Frage: Gibt es einen String s mit $h(s, s_i) \leq m$ für alle $1 \leq i \leq k$.

Der Parameter sei m

Motivation: Einen Marker konstruieren, der gut zu gegebenen DNA-Sequenzen passt
In der Praxis ist m klein, z.B. 5

Beispiel Center String

agcacagtacgcaatagtgtcgcaggt
agctcagtagccaatagagtcccaggt
agatcagttcccaatagagtcgcacgt
agctcagtaaaaaatagagtcgcaggt
agcgcagtacacaatagagtcgcaagt

Beispiel Center String

agcacagtacgcaatagtgtcgcaggt

agctcagtagccaatagagtcccaggt

agatcagttcccaatagagtcgcacgt

agctcagtaaaaaatagagtcgcaggt

agcgcagtacacaatagagtcgcaagt

---

agctcagtacccaatagagtcgcaggt

Beispiel Center String

gctaggagtcagaagtaggcgttgcat

gcaatgaatcagaactgggcctagcat

gctagggatcagaactaggcctagcat

gcaaggaatcataactaggcctagcat

gcaaggaattagaaataggcctagcat

gcaagaaatcagaactagccctagcat

Beispiel Center String

gctaggagtcagaagtaggcgttgcat

gcaatgaatcagaactgggcctagcat

gctagggatcagaactaggcctagcat

gcaaggaatcataactaggcctagcat

gcaaggaattagaaataggcctagcat

gcaagaaatcagaactagccctagcat

---

gcaaggagtcagaactaggcctagcat

Algorithmus für Center String

Gegeben $s_1, \ldots, s_k$, Zahl m.
Algorithmus center(s, l) stellt fest, ob es s' gibt, mit

- $h(s, s') \leq l$
- $h(s', s_i) \leq m$ für $1 \leq i \leq k$

Mit center ist Center String leicht lösbar:
Rufe center(s_1, m) auf!

Algorithmus für Center String

So implementieren wir center(s, l):
Wähle einen String s_i mit $h(s, s_i) > m$.
(Falls keiner existiert, ist s eine Lösung des Problems und wir sind fertig)
Wähle eine Menge P von $m + 1$ Positionen, in welchen sich s und s_i unterscheiden.
Für jede Position p in P, bilde s' aus s, aber setze Position p von s' wie in s_i.
Rufe jeweils center$(s', l - 1)$ auf. Falls ein Aufruf Ja liefert, antworte mit Ja.

Algorithmus für Center String

Die Größe des Suchbaums ist höchstens $(m+1)^m$.

- Die Wurzel ist mit m markiert
- Die Markierung der Kinder ist kleiner als die der Eltern
- Ist die Markierung 0, finden wir die Antwort in polynomieller Zeit
- Jeder Knoten hat nur $m+1$ Kinder

Der Algorithmus ist effizient und praktikabel.

Algorithmus für Center String

Es war schon lange bekannt, daß dieses Problem *fixed parameter tractable* ist, falls k und m *beide* Parameter sind.

Die Größe de

- Die Wurzel ist mit m markiert
- Die Markierung der Kinder ist kleiner als die der Eltern
- Ist die Markierung 0, finden wir die Antwort in polynomieller Zeit
- Jeder Knoten hat nur $m + 1$ Kinder

Der Algorithmus ist effizient und praktikabel.

Algorithmus für Center String

Es war schon lange bekannt, daß dieses Problem *fixed parameter tractable* ist, falls k und m *beide* Parameter sind.

Die Größe de

- Die Wurzel ist mit m markiert
- Die Markierung der Kinder ist kleiner als die der Eltern
- Is
  Z
- J

Für die Praxis ist m der interessante Parameter.
Es ist aber auch interessant, k als Parameter zu betrachten.
Frage: Ist Center String fixed parameter tractable, falls k der Parameter ist?
(Beide Fragen, k und m, waren noch vor kurzem offene Probleme)

Der A

Analyse von beschränkten Suchbäumen

Falls

1. die Wurzel eines Baums mit k markiert ist,
2. jeder Knoten höchstens zwei Kinder hat,
3. keine Markierung negativ ist,
4. die Markierungen von Kindern kleiner als die der Eltern sind,

dann ist klar, daß der Baum höchstens 2^k Blätter hat.
Wie läßt sich dieses offensichtliche Ergebnis verallgemeinern?

Verzweigungsvektoren

Falls jeder Knoten zwei Kinder hat und deren Markierung um eins kleiner ist, entspricht das der Rekursionsgleichung

$$B_k = B_{k-1} + B_{k-1}.$$

Der zugehörige Verzweigungsvektor ist $(1, 1)$.
Einer Rekursionsgleichung

$$B_k = B_{k-z_1} + B_{k-z_2} + \cdots + B_{k-z_m}$$

entspricht der Verzweigungsvektor $(z_1, \ldots, z_m)$.
Mit Verzweigungsvektoren lassen sich beschränkte Suchbäume kurz beschreiben.

Verzweigungsvektoren

Falls in einem beschränten Suchbaum die Verzweigungsvektoren $(1, 1)$ und $(2, 2, 3)$ vorkommen können, erhalten wir die Rekursionsgleichung

$$B_k = \max\{2B_{k-1}, 2B_{k-2} + B_{k-3}\}.$$

Wir möchten Suchbaumalgorithmen analysieren, in welchen verschiedene Verzweigungsvektoren vorkommen können. Dazu müssen wir solche Rekursionsgleichungen lösen können.

Lineare Rekursionsgleichungen mit konstanten Koeffizienten

Die Rekursionsgleichung, die einem Verzweigungsvektor beziehungsweise dem Verhältnissen von Markierungen in einem Suchbaum entspricht, ist eine lineare Rekursionsgleichungen mit konstanten Koeffizienten.
Ihre allgemeine Form ist

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_t a_{n-t} \text{ für } n \geq t.$$

Wir entwickeln nun ein einfaches Schema, solche Rekursionsgleichungen zu lösen.

Lineare Rekursionsgleichungen mit konstanten Koeffizienten

Nehmen wir zunächst einmal an, daß es eine Lösung gibt, welche die Form $a_n = \alpha^n$ hat, wobei $\alpha \in \mathbf{C}$ eine komplexe Zahl sein darf. Setzen wir dies in die Rekursionsgleichung ein und setzen $n = t$, erhalten wir

$$\alpha^t = c_1\alpha^{t-1} + c_2\alpha^{t-2} + \cdots + c_{t-1}\alpha + c_t$$

und das bedeutet natürlich, daß α eine Nullstelle des *charakteristischen Polynoms*

$$\chi(z) = z^t - c_1 z^{t-1} - c_2 z^{t-2} - \cdots - c_{t-1}z - c_t$$

sein muß.

Lineare Rekursionsgleichungen mit konstanten Koeffizienten

Umgekehrt ist aber auch $a_n = \alpha^n$ eine Lösung der Rekursionsgleichung, wenn α eine Nullstelle eine Nullstelle von

$$\chi(z) = z^t - c_1 z^{t-1} - c_2 z^{t-2} - \cdots - c_{t-1} z - c_t$$

ist.
Man sieht es leicht, wenn man in die Rekursionsgleichung einsetzt:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_t a_{n-t}$$

Lineare Rekursionsgleichungen mit konstanten Koeffizienten

Wenn α eine k-fache Nullstelle von χ ist, dann ist auch $a_n = n^j\alpha^n$ für $0 \leq j < k$ eine Lösung der Rekursionsgleichung. Wir überprüfen dies durch Einsetzen:

$$n^j\alpha^n = \sum_{r=1}^{t} c_r(n-r)^j\alpha^{n-r} \text{ bzw. } n^j\alpha^t - \sum_{r=1}^{t} c_r(n-r)^j\alpha^{t-r} = 0.$$

Die linke Seite ist aber eine Linearkombination von $\chi(\alpha)$, $\chi'(\alpha)$, $\chi''(\alpha)$, $\ldots$, $\chi^{(j)}(\alpha)$. Die ersten $k-1$ Ableitungen von χ sind an der Stelle α natürlich 0, da ja α eine k-fache Nullstelle von χ ist.

Lineare Rekursionsgleichungen mit konstanten Koeffizienten

Theorem

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_t a_{n-t} \ \textit{für}\ n \geq t$$

hat die Lösungen $a_n = n^j \alpha^n$, *für alle Nullstellen* α *des charakteristischen Polynoms*

$$\chi(z) = z^t - c_1 z^{t-1} - c_2 z^{t-2} - \cdots - c_{t-1} z - c_t,$$

und für alle $j = 0, 1, \ldots, k-1$, *wobei* k *die Vielfachheit der Nullstelle* α *ist. Diese Lösungen sind alle linear unabhängig. Sie bilden eine Basis des Vektorraums aller Lösungen.*

Größe von beschränkten Suchbäumen

Theorem

Ein Suchbaum mit Verzweigungsvektor $(r_1, \ldots, r_m)$*, dessen Wurzel mit* k *beschriftet ist, hat die Größe*

$$k^{O(1)}\alpha^k,$$

wobei α *die betragsmäßig größte Nullstelle des charakteristischen Polynoms*

$$\chi(z) = z^t - z^{t-r_1} - z^{t-2} - \ldots - z^{t-r_m}$$

ist, wobei $t = \max\{r_1, \ldots, r_m\}$.

Größe von beschränkten Suchbäumen

Beispiel:
Der Verzweigungsvektor $(1, 3)$ hat das charakteristische Polynom

$$z^3 - z^2 - 1.$$

Die größte Nullstelle ist etwa 1.465571.
Der Suchbaum hat daher die Größe $O(1.465572^k)$.

Größe von beschränkten Suchbäumen

Noch ein Beispiel:
Der Verzweigungsvektor $(1, 2, 2, 3, 6)$ hat das charakteristische Polynom

$$z^6 - z^5 - z^4 - z^4 - z^3 - 1.$$

Die größte Nullstelle ist etwa 2.160912.
Der Suchbaum hat daher die Größe $O(2.160913^k)$.

Das reflektierte charakteristische Polynom

Aus dem Verzweigungsvektor

$$(1, 2, 2, 3, 6)$$

das charakteristische Polynom

$$z^6 - z^5 - z^4 - z^4 - z^3 - 1$$

zu bestimmen, ist fehleranfällig.
Das reflektierte charakteristische Polynom lautet

$$1 - z - z^2 - z^2 - z^3 - z^6.$$

Das reflektierte charakteristische Polynom

Theorem

Das charakteristische Polynom hat die Nullstelle α genau dann, wenn das reflektierte charakteristische Polynom die Nullstelle $1/\alpha$ hat.

Das reflektierte charakteristische Polynom

Theorem

Ein Suchbaum mit Verzweigungsvektor $(r_1, \ldots, r_m)$, dessen Wurzel mit k beschriftet ist, hat die Größe

$$k^{O(1)}\alpha^{-k},$$

wobei α die betragsmäßig kleinste Nullstelle des reflektierten, charakteristischen Polynoms

$$\chi(z) = 1 - z^{r_1} - z^{r_2} - \cdots - z^{r_m}$$

ist.

Verzweigungszahlen

Definition

Zu einem Verzweigungsvektor gehört eine Verzweigungszahl, welche der Kehrwert der kleinsten Nullstelle des reflektierten charakteristischen Polynoms ist.

Theorem

Ein Suchbaum mit Verzweigungszahl α, dessen Wurzel mit k markiert ist, hat Größe

$$k^{O(1)}\alpha^k.$$

Falls die Nullstelle einfach ist, ist die Größe $O(\alpha^k)$.

Verzweigungszahlen — Beispiel 1

- Nehmen wir den einfachen Algorithmus für Vertex Cover.
- Der Verzweigungsvektor ist $(1, 1)$.
- Das reflektierte charakteristische Polynom ist $1 - 2z$.
- Die Verzweigungszahl ist 2.
- Die Größe des Suchbaums ist $O(2^k)$.

Verzweigungszahlen — Beispiel 2

- Falls in einem Graphen alle Knoten höchstens Grad 2 haben, läßt sich ein optimales Vertex Cover in polynomieller Zeit bestimmen.
- Ein verbesserter Algorithmus kann also einen Knoten mit Grad mindestens 3 wählen.
- Dies führt zu Suchbaum mit Verzweigungsvektor $(1, 3)$.
- Die zugehörige Verzweigungszahl ist etwa 1.465571.
- Die Größe des Suchbaums ist $O(1.465572^k)$.

Mehrere Verzweigungsvektoren

Theorem

Gegeben sei eine Menge von Verzweigungsvektoren M. Ein Suchbaum dessen Verzweigungen je einem Verzweigungsvektor aus M entsprechen und dessen Wurzel mit k markiert ist, hat die Größe

$$k^{O(1)}\alpha^k,$$

wobei α die größte Verzweigungszahl ist, die zu Verzweigungsvektoren aus M gehört.

Verbesserung der Laufzeit durch Fallunterscheidung

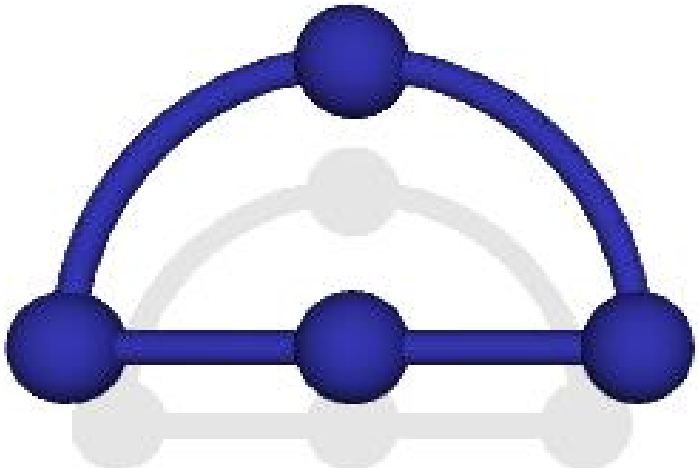
- Um Vertex Cover schneller lösen zu können, führen wir verschiedene Fälle ein.
- Zu jedem Fall wird ein korrekter Suchbaumschritt angegeben mit einem guten Verzweigungsvektor.
- Alle Fälle werden abgedeckt.
- Die größte Verzweigungszahl ergibt eine Schranke für die Größe des gesamten Suchbaums.

Überblick des Algorithmus

Fallunterscheidung:

- Knoten x mit Grad 1: $N(x)$
- Knoten x mit Grad ≥ 6: $N(x)$ und x
- Graph regulär: $N(x)$ und x für beliebiges x
- Grad zwischen 2 und 5 (weggelassen)
- Grad zwischen 3 und 5
- Grad zwischen 4 und 5

Brücken und Dreiecke



Brücke

Dreieck

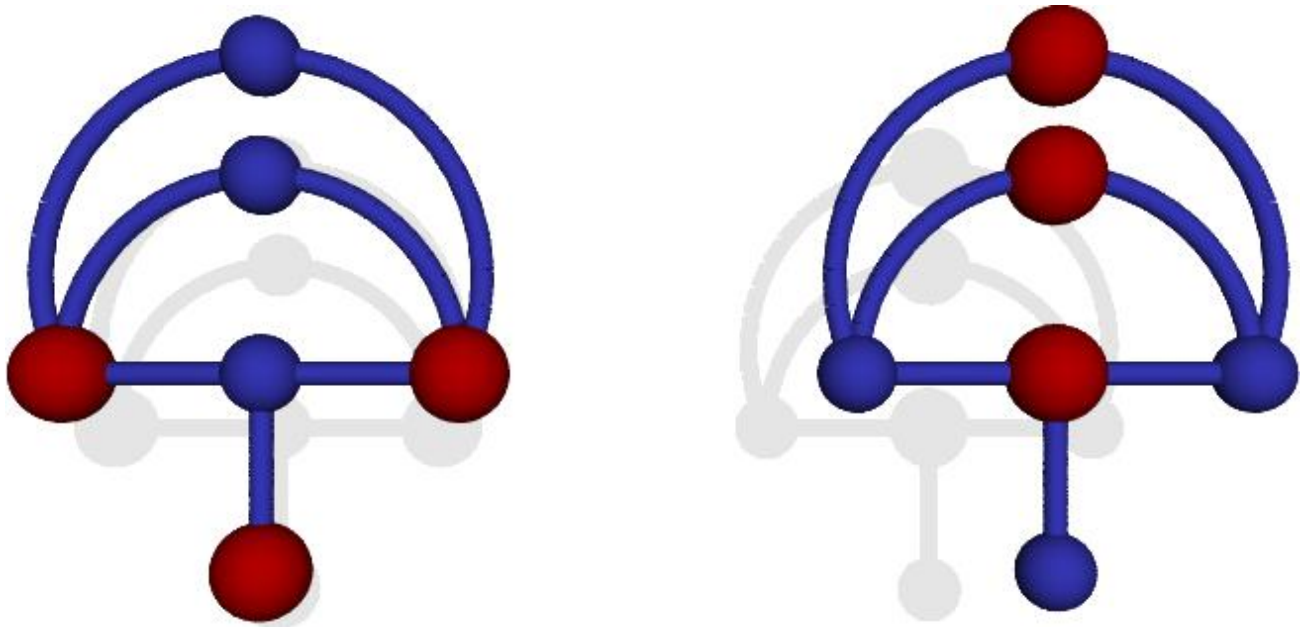
- Spielen große Rolle
- Dreiecke einfach
- 1. Fall: Es gibt Brücken
- 2. Fall: Es gibt keine Brücken

Grad 3

- Alle Knoten haben Grad zwischen 3 und 6
- Es gibt mindestens einen Knoten mit Grad 3
- Der Graph ist nicht regulär
- Der Graph ist zusammenhängend

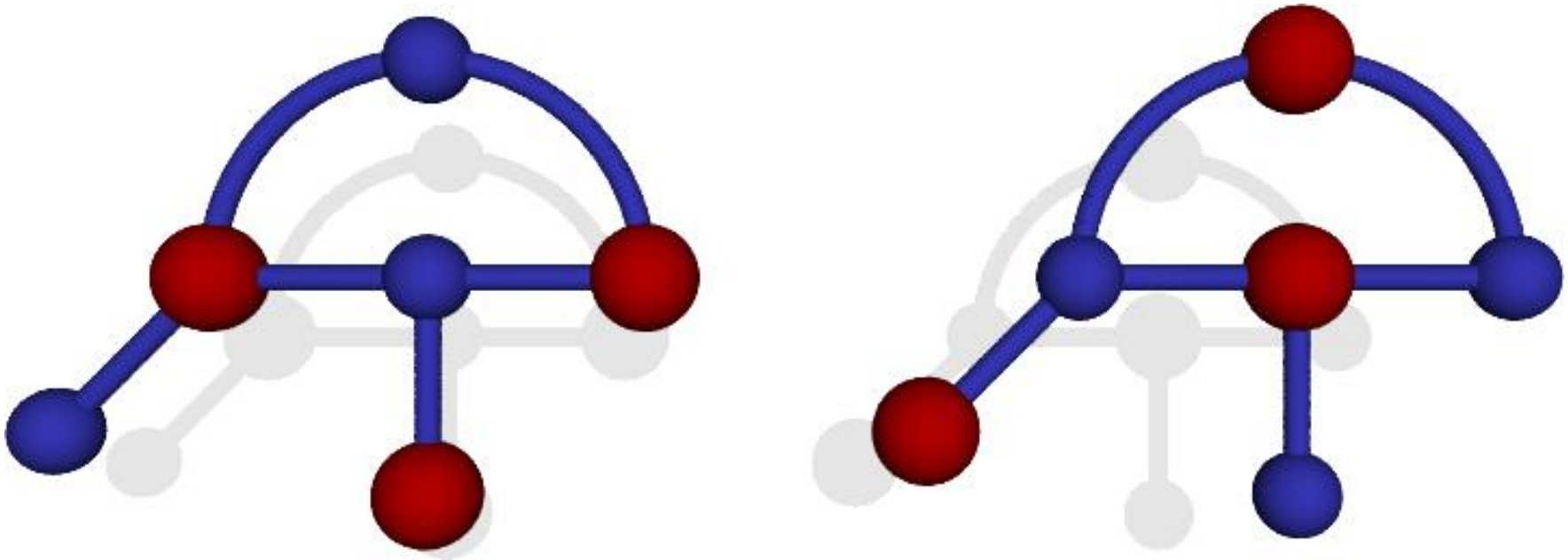
- Es gibt einen Knoten x mit Grad 3
- 1. Unterfall: x hat ein Dreieck (weggelassen)
- 2. Unterfall: x hat mindestens 2 Brücken
- 3. Unterfall: x hat genau 1 Brücke, ein Brückennachbar von x hat Grad 3
- 4. Unterfall: x hat genau 1 Brücke, beide Brückennachbarn von x haben Grad ≥ 4
- 5. Unterfall: Es gibt keine Knoten mit Grad 3, die Brücken oder Dreiecke haben

Grad 3 — 2 Brücken



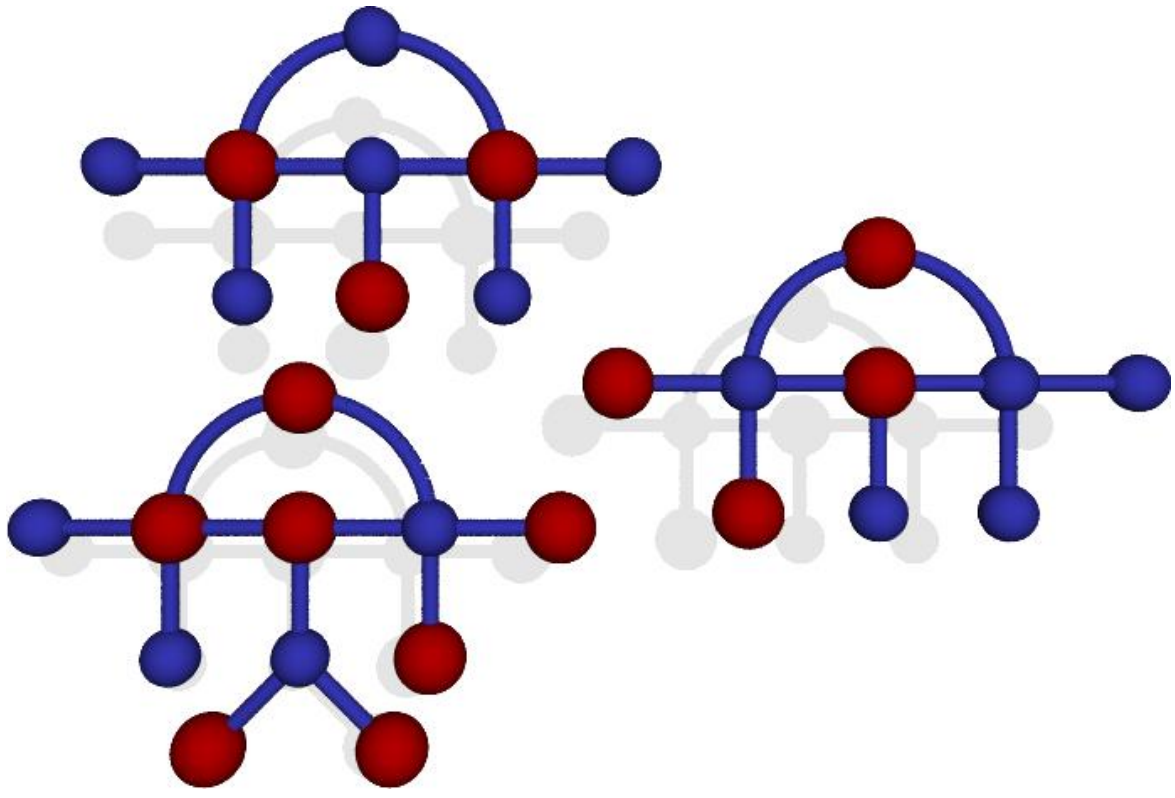
- Es gibt einen Knoten x mit Grad 3
- Er hat mindestens zwei Brücken

Grad 3 — 1 Brücke (a)



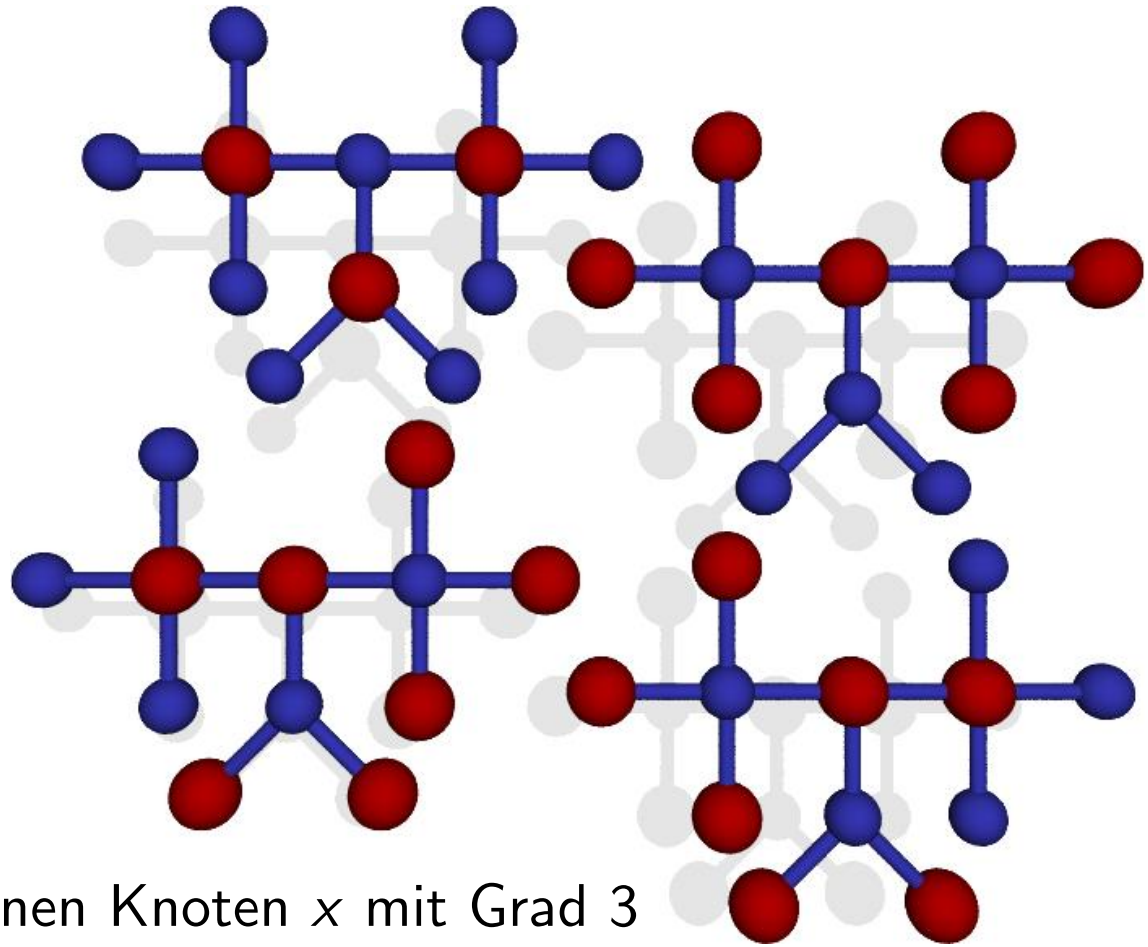
- Es gibt einen Knoten x mit Grad 3
- x hat genau eine Brücke
- x hat einen Brückennachbarn mit ebenfalls Grad 3

Grad 3 — 1 Brücke (b)



- Es gibt einen Knoten x mit Grad 3
- x hat genau eine Brücke
- Beide Brückennachbarn haben Grad ≥ 4

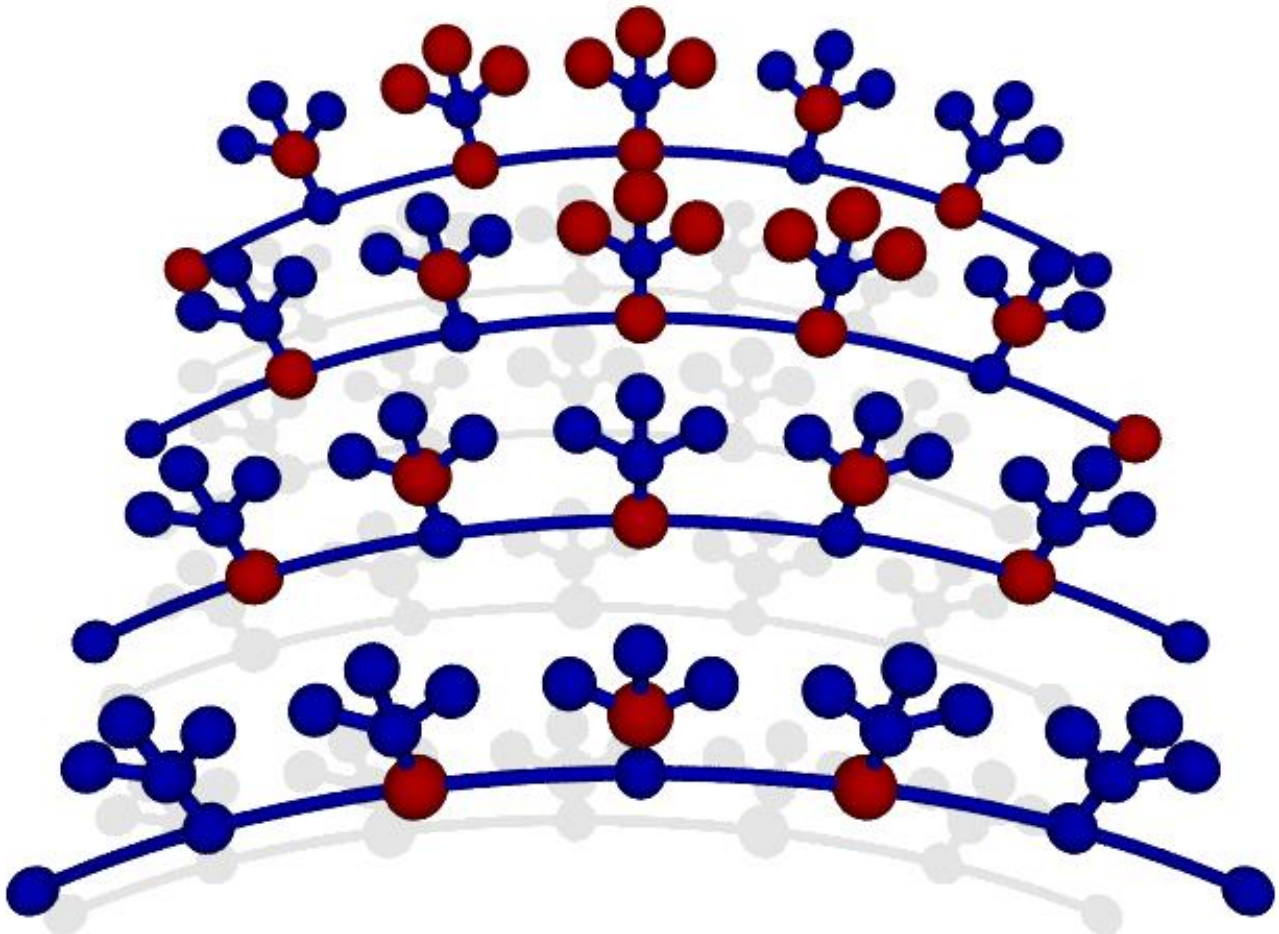
Grad 3 — Keine Brücken (a)



- Es gibt einen Knoten x mit Grad 3
- x hat keine Brücke
- Zwei Nachbarn haben Grad ≥ 4

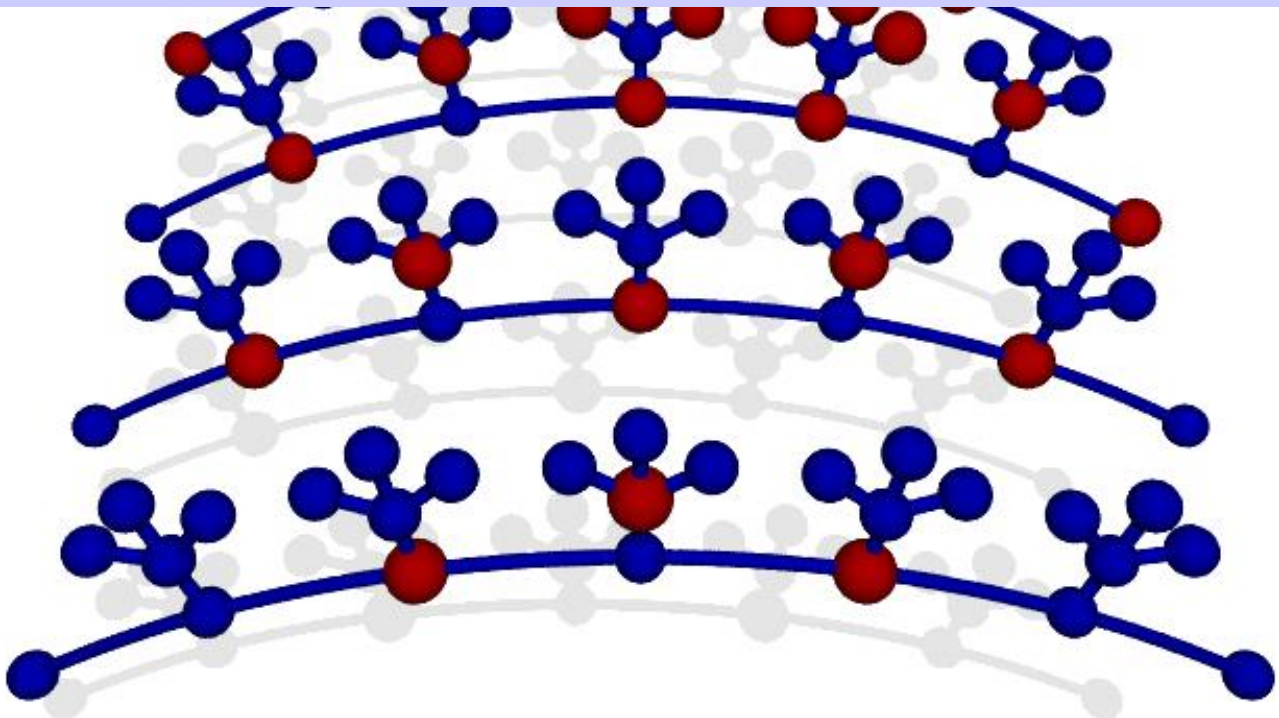
Grad 3 — Keine Brücken (b)

- Alle Knoten mit Grad 3 haben ≥ 2 Nachbarn mit Grad 3



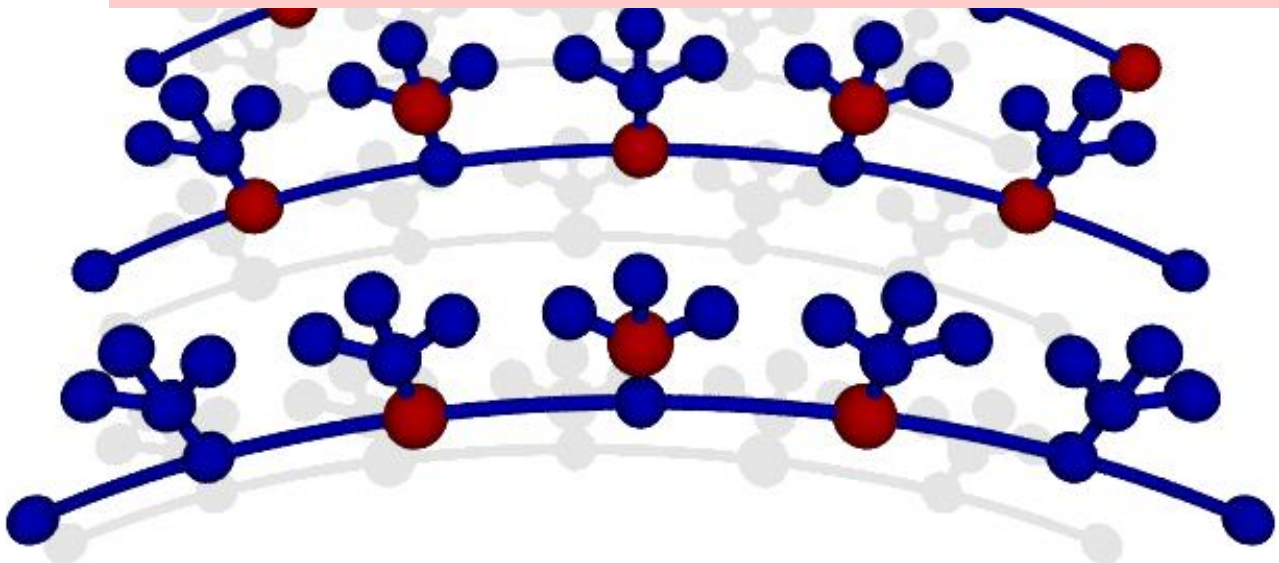
Grad 3 — Keine Brücken (b)

- A Weitere Annahme: Es gibt keine Kreise der Länge 5, die nur aus Knoten mit Grad 3 bestehen und einen Nachbarn mit Grad 4 besitzen.
  Gewünschter Branching-Vektor: $(3, 5, 8, 8)$

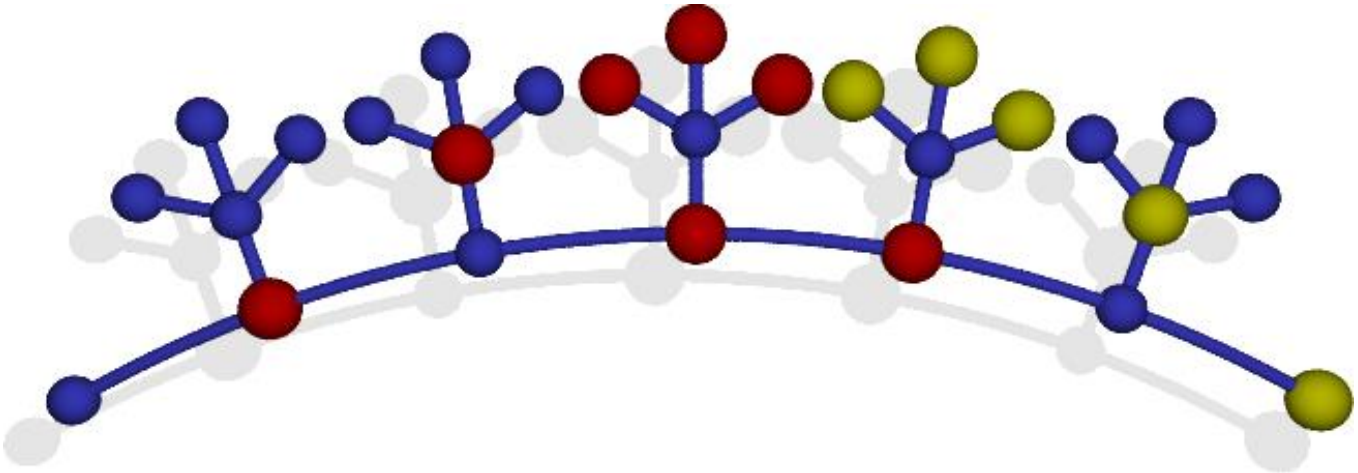


Grad 3 — Keine Brücken (b)

- A Weitere Annahme: Es gibt keine Kreise der Länge 5, die nur aus Knoten mit Grad 3 bestehen und einen Nachbarn mit Grad 4 besitzen.
  Gewü Weggelassen: Es gibt so einen Kreis der Länge 5.
  $\longrightarrow$ Übungsaufgabe!
  Verzweigungsvektor $(5, 5, 7, 8, 8)$



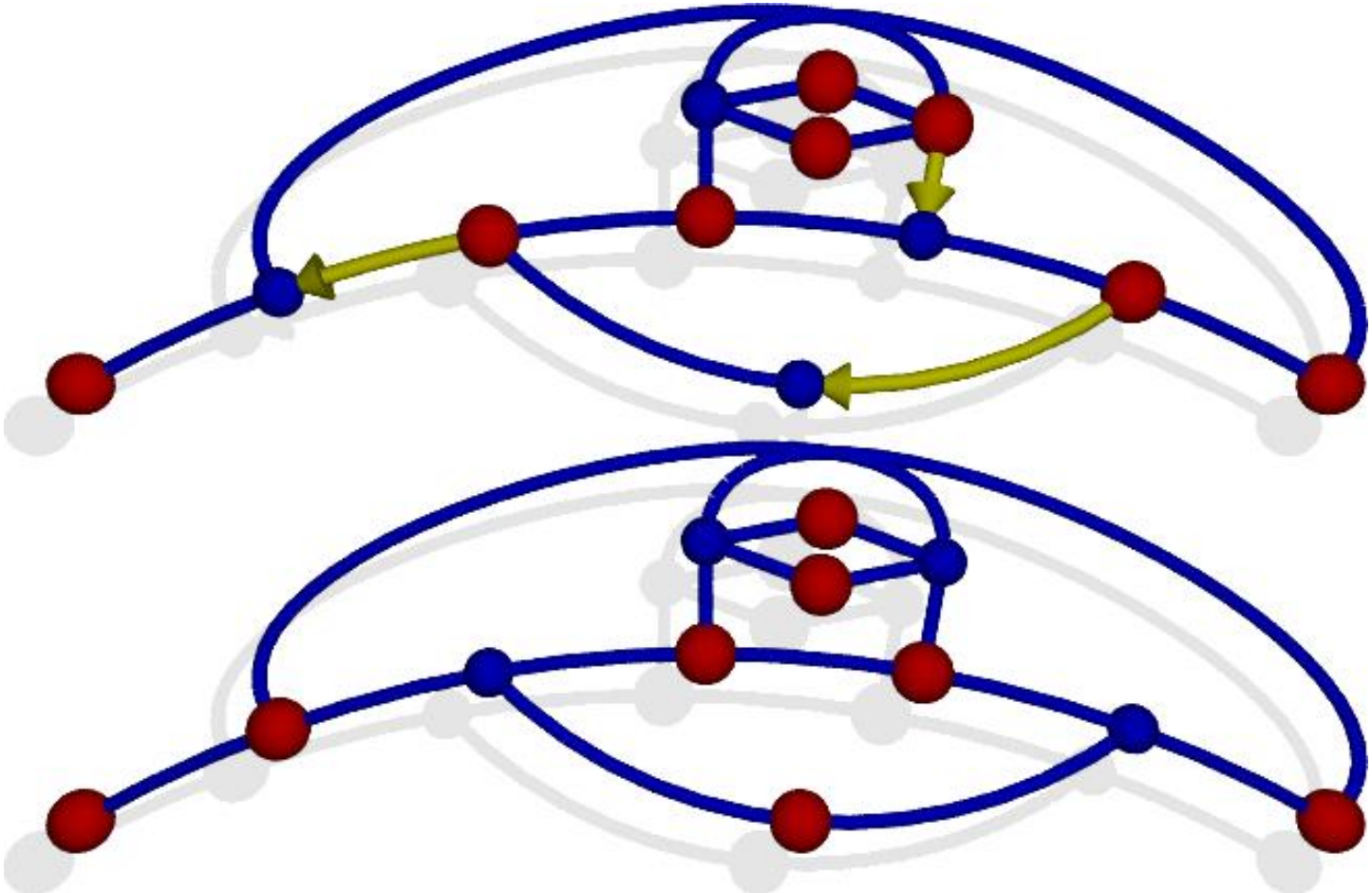
Grad 3 — Keine Brücken (b)



- Gewünscht sind 8 markierte, paarweise verschiedene Knoten
- Rote Knoten sind paarweise verschieden
- Gelbe Knoten sind paarweise verschieden
- $\Rightarrow$ Es nur geht schief, wenn alle orangen Knoten mit roten Knoten identisch sind
- Es gibt genau eine Möglichkeit, daß dies passiert

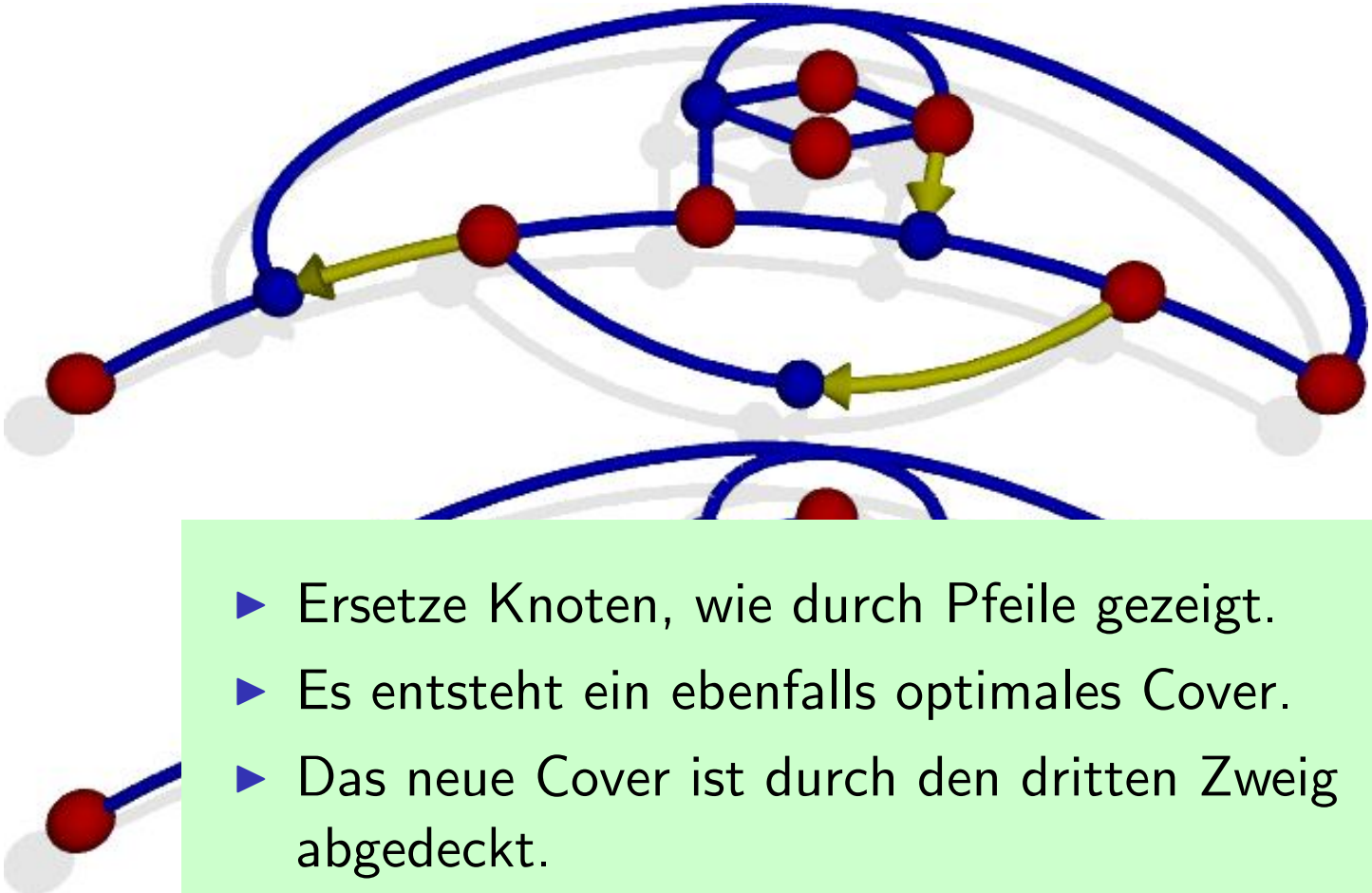
Grad 3 — Keine Brücken (b)

So sieht es aus, wenn das Unglück passiert:



Grad 3 — Keine Brücken (b)

So sieht es aus, wenn das Unglück passiert:



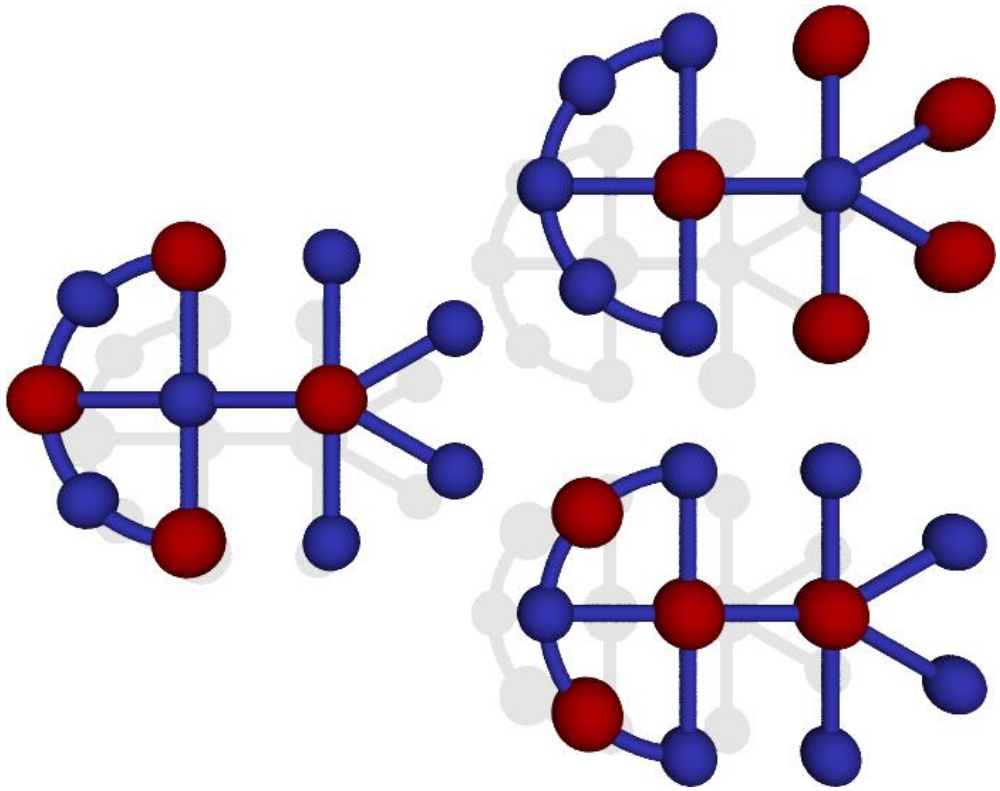
- Ersetze Knoten, wie durch Pfeile gezeigt.
- Es entsteht ein ebenfalls optimales Cover.
- Das neue Cover ist durch den dritten Zweig abgedeckt.
- $\Rightarrow$ Branching-Vektor $(3, 5, 7)$

Grad 4

- Alle Knoten haben Grad zwischen 4 und 6
- Es gibt mindestens einen Knoten mit Grad 4
- Der Graph ist nicht regulär
- Der Graph ist zusammenhängend

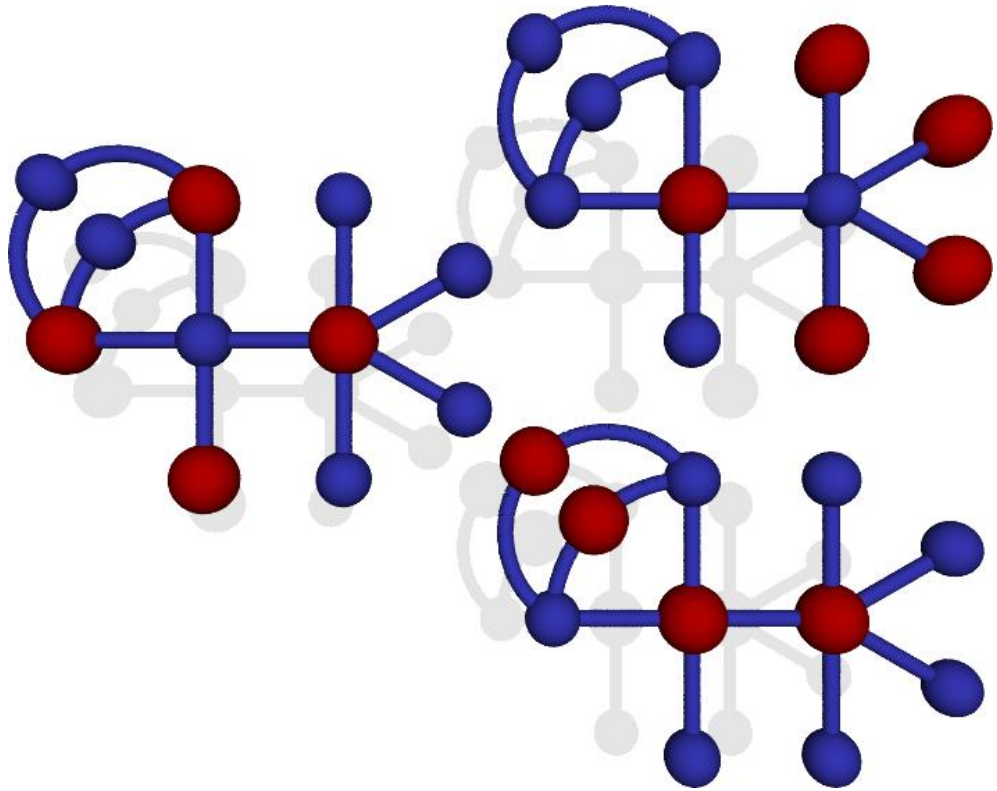
- Es gibt einen Knoten x mit Grad 4, der einen Nachbarn y mit Grad ≥ 5 hat
- 1. Unterfall: x hat ein Dreieck (weggelassen)
- 2. Unterfall: x hat mindestens 2 Brücken, auf denen y nicht liegt
- 3. Unterfall: x hat höchstens 1 Brücke, auf der y nicht liegt

Grad 4 — 2 Brücken



- Es gibt einen Knoten x mit Grad 4, der einen Nachbarn y mit Grad 5 hat
- Hier: x hat zwei Brücken, die nicht über y gehen

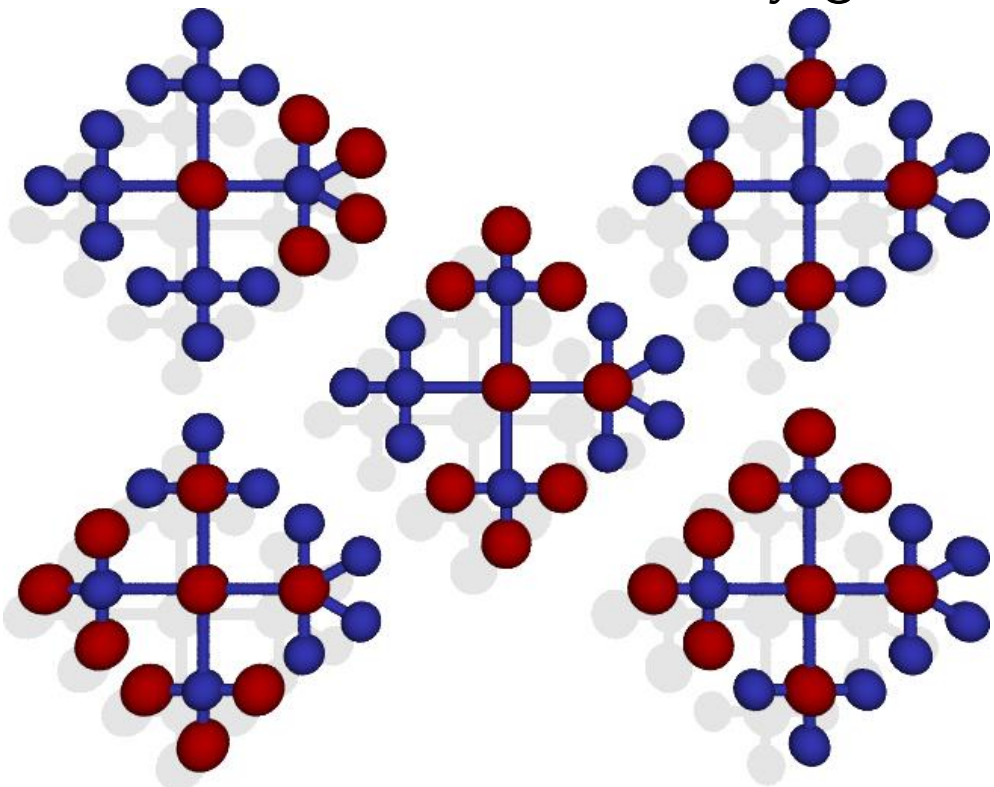
Grad 4 — Doppelbrücke



- Es gibt einen Knoten x mit Grad 4, der einen Nachbarn y mit Grad 5 hat
- Hier: x hat zwei Brücken, die nicht über y gehen, und die auch eine Doppelbrücke sein dürfen

Grad 4 — keine zwei Brücken

Letzter Fall: Es gibt einen Knoten x mit Grad 4, der einen Nachbarn y mit Grad 5 hat.
x hat höchstens eine Brücke, die nicht über y geht



Der Branching-Vektor ist mindestens $(4, 5, 8, 8, 9)$

Zusammenfassung

Es kamen vor:

Verzweigungsvektor	Verzweigungszahl
$(1, 6)$	1.286
$(3, 3)$	1.260
$(3, 4, 7)$	1.289
$(3, 7, 7, 7)$	1.283
$(3, 5, 8, 8)$	1.292
$(3, 5, 7)$	—
$(4, 4, 5)$	1.291
$(4, 5, 8, 8, 9)$	1.291

Suchbaumgröße $O(1.292^k)$

Problemkernreduktion

Sei L ein parametrisiertes Problem.
Manchmal kann man die Frage $(w, k) \in L$ so beantworten:

- Wenn k sehr groß ist, dann kann man mit brute force vorgehen.
- Wenn k sehr klein ist und w ist kompliziert, dann kann (w, k) keine Lösung sein
- Wenn k sehr klein ist und w ist einfach, dann können wir $(w, k) \in L$ auch einfach lösen.

Problemkernreduktion

Definition

Eine Funktion $f : \Sigma^* \times \mathbf{N} \to \Sigma^* \times \mathbf{N}$ ist eine Problemkernreduktion für ein parametrisiertes Problem L, falls

- $(w, k) \in L$ genau dann, wenn $f(w, k) \in L$,
- es eine Funktion $f' : \mathbf{N} \to \mathbf{N}$ gibt, so daß $k, |w'| \leq f'(k)$, falls $f(w, k) = (w', k')$,
- f in polynomieller Zeit berechenbar ist.

Informell: Eine Reduktion auf eine Instanz, deren Größe durch den Parameter bestimmt ist.

Beispiel Matrix-Dominierung

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Falls es mehr als $2k$ Einsen gibt, die sich paarweise in Zeile und Spalte unterscheiden, dann kann es keine Lösung geben.

Beispiel Matrix-Dominierung

Einige Definitionen:

- Eine Zeile ist groß, falls sie mehr als k Einsen enthält.
- Eine Spalte ist groß, falls sie mehr als k Einsen enthält.
- Eine Eins ist klein, falls sie nicht in einer großen Zeile oder Spalte liegt.

Einige nützliche Eigenschaften, falls es eine Lösung gibt:

- Es gibt höchstens k große Zeilen
- Es gibt höchstens k große Spalten
- Es gibt höchstens $2k^2$ kleine Einsen

Beispiel Matrix-Dominierung

Einige Definitionen:

- Eine Zeile ist groß, falls sie mehr als k Einsen enthält.
- Eine Spalte ist groß, falls sie mehr als k Einsen enthält.
- Eine Eins ist klein, falls sie nicht in einer großen Zeile oder Spalte liegt.

Einige nützliche Eigenschaften, falls es eine Lösung gibt:

- Es gibt höchstens k große Zeilen
- Es gibt höchstens k große Spalten
- Es gibt höchstens $2k^2$ kleine Einsen

Beispiel Matrix-Dominierung

Einige Definitionen:

- Eine Zeile ist groß, falls sie mehr als k Einsen enthält.
- Eine Spalte ist groß, falls sie mehr als k Einsen enthält.
- Eine Eins ist klein, falls sie nicht in einer großen Zeile oder Spalte liegt.

Einige nützliche Eigenschaften, falls es eine Lösung gibt:

- Es gibt höchstens k große Zeilen
- Es gibt höchstens k große Spalten
- Es gibt höchstens $2k^2$ kleine Einsen

Beispiel Matrix-Dominierung

Einige Definitionen:

- Eine Zeile ist groß, falls sie mehr als k Einsen enthält.
- Eine Spalte ist groß, falls sie mehr als k Einsen enthält.
- Eine Eins ist klein, falls sie nicht in einer großen Zeile oder Spalte liegt.

Einige nützliche Eigenschaften, falls es eine Lösung gibt:

- Es gibt höchstens k große Zeilen
- Es gibt höchstens k große Spalten
- Es gibt höchstens $2k^2$ kleine Einsen

Beispiel Matrix-Dominierung

Um einen Problemkern zu erhalten, wiederhole dies:

- Enthält eine Zeile nur Nullen, lösche sie
- Enthält eine Spalte nur Nullen, lösche sie
- Gibt es eine Eins, die nicht gleichzeitig in einer großen Zeile und einer großen Spalte liegt und keine kleine Eins dominiert, dann verwandele sie in eine Null.

Eine Ja-Instanz bleibt hierbei eine Ja-Instanz!
Die Größe der Matrix ist jetzt durch eine Funktion von k beschränkt.

Beispiel Matrix-Dominierung

Um einen Problemkern zu erhalten, wiederhole dies:

- Enthält eine Zeile nur Nullen, lösche sie
- Enthält eine Spalte nur Nullen, lösche sie
- Gibt es eine Eins, die nicht gleichzeitig in einer großen Zeile und einer großen Spalte liegt und keine kleine Eins dominiert, dann verwandele sie in eine Null.

Eine Ja-Instanz bleibt hierbei eine Ja-Instanz!

Die Größe der Matrix ist jetzt durch eine Funktion von k beschränkt.

Beispiel Matrix-Dominierung

Um einen Problemkern zu erhalten, wiederhole dies:

- Enthält eine Zeile nur Nullen, lösche sie
- Enthält eine Spalte nur Nullen, lösche sie
- Gibt es eine Eins, die nicht gleichzeitig in einer großen Zeile und einer großen Spalte liegt und keine kleine Eins dominiert, dann verwandele sie in eine Null.

Eine Ja-Instanz bleibt hierbei eine Ja-Instanz!
Die Größe der Matrix ist jetzt durch eine Funktion von k beschränkt.

Beispiel Vertex Cover

Nehmen wir an, ein Graph habe ein Vertex Cover der Größe k.
Sei v ein Knoten, dessen Grad mindestens $k + 1$ ist.

Frage:
Muß v zum Vertex Cover der Größe k gehören?

Resultierende Problemkernreduktion:
Falls ein Knoten mit Grad $> k$ existiert, lösche ihn. Der ursprüngliche Graph hat ein VC der Größe k, genau dann, wenn der reduzierte Graph ein VC der Größe $k - 1$ hat.

Beispiel Vertex Cover

Frage:
Wie groß kann der Graph nach der Problemkernreduktion noch sein?
(falls wir isolierte Knoten auch entfernt haben)

Antwort:

- Das Vertex Cover selbst besteht aus k Knoten.
- Jeder dieser Knoten kann noch k adjazente andere Knoten besitzen.
- Insgesamt gibt es also höchstens $k(k+1)$ Knoten.

Ein kleinerer Problemkern

Theorem (Nemhauser und Trotter)

Sei $G = (V, E)$ ein Graph mit n Knoten und m Kanten.
Dann lassen sich in polynomieller Zeit zwei disjunkte Knotenmengen C_0 und V_0 berechnen mit:

1. *Ist $D \subseteq V_0$ ein Vertex Cover von $G[V_0]$, dann ist $D \cup C_0$ ein Vertex Cover von G.*
2. *Es gibt ein optimales Vertex Cover von G, das C_0 enthält.*
3. *Jedes Vertex Cover von $G[V_0]$ hat mindestens Größe $|V_0|/2$.*

Ein kleinerer Problemkern

Theorem (Nemhauser und Trotter)

Sei $G = (V, E)$ ein Graph mit n Knoten und m Kanten.
Dann lassen sich in polynomieller Zeit zwei disjunkte Knotenmengen C_0 und V_0 berechnen mit:

1. *Ist $D \subseteq V_0$ ein Vertex Cover von $G[V_0]$, dann ist $D \cup C_0$ ein Vertex Cover von G.*
2. *Es gibt ein optimales Vertex Cover von G, das C_0 enthält.*
3. *Jedes Vertex Cover von $G[V_0]$ hat mindestens Größe $|V_0|/2$.*

Falls G ein Vertex Cover der Größe k hat, dann ist $|V_0| + |C_0| \leq 2k$
Warum????

Ein kleinerer Problemkern

Theorem (Nemhauser und Trotter)

Sei $G = (V, E)$ ein Graph mit n Knoten und m Kanten.
Dann lassen sich in polynomieller Zeit zwei disjunkte Knotenmengen C_0 und V_0 berechnen mit:

1. *Ist $D \subseteq V_0$ ein Vertex Cover von $G[V_0]$, dann ist $D \cup C_0$ ein Vertex Cover von G.*
2. *Es gibt ein optimales Vertex Cover von G, das C_0 enthält.*
3. *Jedes Vertex Cover von $G[V_0]$ hat mindestens Größe $|V_0|/2$.*

Ein optimales Vertex Cover von $G[V_0]$ zusammen mit C_0 ist ein optimales Vertex Cover von G.
Warum????

Warum????

Ein kleinerer Problemkern

Es resultiert folgender Algorithmus, der (G, k) auf $(G[V_0], k')$ reduziert.

- Berechne C_0 und V_0
- Setze $k' = k - |C_0|$
- G hat jetzt ein Vertex Cover der Größe k, genau dann wenn $G[V_0]$ ein Vertex Cover der Größe k' hat.

Falls $2k' < |V_0|$, dann kann G kein Vertex Cover der Größe k haben.

Ein kleinerer Problemkern

Folgender Algorithmus löst Vertex Cover:

1. Berechne V_0 und C_0
2. Sage Nein, falls $2(k - |C_0|) < |V_0|$
3. Berechne ein optimales Vertex Cover C_1 von $G[V_0]$
4. Falls $|C_1| + |C_0| \leq k$ sage Ja, sonst Nein

Laufzeit: $n^{O(1)} + O(k2^k)$

Beweis des Theorems von Nemhauser und Trotter

Ein Algorithmus, der C_0 und V_0 berechnet:

Gegeben ist $G = (V, E)$. Sei V' eine disjunkte Kopie von V und sei $G_B = (V, V', E_B)$ der bipartite Teilgraph mit

$$\{x, y'\} \in E_B \iff \{x, y\} \in E.$$

- Berechne ein optimales Vertex Cover C_B für G_B.
- Setze $C_0 = \{ x \mid x \in C_B \text{ und } x' \in C_B \}$.
- Setze $V_0 = \{ x \mid \text{ entweder } x \in C_B \text{ oder } x' \in C_B \}$.

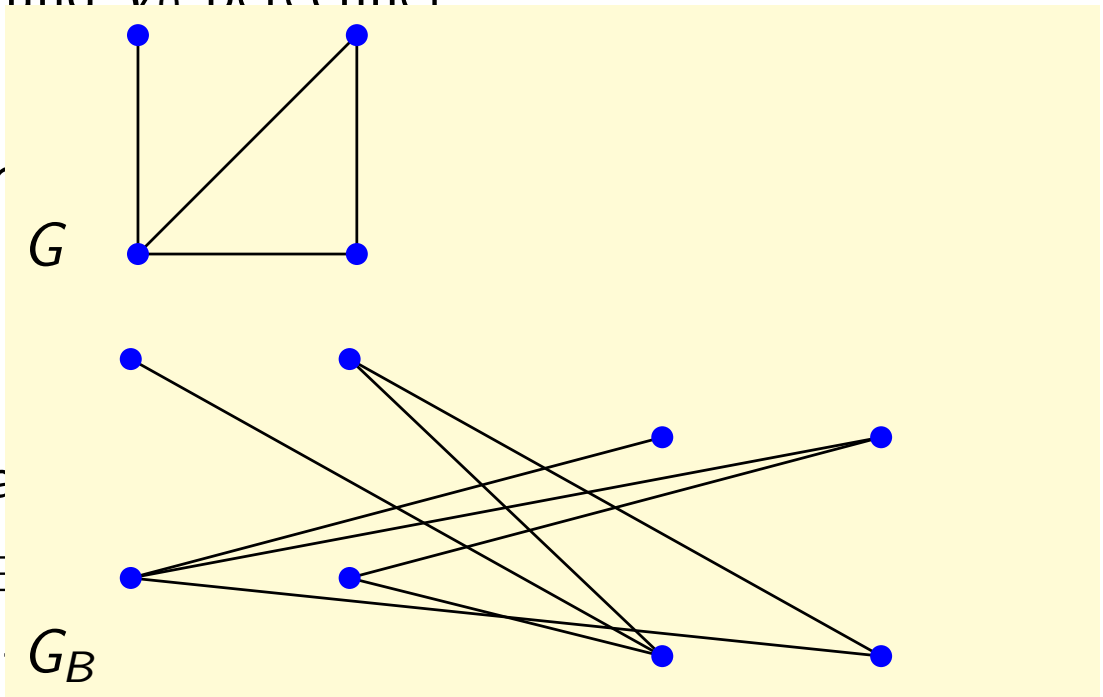
Beweis des Theorems von Nemhauser und Trotter

Ein Algorithmus, der C_0 und V_0 berechnet:

Gegeben ist $G = (V, E)$.
sei $G_B = (V, V', E_B)$ der

$\{x, y'\}$

- Berechne ein optima
- Setze $C_0 = \{ x \mid x \in$
- Setze $V_0 = \{ x \mid$ en

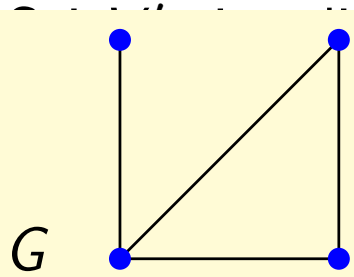


Beweis des Theorems von Nemhauser und Trotter

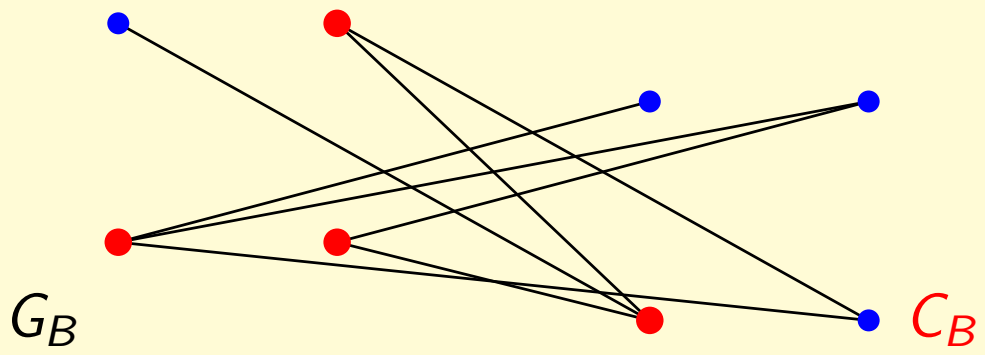
Ein Algorithmus, der C_0 und V_0 berechnet:

Gegeben ist $G = (V, E)$. [illegible]
sei $G_B = (V, V', E_B)$ der

$\{x, y'\}$



- Berechne ein optima
- Setze $C_0 = \{ x \mid x \in$
- Setze $V_0 = \{ x \mid$ en



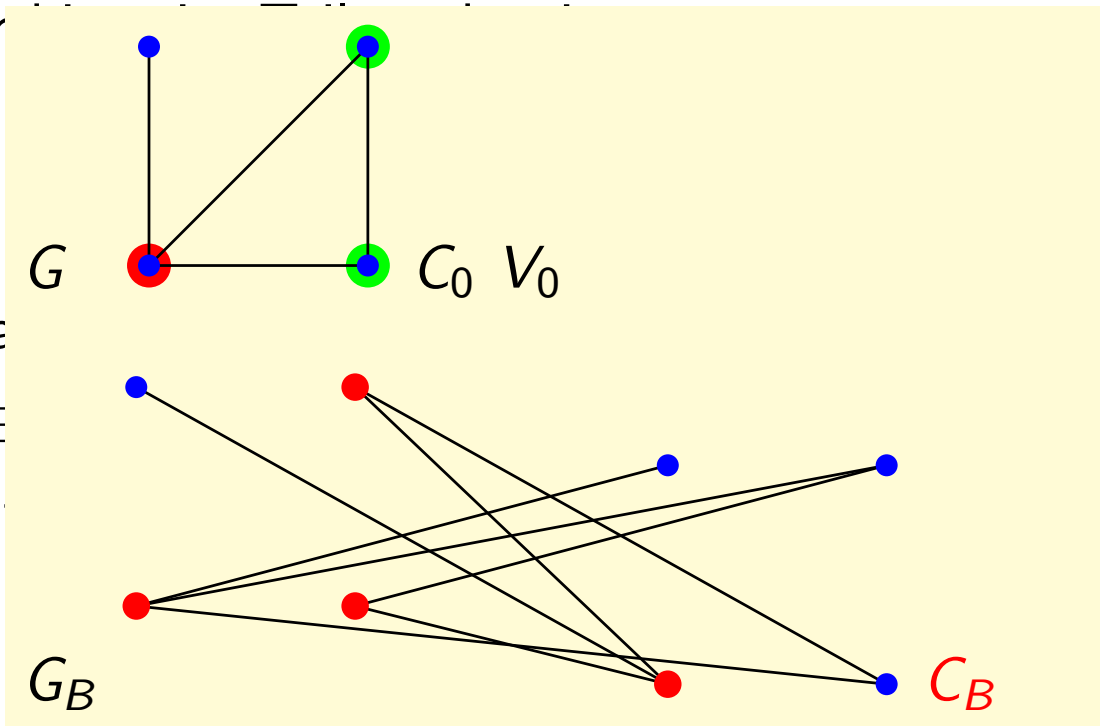
Beweis des Theorems von Nemhauser und Trotter

Ein Algorithmus, der C_0 und V_0 berechnet:

Gegeben ist $G = (V, E)$. Sei V' eine disjunkte Kopie von V und sei $G_B = (V, V', E_B)$ der

$$\{x, y'\}$$

- Berechne ein optima
- Setze $C_0 = \{ x \mid x \in$
- Setze $V_0 = \{ x \mid$ en



Beweis des Theorems von Nemhauser und Trotter

Folgendes ist offensichtlich:

- C_0 und V_0 sind disjunkt
- C_0 und V_0 in polynomieller Zeit berechenbar

Wir müssen die drei Aussagen des Theorems beweisen:

1. Ist $D \subseteq V_0$ ein Vertex Cover von $G[V_0]$, dann ist $D \cup C_0$ ein Vertex Cover von G.
2. Es gibt ein optimales Vertex Cover, das C_0 enthält.
3. Jedes VC von $G[V_0]$ hat mindestens Größe $|V_0|/2$.

Beweis von Aussage 1

Behauptung: Ist $D \subseteq V_0$ ein Vertex Cover von $G[V_0]$, dann ist $D \cup C_0$ ein Vertex Cover von G.

Sei $D \subseteq V_0$ ein Vertex Cover von $G[V_0]$ und $e = \{x, y\} \in E$ eine beliebige Kante.
Sei $I_0 = V - V_0 - C_0$.

- Ist ein Endpunkt von e in C_0, dann okay
- Sind beide Endpunkte in V_0, dann okay
- $x \in I_0 \;\Rightarrow\; y, y' \in C_B \;\Rightarrow\; y \in C_0$, ... okay

Beweis von Aussage 2

Behauptung: Es gibt ein optimales Vertex Cover, das C_0 enthält.
Sei S ein optimales Vertex Cover und $S_V = S \cap V_0$, $S_C = S \cap C_0$, $S_I = S \cap I_0$, $\bar{S}_I = I_0 - S_I$.
Lemma
$(V - \bar{S}_I) \cup S'_C$ ist ein Vertex Cover von G_B.
Beweis
Sei $\{x, y'\} \in E_B$.
Falls $x \notin \bar{S}_I$, dann $x \in (V - \bar{S}_I) \cup S'_C$.
Falls $x \in \bar{S}_I$, dann $x \in I_0$, $x \notin S \Rightarrow y \in S$, $y, y' \in C_B \Rightarrow$
$\Rightarrow y \in C_0 \Rightarrow y \in S \cap C_0 = S_C \Rightarrow y' \in S'_C$.

Beweis von Aussage 2

$$
\begin{aligned}
|V_0| + 2|C_0| &= |V_0 \cup C_0 \cup C_0'| \\
&= |C_B| \\
&\leq |(V - \bar{S}_I) \cup S_C'| \text{ wegen des Lemmas} \\
&= |V - \bar{S}_I| + |S_C'| \\
&= |V_0 \cup C_0 \cup I_0 - (I_0 - S_I)| + |S_C'| \\
&= |V_0| + |C_0| + |S_I| + |S_C|
\end{aligned}
$$

Jetzt folgt $|C_0| \leq |S_I| + |S_C| = |S| - |S_V|$ und daraus $|C_0 \cup S_V| \leq |S|$.

Beweis von Aussage 3

Behauptung: Jedes Vertex Cover von $G[V_0]$ hat mindestens Größe $|V_0|/2$.

Sei S_0 ein optimales Vertex Cover von $G[V_0]$.
$C_0 \cup C_0' \cup S_0 \cup S_0'$ ist ein Vertex Cover von G_B, denn $C_0 \cup S_0$ ist ein Vertex Cover von G.

$$|V_0| + 2|C_0| = |C_B| \leq |C_0 \cup C_0' \cup S_0 \cup S_0'| = 2|C_0| + 2|S_0|$$

Woraus die Behauptung folgt.

Grapheigenschaften

Definition

Eine Grapheigenschaft Π ist eine Klasse von Graphen, die unter Graphisomorphie abgeschlossen ist.

Sind zwei Graphen G_1 und G_2 isomorph, so gehören sie also beide zu Π oder beide nicht.

Grapheigenschaften

Beispiel

- Zusammenhängende Graphen
- Bäume
- Graphen mit einer Clique der Größe 100
- Planare Graphen
- Reguläre Graphen
- Endliche Graphen

Grapheigenschaften

Dies sind keine Grapheigenschaften:

- Graphen, deren Knoten natürliche Zahlen sind
- Jede nichtleere endliche Menge von Graphen
- (Für Logiker: Jede nichtleere Menge von Graphen)

Vererbbare Grapheigenschaften

Eine Grapheigenschaft Π heißt vererbbare Grapheigenschaft, falls gilt:

Sei $G \in \Pi$ und H eine induzierter Untergraph von G.
Dann ist auch $H \in \Pi$.

Mit anderen Worten: Π ist unter Bildung von induzierten Untergraphen abgeschlossen.

Vererbbare Grapheigenschaften

Eine Grapheigenschaft Π heißt vererbbare Grapheigenschaft, falls gilt:

Sei $G \in \Pi$ und H eine induzierter Untergraph von G.
Dann ist auch $H \in \Pi$.

Mit anderen Worten: Π ist unter Bildung von induzierten Unterg

Fragen:

1. Enthält jede vererbbare Grapheigenschaft den leeren Graphen?
2. Sind vererbbare Grapheigenschaften Verbände bezüglich der Relation „induzierter Untergraph“?

Vererbbare Grapheigenschaften

Welche Grapheigenschaften sind vererbbar?

- Bipartite Graphen
- Vollständige Graphen
- Planare Graphen
- Bäume
- Zusammenhängende Graphen
- Graphen mit maximalem Durchmesser d
- Reguläre Graphen

Vererbbare Grapheigenschaften

Welche Grapheigenschaften sind vererbbar?

- Wälder
- Graphen mit einer unabhängigen Menge der Größe 8
- Graphen mit mindestens 17 Knoten
- Graphen, die kein Matching der Größe 35 haben
- 5-reguläre Graphen
- Unendliche Graphen
- Chordale Graphen

Charakterisierung durch Ausschlußmengen

Definition

Eine Grapheigenschaft Π besitzt eine Charakterisierung durch Ausschlußmengen, wenn es eine Grapheigenschaft $\mathcal{F}$ gibt, so daß $G \in \Pi$ genau dann gilt, wenn $\mathcal{F}$ keinen induzierten Untergraph von G enthält.

Charakterisierung durch Ausschlußmengen

Definition

Eine Grapheigenschaft Π besitzt eine Charakterisierung durch Ausschlußmengen, wenn es eine $\mathcal{I}$ daß $G \in \Pi$ genau dann gilt, wenn $\mathcal{I}$ von G enthält.

Frage:
Besitzt jede vererbbare Grapheigenschaft eine Charakterisierung durch Ausschlußmengen?

Charakterisierung durch Ausschlußmengen

Definition

Eine Grapheigenschaft Π besitzt eine Charakterisierung durch Ausschlußmengen, wenn es eine[daß $G \in \Pi$ genau dann gilt, wenn $\mathcal{I}$[von G enthält.

Frage:
Besitzt jede vererbbare Grapheigenschaft eine

Antwort:
Ja. Wähle $\mathcal{F} = \mathcal{G} - \Pi$, wobei $\mathcal{G}$ alle Graphen enthält.

Endliche Ausschlußmengen

Definition

Eine Grapheigenschaft Π besitzt eine endliche Charakterisierung durch Ausschlußmengen, wenn sie eine Charakterisierung durch $\mathcal{F}$ besitzt und $\mathcal{F}$ nur endlich viele nicht-isomorphe Graphen enthält.

Endliche Ausschlußmengen

Welche Grapheigenschaften besitzen eine endliche Charakterisierung durch Ausschlußmengen?

- Graphen mit einer unabhängigen Menge der Größe 7?
- Bipartite Graphen?
- Wälder?
- Planare Graphen?
- 5-färbbare Graphen?
- Graphen mit einem Vertex Cover der Größe k?

Endliche Ausschlußmengen

Welche Grapheigenschaften besitzen eine endliche Charakterisierung durch Ausschlußmengen?

- Dreiecksfreie Graphen?
- Graphen, die keine k-Clique enthalten?
- Graphen, deren Durchmesser höchstens d ist?
- Kreisfreie Graphen?
- Graphen, die keinen Kreis der Länge k enthalten?

Graphenmodifikationsprobleme

Sei Π eine Grapheigenschaft. Es gibt folgende populäre Graphmodifikationsprobleme für eine Eingabe G:

1. Kantenlöschungsproblem: Können wir k Kanten aus G löschen, um einen Graphen aus Π zu erhalten?
2. Knotenlöschungsproblem: Können wir k Knoten aus G löschen, um einen Graphen aus Π zu erhalten?
3. Knoten/Kantenlöschungsproblem: Können wir k Knoten und l Kanten aus G löschen, um einen Graphen aus Π zu erhalten?
4. Kantenadditionsproblem: Können wir k Kanten in G hinzufügen, um einen Graphen aus Π zu erhalten?

Verallgemeinerung

Definition

$\Pi_{i,j,k}$-Graphmodifikationsproblem

Eingabe: Graph $G = (V, E)$
Parameter: $i, j, k \in \mathbf{N}$
Frage: Können wir bis zu i Knoten und j Kanten aus G entfernen und bis zu k Kanten hinzufügen, um einen Graphen aus Π zu erhalten?

Das Leizhen Cai–Theorem

Theorem

Sei Π eine Grapheigenschaft mit einer endlichen Charakterisierung durch Ausschlußmengen.
Dann ist das $\Pi_{i,j,k}$-Graphmodifikationsproblem in $O(N^{i+2j+2k}|G|^{N+1})$ Schritten lösbar und damit fixed paramater tractable.
Dabei ist N die Knotenzahl des größten Graphen in der Ausschlußmenge, also eine Konstante.

Beweis des Leizhen Cai–Theorems

Lemma

Sei Π eine vererbbare Grapheigenschaft, die in $T(G)$ Schritten überprüfbar ist.
Dann kann für jedes $G = (V, E) \notin \Pi$ in $O(|V|T(G))$ Schritten ein minimaler verbotener induzierter Untergraph gefunden werden.

„Minimal“ bezieht sich hierbei auf die Ordnung „induzierter Untergraph“.

Beweis des Leizhen Cai–Theorems

Beweis des Lemmas:

Beweis.
Sei $V = \{v_1, \ldots, v_n\}$.

$H := G$
for $i = 1, \ldots, n$ **do**
  if $H - \{v_i\} \notin \Pi$ **then** $H := H - \{v_i\}$
od

Am Ende ist H ein minimaler verbotener induzierter Untergraph. $\square$

Beweis des Leizhen Cai–Theorems

Eingabe: $G = (V, E)$
Parameter: $i, j, k \in \mathbf{N}$
Frage: $G \in \Pi_{i,j,k}$

while $i + j + k > 0$ **do**
  $H :=$ minimaler verbotener induzierter Untergraph von G
  Modifiziere G indem von H ein Knoten oder eine Kante gelöscht wird oder eine Kante hinzugefügt wird.
  Setze $i := i - 1$, $j := j - 1$ oder $k := k - 1$.
  if $G \in \Pi$ **then** Antwort JA
od
Antwort NEIN

Beweis des Leizhen Cai–Theorems

Laufzeit:

Finden von H: $O(|V| \cdot |V|^N)$ nach Lemma

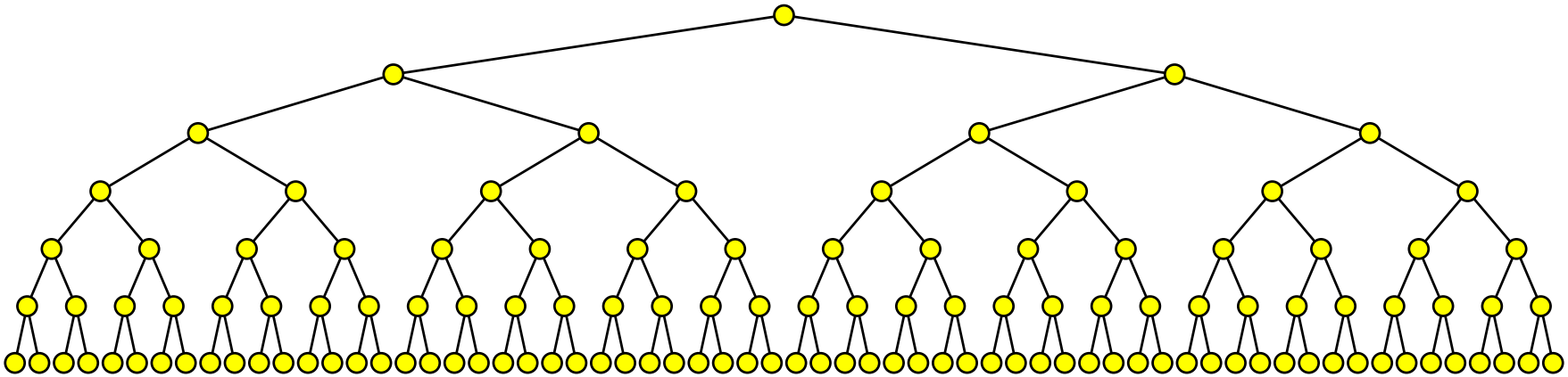
Es gibt nur N Möglichkeiten einen Knoten aus H zu löschen.

Es gibt nur je $\binom{N}{2}$ Möglichkeiten eine Kante zu H hinzuzufügen oder aus H zu löschen.

Gesamtlaufzeit also

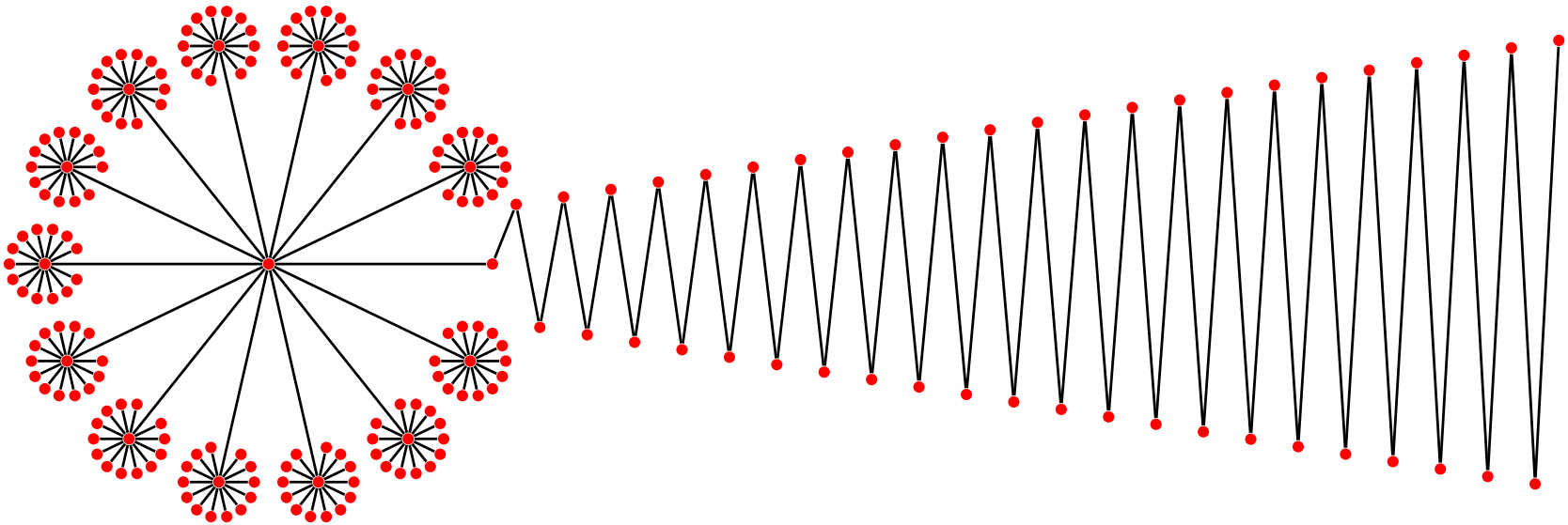
$$O(N^{i+2j+2k}|V|^{N+1}).$$

Interleaving — Suchbäume und Problemkern



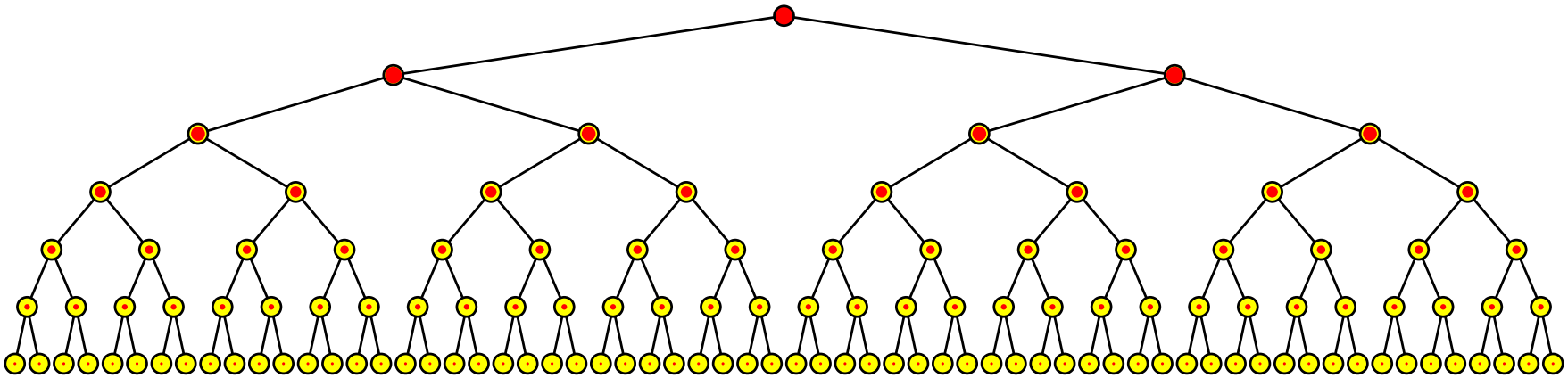
In einem Suchbaum wird der Parameter zu den Blättern hin kleiner, die Größe der Instanz aber nicht unbedingt (sie könnte sogar größer werden).
Falls Expansion eines Suchbaumknotens $p(n)$ Schritte benötigt, dann ist die Gesamtlaufzeit $O(s(k,n)p(n))$, wobei $s(k,n)$ die Größe des Suchbaums ist.

Interleaving



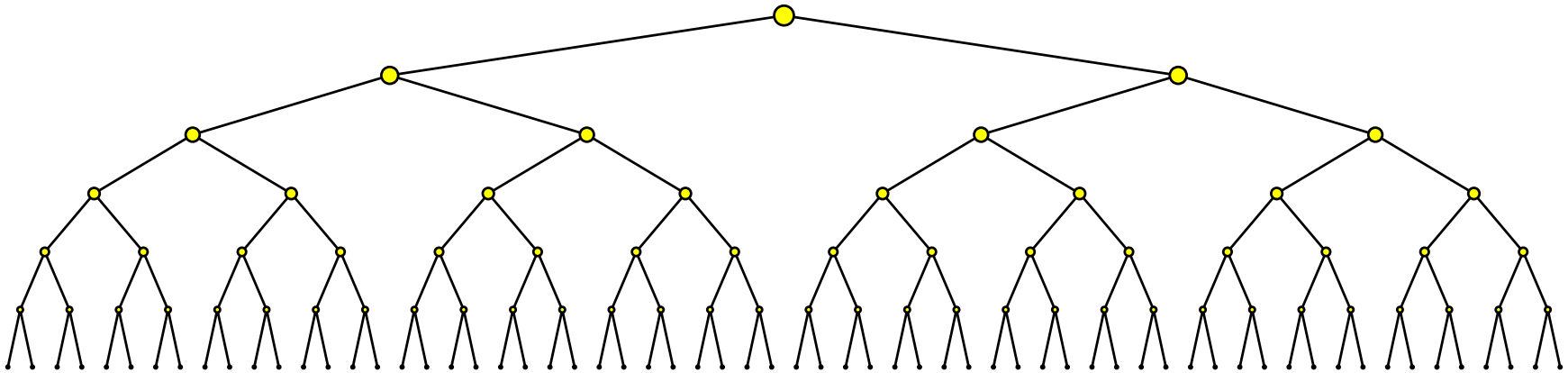
Die Größe dieses Graphen wird nicht kleiner als $n/2$ innerhalb des gesamten Suchbaums eines Vertex Cover–Algorithmus, falls wir nie eine Reduktion auf den Problemkern ausführen.

Interleaving



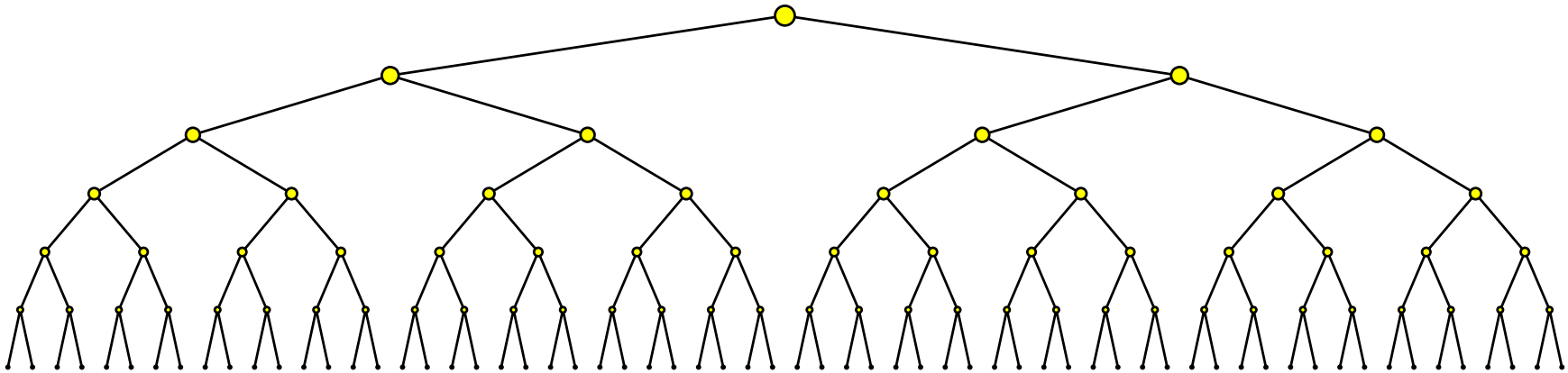
Die Größe des Parameters ist rot gezeigt. Die Rekursion endet, wenn der Parameter klein ist. In den Blättern ist der Parameter durch eine Konstante beschränkt.
Fast alle Knoten sind in der Nähe eines Blatts $\Longrightarrow$ fast alle Knoten haben einen kleinen Parameter.

Interleaving



Falls wir nach jeder Expansion eines Suchbaumknotens eine Reduktion auf den Problemkern anwenden, dann werden die Instanzen zu den Blättern hin kleiner.
Eine genaue Analyse zeigt, daß die Gesamtlaufzeit nur $O(s(n,k)+t(n)+r(n))$ statt $O(s(n,k)t(n))$ beträgt, wobei $r(n)$ die Zeit für die Problemkernreduktion ist.

Suchbäume und dynamisches Programmmieren



Falls alle Knoten nahe den Blättern sind und es gibt sehr viele von ihnen, dann müssen einige identisch sein.

$\Longrightarrow$ Wir können die Laufzeit verbessern, falls wir ihre Lösung nur einmal berechnen und in einer Datenbank speichern.

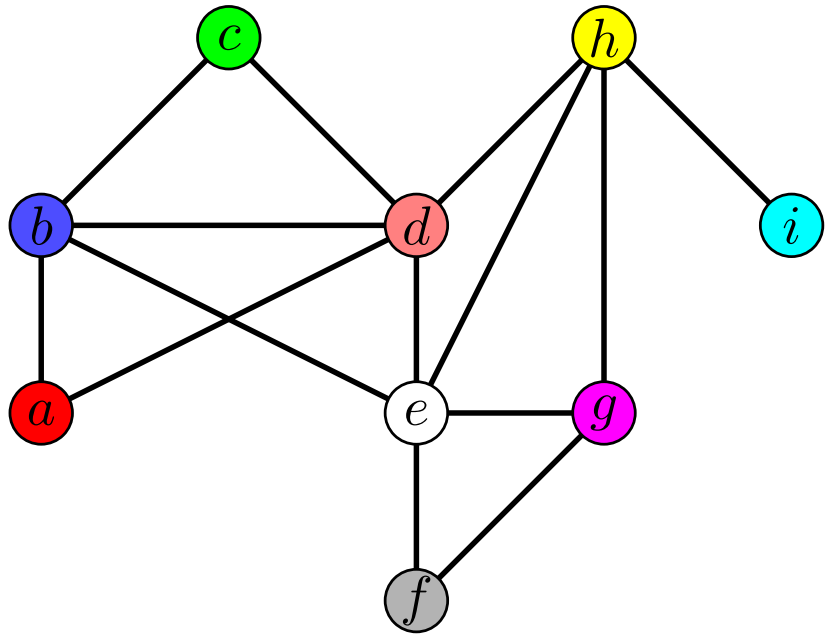
Suchbäume und dynamisches Programmmieren

Beispiel: Vertex Cover mit dem 2^k-Algorithmus

- Jeder Suchbaumknoten ist ein induzierter Untergraph.
- Nach einer Problemkernreduktion ist die Größe eines Graphen höchstens $2k'$, wenn wir eine Lösung der Größe k' suchen.
- Es gibt höchstens $O\left(\binom{2k}{2k'}\right)$ induzierte Untergraphen bis zur Größe $2k'$.
- Die Laufzeit ist $O\left(2^{k-k'}\binom{2k}{2k'}\right)$, falls wir Lösungen der Größe $2k'$ in der Datenbank speichern.
- Wählen wir k' optimal, ist die Laufzeit 1.886^k.

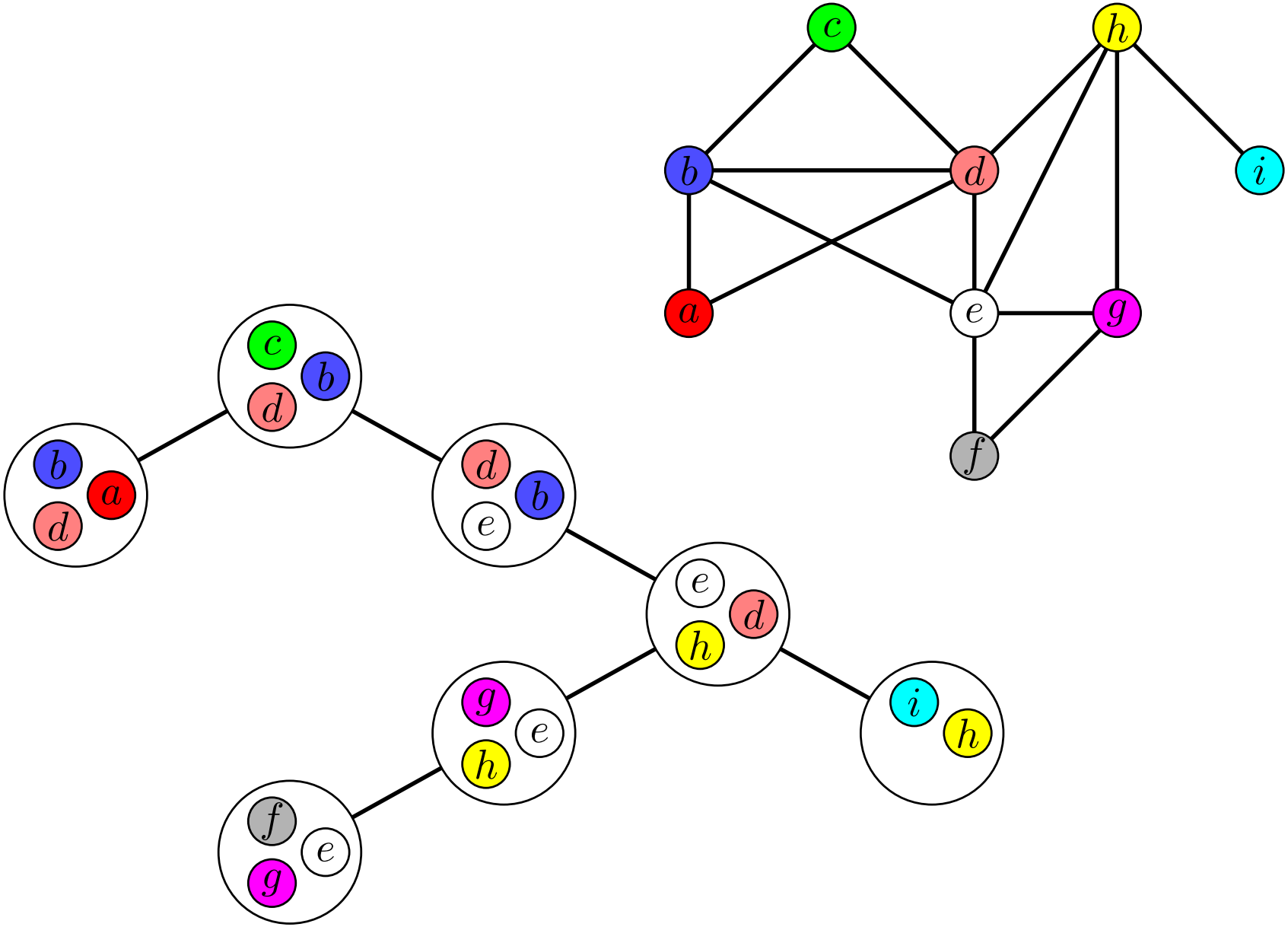
Baumzerlegungen

Eine Baumzerlegung eines Graphen G ist ein Baum, dessen Knoten bags genannt werden. Jeder bag ist eine Menge von Knoten aus G.

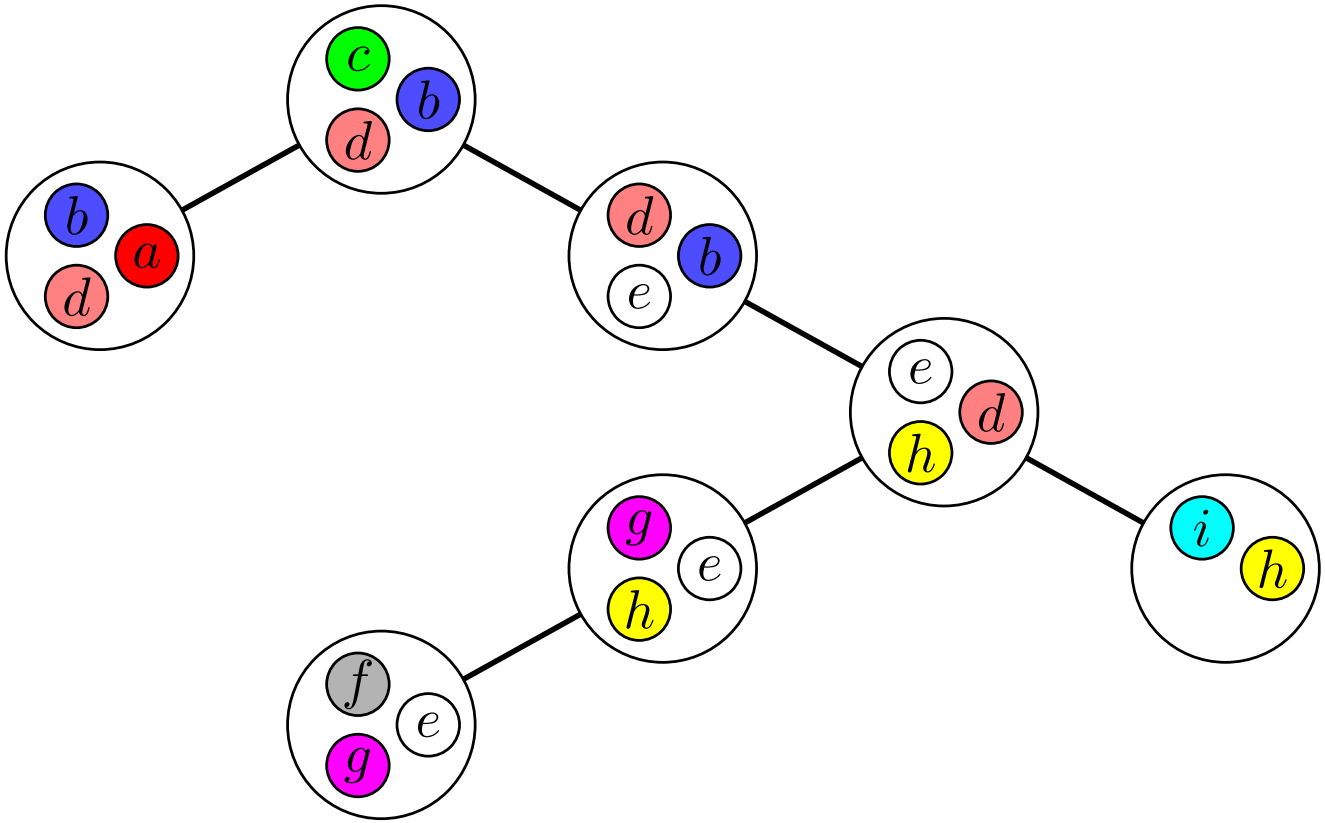


- Jeder Knoten und jede Kante von G muß in einem bag vorkommen.
- Ist ein Knoten in zwei bags, dann auch in allen dazwischen

Baumzerlegungen



Baumzerlegungen und Baumweite

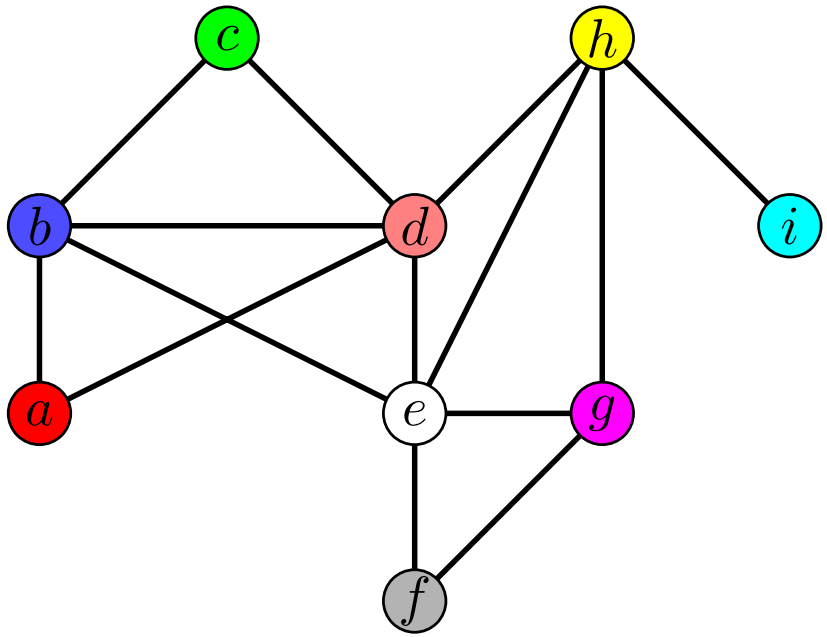


Die Weite einer Baumzerlegung ist die Größe des größten Bags minus 1.

$\Rightarrow$ Hier ist die Baumweite 2.

Baumzerlegungen und Baumweite

Alternative Definition:



Die Baumweite von G ist die minimale Anzahl von Polizisten, die einen Räuber in G fangen können minus 1.

Baumzerlegungen und Baumweite

Ist eine Baumzerlegung von G mit Weite w gegeben, dann können viele Optimierungsprobleme für G in $c^w \cdot poly(n)$ gelöst werden durch dynamisches Programmierung auf der Baumzerlegung.

Wir können also viele Probleme schnell lösen, falls wir eine Baumzerlegung mit kleiner Weite bekommen können.

Allgemeines Ergebnis

Probleme, die alle diese Eigenschaften besitzen, sind fixed-parameter tractable:

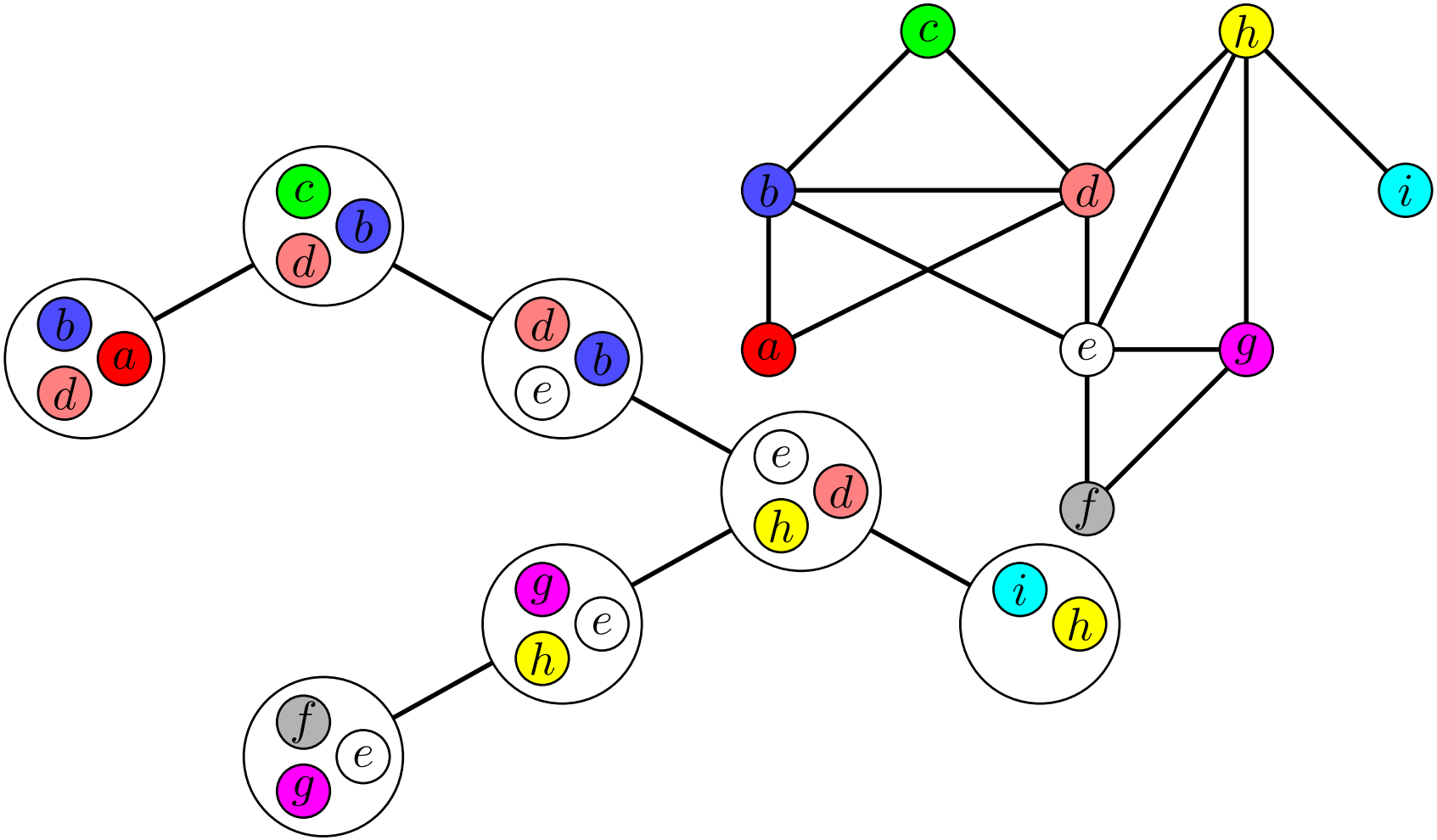
- Gegeben ist ein planarer Graph $G = (V, E)$ und eine Zahl k. Frage: Gibt es ein $S \subseteq V$ der Größe k mit einer bestimmten Eigenschaft (z.B. daß S ein Vertex Cover ist).
- Es gibt eine Konstante c so daß jeder Knoten von G höchstens c von einem Knoten in S entfernt ist.
- Hat man eine Baumzerlegung der Weite w, kann man das Problem in $f(w)n^{O(1)}$ Schritten lösen.

Spezialfälle sind Vertex Cover, Independent Set, und Dominating Set.

Beweisidee

- Da jeder Knoten nur c Schritte von einem Knoten in S entfernt ist, gibt es keinen Pfad länger als $2c|S|$.
- Der Durchmesser ist also $O(k)$, falls es ein S der Größe k gibt.
- Ein planarer Graph mit Durchmesser d hat höchstens Baumweite $3d$ (ohne Beweis).
- Hat der Graph einen Durchmesser größer als $2ck$, können wir Nein sagen.
- Andernfalls bekommen wir eine Baumzerlegung der Weite $6ck$ und können mit ihr die Frage beantworten.

Dynamisches Programmieren auf der Baumzerlegung

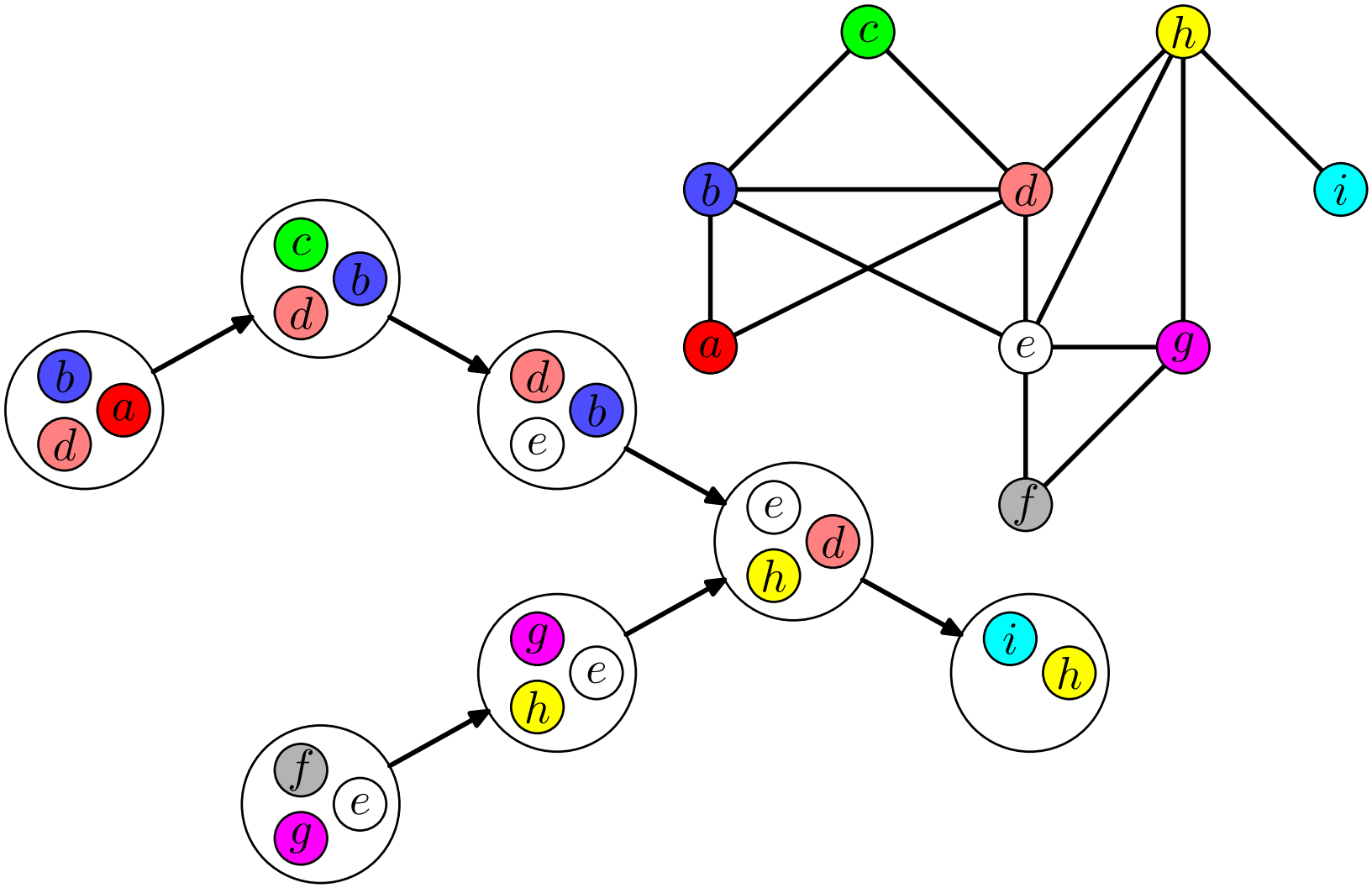


Dynamisches Programmieren auf der Baumzerlegung

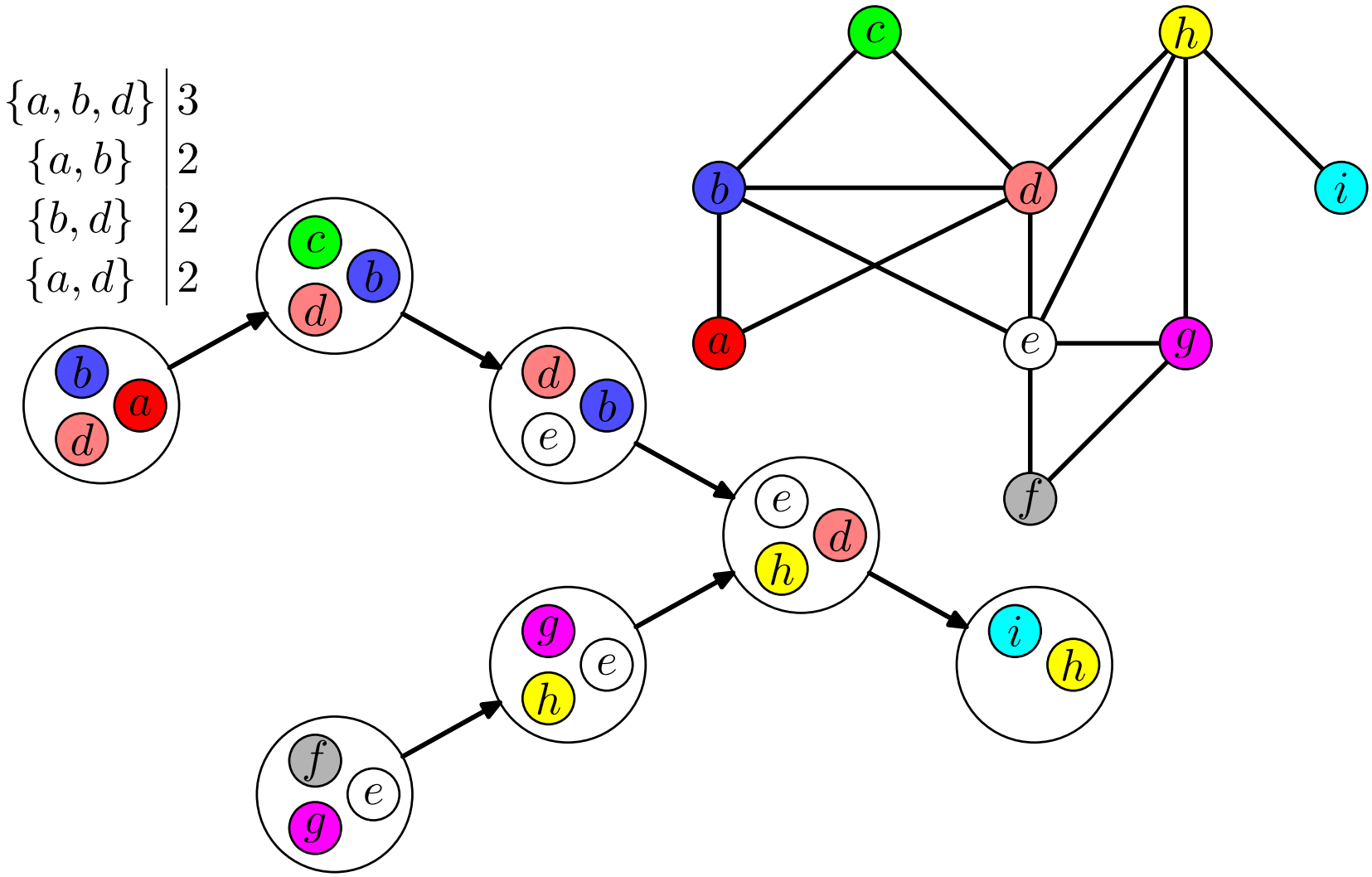
Allgemein gehen wir so vor:

- Die Baumzerlegung wird in einen Wurzelbaum verwandelt: Ein beliebiger Knoten ist die Wurzel. Kinder zeigen auf ihre Eltern.
- Ein bag repräsentiert den Untergraphen, der durch die Knoten all seiner Abkömmlinge induziert wird.
- Zu jedem bag wird eine Tabelle berechnet, aus der die optimale Lösung für seinen Untergraphen hervorgeht.
- Die Tabellen der Kinder werden zuerst berechnet.

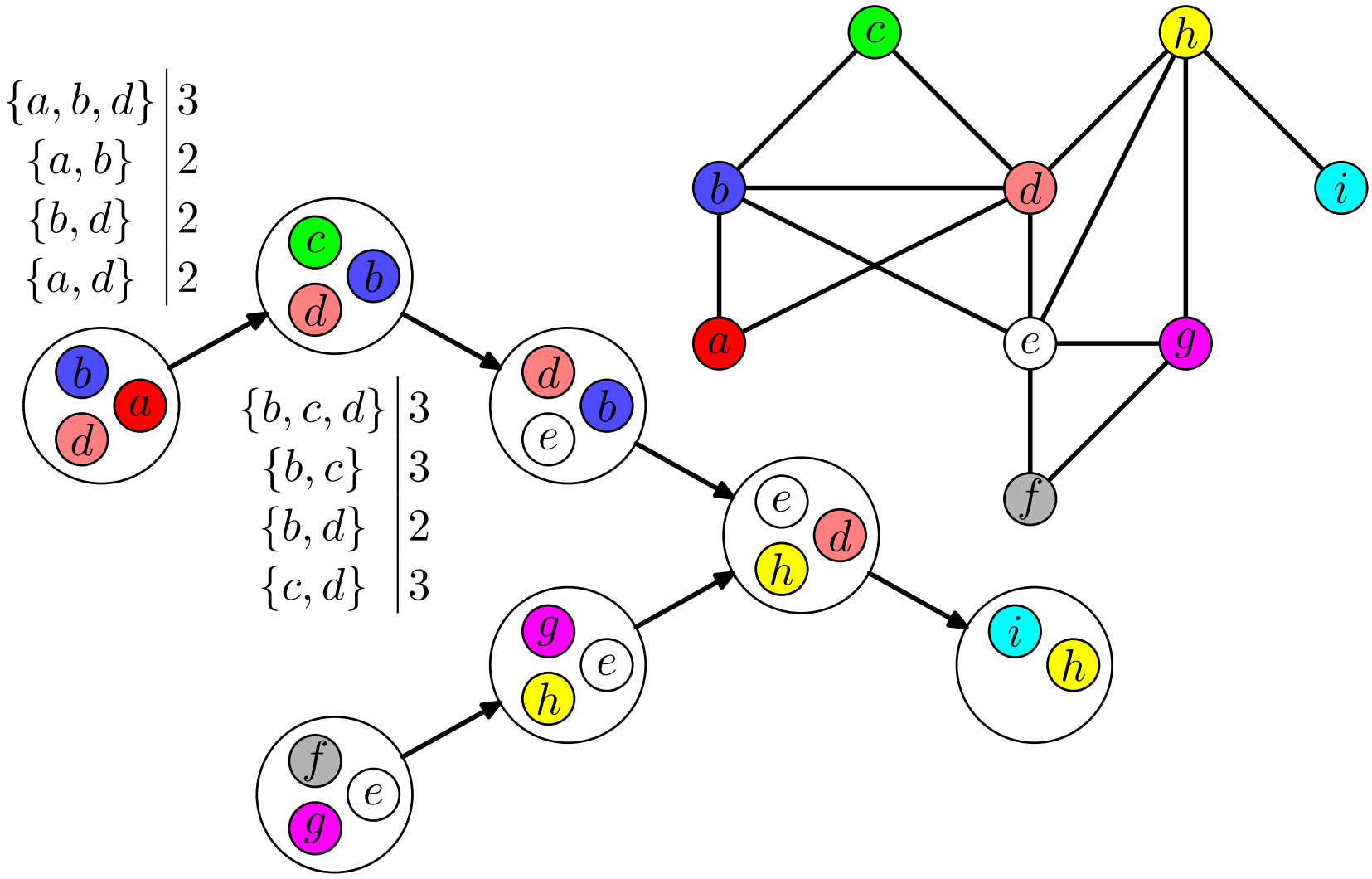
Dynamisches Programmieren auf der Baumzerlegung



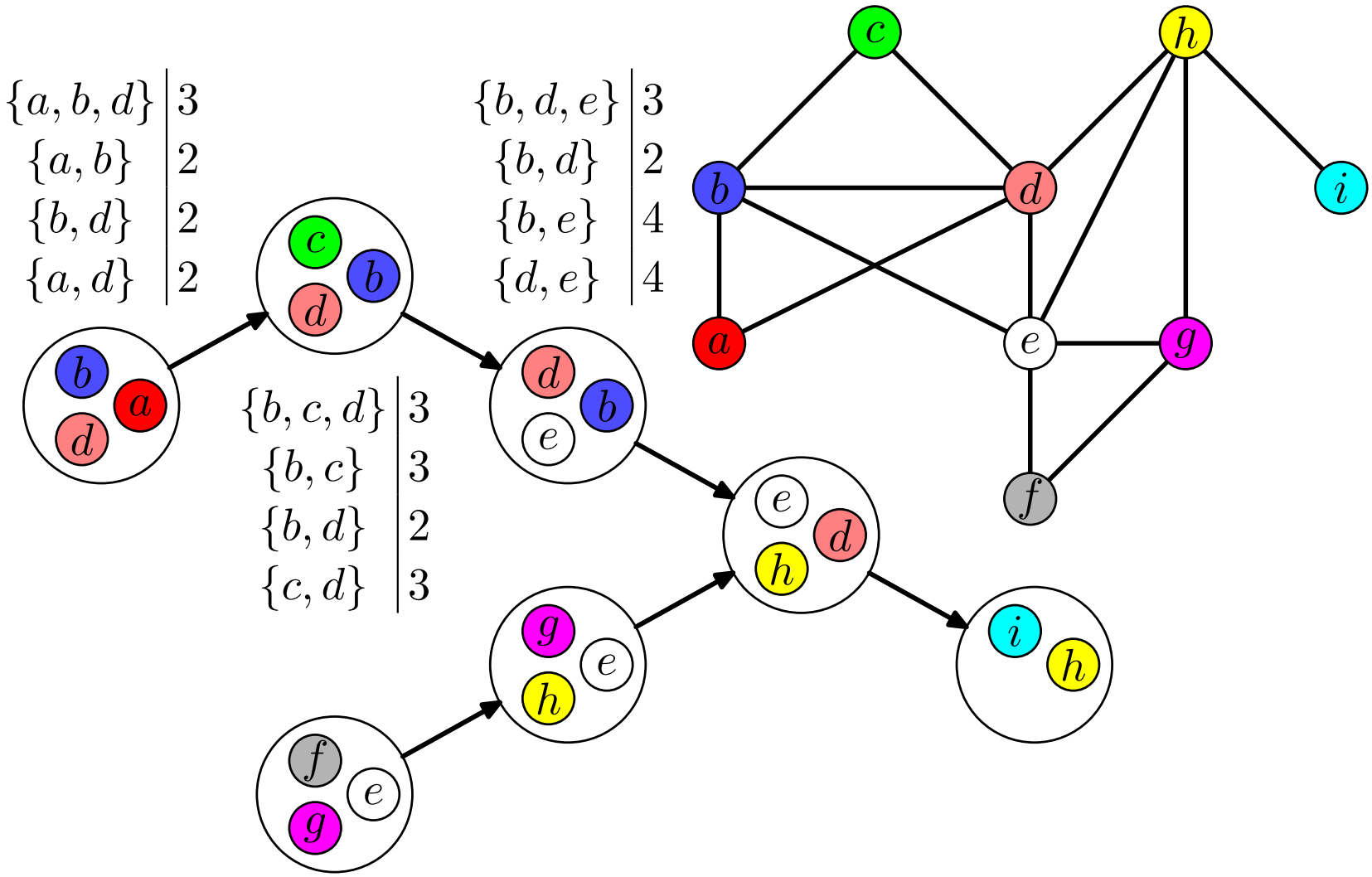
Dynamisches Programmieren auf der Baumzerlegung



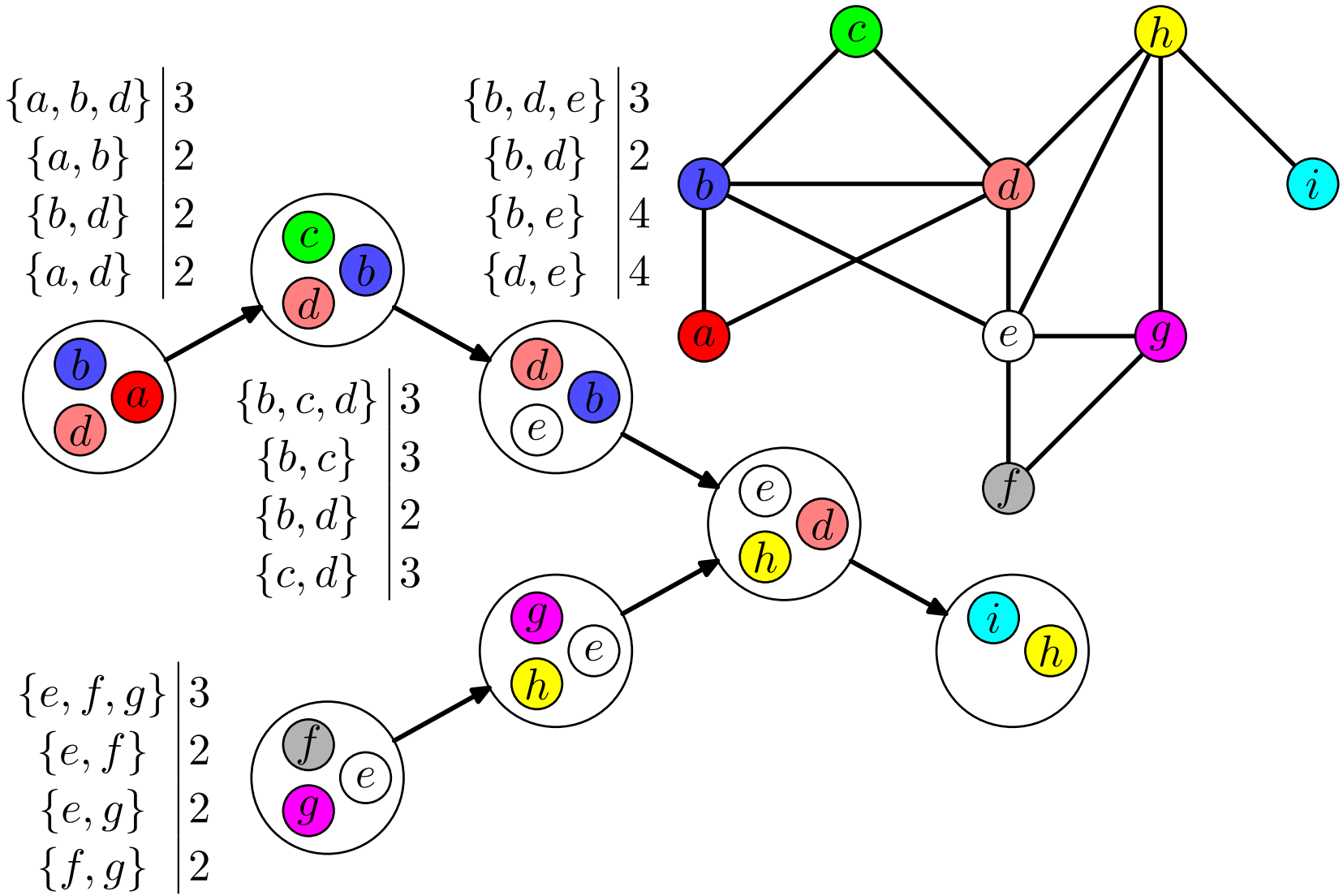
Dynamisches Programmieren auf der Baumzerlegung



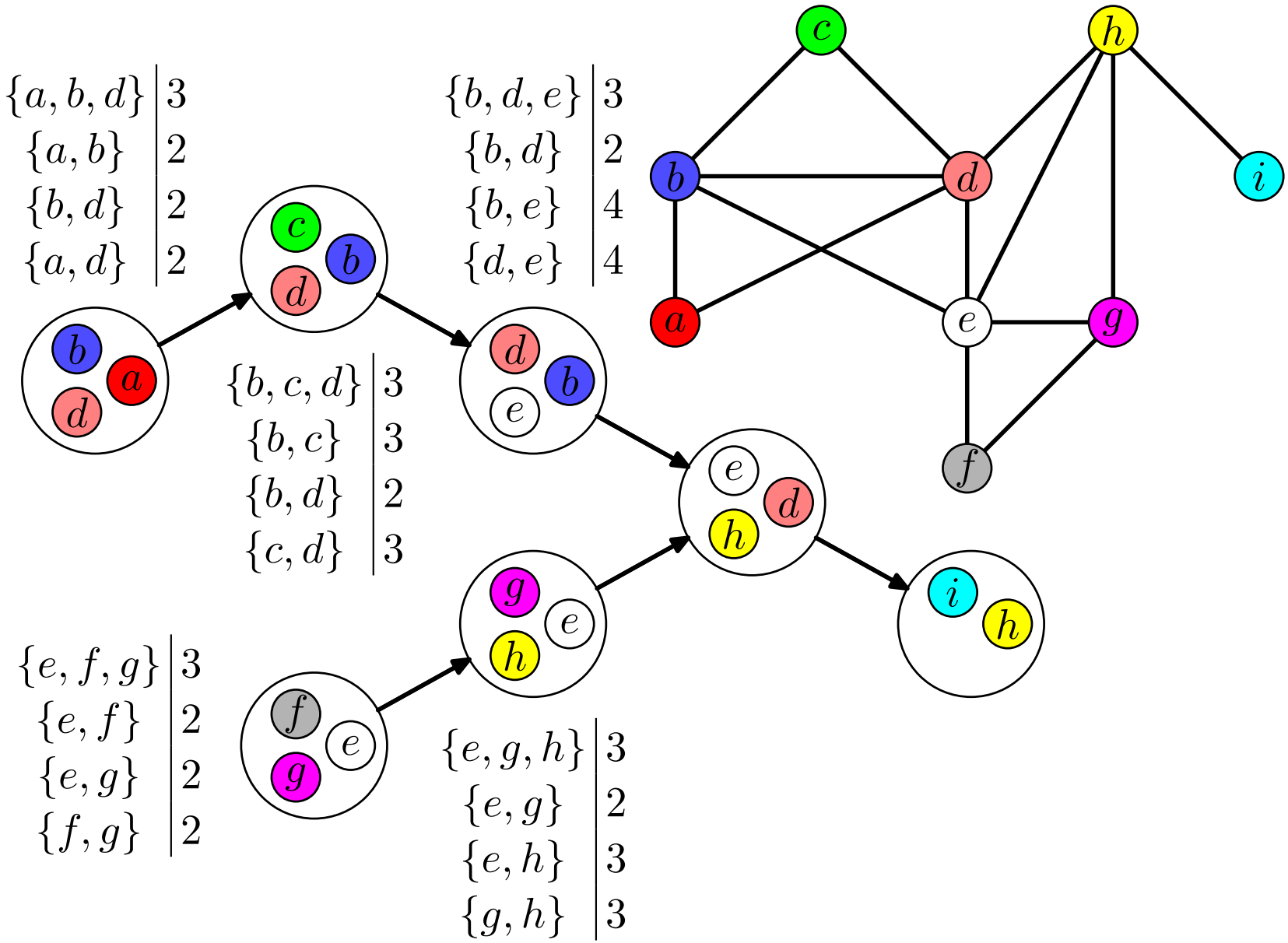
Dynamisches Programmieren auf der Baumzerlegung



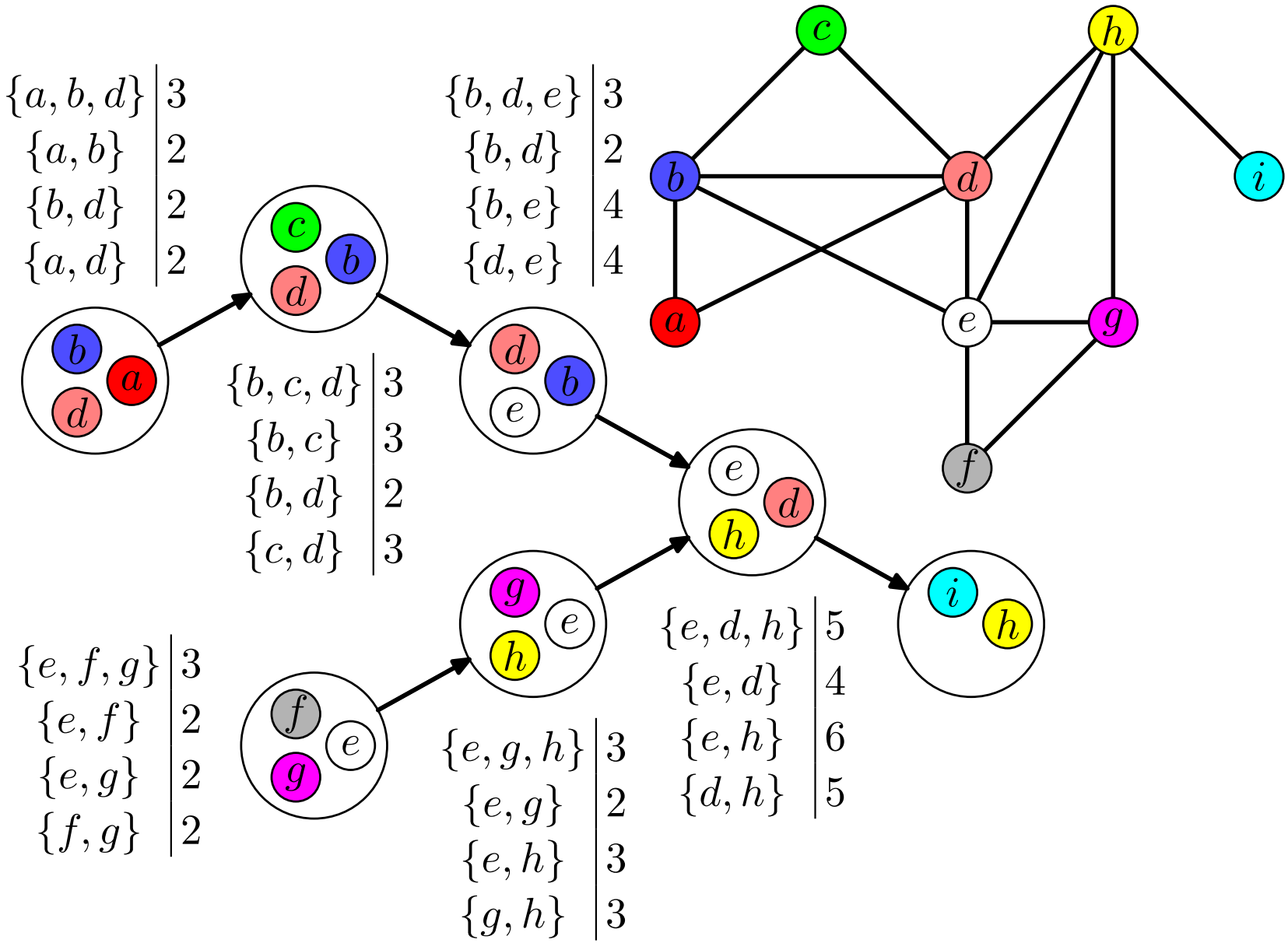
Dynamisches Programmieren auf der Baumzerlegung



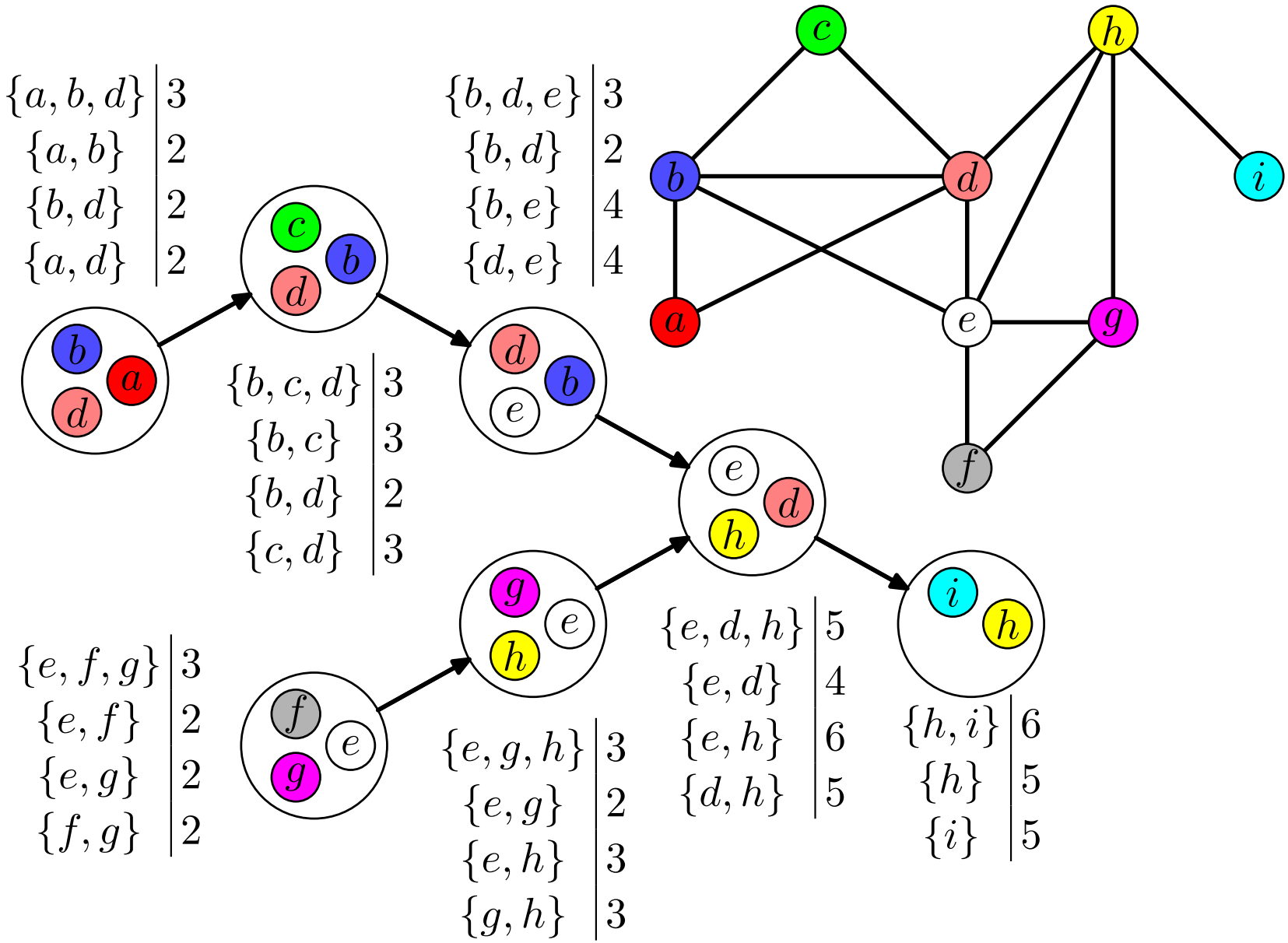
Dynamisches Programmieren auf der Baumzerlegung



Dynamisches Programmieren auf der Baumzerlegung



Dynamisches Programmieren auf der Baumzerlegung



Erweiterte monadische Graphtheorie zweiter Ordnung

Wir führen eine Logik ein, die wir kurz MS_2-Theorie nennen.
In dieser Theorie gibt es Variablen für Knoten, für Kanten, für Mengen von Knoten und für Mengen von Kanten.
Es gibt die Quantoren $\exists$ und $\forall$ und die Verknüpfungen $\wedge$, $\vee$ und $\neg$.
Es gibt außerdem folgende Relationen:

$$u \in U,\ d \in D,\ inc(d, u),\ adj(u, v),\ \cdot = \cdot$$

wobei u, v Knotenvariablen, d eine Kantenvariable, U eine Knotenmengenvariable, D eine Kantenmengenvariable ist.

Erweiterte monadische Graphtheorie zweiter Ordnung

Ein Graph kann eine Formel erfüllen oder nicht. So lassen sich Klassen von Graphen durch Formeln beschreiben.

Beispiel
Welche Graphen erfüllen folgende Formel:

$$\exists u \exists v \exists w (adj(u, v) \wedge adj(u, w) \wedge adj(v, w))$$

Gibt es eine Formel, die bipartite Graphen beschreibt?

Courcelle's Theorem

Theorem

Gegeben ist eine Graphklasse $\mathcal{G}$, die durch eine Formel in MS_2-Theorie beschrieben ist.
Folgendes Problem ist fixed parameter tractable:

Eingabe: Graph G mit Baumweite k
Parameter: k
Frage: Gehört G zu $\mathcal{G}$

Beweis.
schwierig. . . □

Courcelle's Theorem

Theorem

Gegeben ist eine Graphklasse $\mathcal{G}$, die durch eine Formel in MS_2-Theorie beschrieben ist.
Folgendes Problem ist fixed parameter tractable:

Eingabe: Graph G mit Baumweite k
Parameter: k
Frage: Gehört G zu $\mathcal{G}$

Beweis.

schwierig. . . □

Courcelle's Theorem

Wir konnten Vertex-Cover lösen, falls der Parameter die Baumweite ist.

Falls wir Courcelles Theorem verwenden, ist aber die Größe der Formel von der Größe des gesuchten Vertex-Covers abhängig:

$$\exists v_1 \ldots \exists v_k \forall e (inc(e, v_1) \vee \cdots \vee inc(e, v_k)$$

Warum ist das ein Problem?

Courcelle's Theorem (Erweiterungen)

Erweiterung der MS_2-Logik:

Wir erlauben folgenden neuen Quantor:

$$\min U \; \forall e \exists u (u \in U \wedge inc(e, u))$$

Ist die Baumweite beschränkt, so kann eine minimale Menge von Knoten U in polynomieller Zeit berechnet werden, für die eine MS_2 Formel $F(U)$ erfüllt ist.

Courcelle's Theorem (Erweiterungen)

Gegeben sei ein Graph G mit Knotenbeschriftungen aus der Menge $\{1, \ldots, c\}$. Die entsprechenden Mengen von Knoten sind $V_1, \ldots, V_c\}$.

Wir dürfen Enthaltensein in V_i ausdrücken.

Beispiel

$$\max U \subseteq V_1 \forall x \in V_2 \exists y (adj(x, y) \vee y \notin U)$$

(Gibt es eine Menge U von roten Knoten, so daß jeder blaue Knoten einen Nachbarn hat, der nicht in U ist.)
Dieses Problem heißt Red-Blue-Nonblocker.

Baumweite und Courcelle's Theorem

Ein Minor eines Graphen, ist ein Graph der aus ihm durch Kontraktion von Kanten und durch Löschen von Knoten und Kanten entsteht.

Lemma

Jeder planare Graph ist Minor eines Gitters.

Beweis.

Einfach. Zum Beispiel durch Induktion. □

Baumweite und Courcelle's Theorem

Theorem

Sei H ein endlicher, planarer Graph und $\mathcal{G}$ eine Klasse von Graphen, die H nicht als Minor enthalten.
Dann gibt es eine Konstante c_H, so daß die Baumweite jedes Graphen in $\mathcal{G}$ höchstens c_H ist.

Beweis.

sehr schwierig und lang $\square$

Baumweite und Courcelle's Theorem

Theorem

Sei H ein endlicher, planarer Graph und $\mathcal{G}$ eine Klasse von Graphen, die H nicht als Minor enthalten.
Dann gibt es eine Konstante c_H, so daß die Baumweite jedes Graphen in $\mathcal{G}$ höchstens c_H ist.

Beweis.

sehr schwierig und lang $\square$

Baumweite und Courcelle's Theorem

Korollar
„Hat ein Graph große Baumweite, dann enthält er ein großes Gitter“:

Gilt $tw(G) > t$, dann enthält G das Gitter $Q_f(t)$ als Minor, wobei f eine monotone, unbeschränkte Funktion ist.

Beweis
direkte Folge des letzten Theorems

Baumweite und Courcelle's Theorem

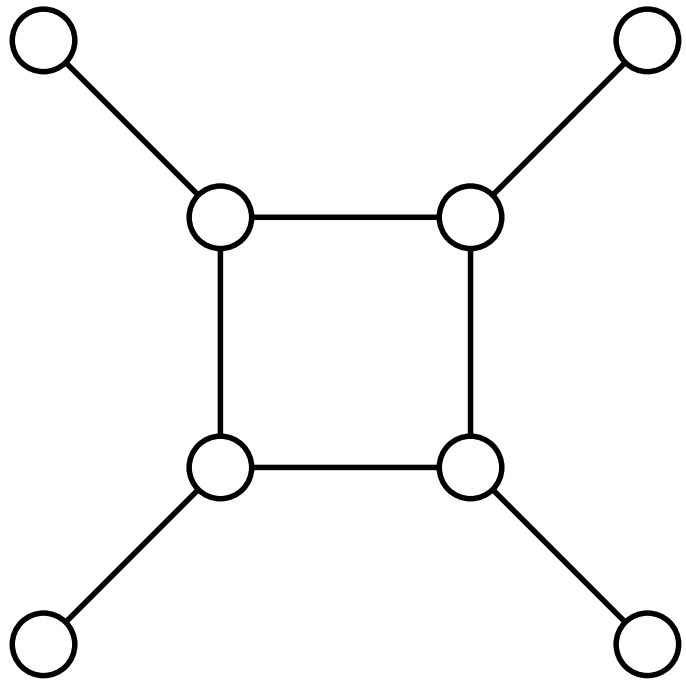
Beispiel
Gegeben: Ein planarer Graph G und k Paare (s_i, t_i) von Knoten aus G. (Parameter ist k)

Frage: Gibt es kantendisjunkte Pfade, die je s_i mit t_i verbinden?

Dieses Problem ist in FPT:
Ist die Baumweite klein, so können wir Courcelle's Theorem anwenden.
Ist die Baumweite groß, dann gibt es ein großes Gitter als Minor.
Aus diesem Gitter läßt sich ein Knoten entfernen, ohne daß dies „schadet“.

Color-Coding

Problem: Enthält ein Graph einen Kreis der Länge k?
Dieser Problem ist NP-vollständig, weil Hamiltonkreis ein Spezialfall ist.



Frage: Ist dieses Problem fixed parameter tractable?

Color-Coding

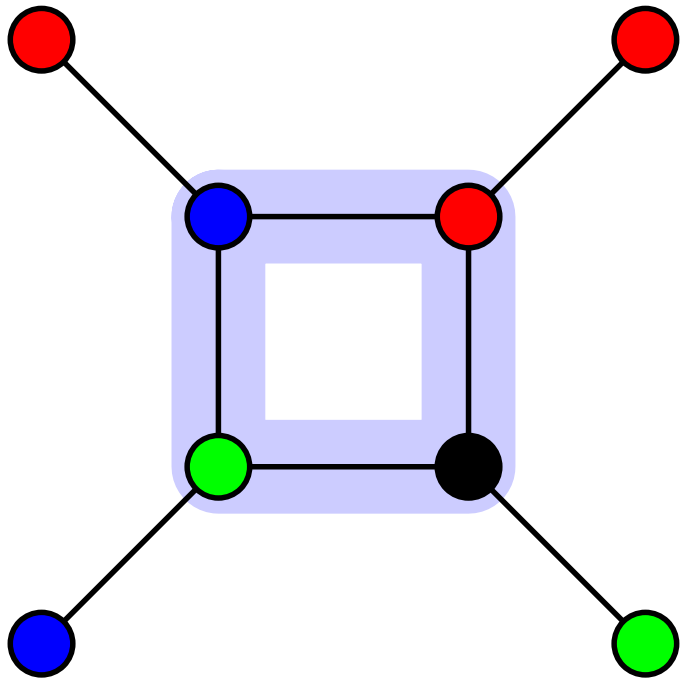
Algorithmus:

1. Färbe jeden Knoten zufällig mit einer von k Farben
2. Stelle fest ob es einen farbenfrohen Kreis der Länge k gibt, also einen Kreis, dessen Knoten paarweise verschieden gefärbt sind.

Analyse:
Ein Kreis der Länge k ist farbenfroh mit Wahrscheinlichkeit

$$k!/k^k \sim \sqrt{2\pi k}\, e^{-k}.$$

Color-Coding



Der Kreis ist mit Wahrscheinlichkeit $4!/4^4 = 3/32$ farbenfroh.

Color-Coding

Versuchen wir N mal, einen farbenfrohen Kreis der Länge k zu finden, dann ist die Wahrscheinlichkeit, daß der Algorithmus jedesmal versagt

$$\left(1 - \frac{k!}{k^k}\right)^N.$$

Setzen wir $N = Mk^k/k! \sim Me^k/\sqrt{k}$, dann ergibt sich

$$\left(1 - \frac{M}{N}\right)^N \sim e^{-M}.$$

Wir können die Wahrscheinlichkeit durch Wahl von M beliebig klein machen.

Color-Coding

Noch offene Frage:

Wie testet man, ob ein Graph einen farbenfrohen Kreis enthält?

Color-Coding

Antwort:

Lege Tabelle $P(u, v, l)$ an.

$P(u, v, l)$ enthalte alle Mengen von paarweise verschiedenen Knoten, die einen Pfad von u nach v der Länge l bilden.

Mit Hilfe von $P(u, v, l-1)$ läßt sich $P(u, v, l)$ berechnen.

Laufzeit: $2^k \cdot poly(n)$

Color-Coding

Definition

Eine k-perfekte Familie von Hashfunktionen ist eine Familie $\mathcal{F}$ von Funktionen $\{1, \ldots, n\} \to \{1, \ldots, k\}$, so daß zu jedem $S \subseteq \{1, \ldots, n\}$ mit $|S| = k$ ein $f \in \mathcal{F}$ existiert, daß auf S bijektiv ist.

Nehmen wir zunächst an, wir haben so eine Familie perfekter Hashfunktionen...

Color-Coding

Deterministischer Algorithmus:

- Färbe den Graphen mittels jeden $f \in \mathcal{F}$.
- Stelle jeweils fest, ob ein farbenfroher Kreis der Länge k existiert.

Dieser Algorithmus funktioniert, falls wir eine k-perfekte Hashfamilie konstruieren können.
Der Algorithmus ist schnell, falls die Familie klein ist, sie klein dargestellt werden kann und schnell evaluierbar ist.

Color-Coding

Glücklicherweise gibt es Familien k-perfekter Hashfunktionen, die nur aus $O(1)^k \log n$ vielen Funktionen bestehen.

Sie lassen sich kompakt speichern.

Sie lassen sich schnell auswerten. (Also $f(i)$ ist schnell berechnet.)

Es gibt also einen deterministischen Algorithmus, um Kreise der Länge k zu finden.

Integer Linear Programming

Eingabe: Ein ganzzahliges lineares Programm mit k Variablen.

Parameter: k

Frage: Hat das ILP eine Lösung?

Dieses Problem ist fixed parameter tractable.

Die Laufzeit ist also $f(k)n^{O(1)}$, wobei f aber unangenehm ist.

Beweis: sehr kompliziert...

Übersicht

Tiefensuchbäume

Gegeben: Ein Graph G und eine Zahl k

Parameter: k

Frage: Gibt es einen Pfad der Länge k in G?

Konstruiere zunächst einen Tiefensuchbaum.

Was ist die schöne Eigenschaft eines Tiefensuchbaums?

Tiefensuchbäume

Gegeben: Ein Graph G und eine Zahl k

Parameter: k

Frage: Gibt es einen Pfad der Länge k in G?

Konstruiere zunächst einen Tiefensuchbaum.

Was ist die schöne Eigenschaft eines Tiefensuchbaums?

Ein einfaches Theorem

Theorem

Gegegen sei ein Graph G und eine Zahl k.
Dann läßt sich in polynomieller Zeit eines der folgenden finden:

1. *Ein Kreis mit mindestens Länge k*
2. *Eine Baumzerlegung mit Baumweite höchstens k*

Beweis.

$k+1$ Polizisten gehen im Gänsemarsch durch den Tiefensuchbaum. □

Ein einfaches Theorem

Theorem

Gegegen sei ein Graph G und eine Zahl k.
Dann läßt sich in polynomieller Zeit eines der folgenden finden:

1. *Ein Kreis mit mindestens Länge k*
2. *Eine Baumzerlegung mit Baumweite höchstens k*

Beweis.

$k + 1$ Polizisten gehen im Gänsemarsch durch den Tiefensuchbaum. $\square$

Lange Pfade

Mit diesem Satz lassen sich leicht Pfade der Länge k finden:

1. Falls wir einen langen Kreis finden, gibt es natürlich auch einen Pfad der Länge k
2. Sonst verwenden wir die Baumzerlegung und das Theorem von Courcelle:
   $\exists x_1 \ldots \exists x_{k+1}(inc(x_1, x_2) \wedge \cdots \wedge inc(x_k, x_{k+1}) \wedge x_1 \neq x_2 \ldots)$

Lange Pfade

Mit diesem Satz lassen sich leicht Pfade der Länge k finden:

1. Falls wir einen langen Kreis finden, gibt es natürlich auch einen Pfad der Länge k
2. Sonst verwenden wir die Baumzerlegung und das Theorem von Courcelle:
   $\exists x_1 \ldots \exists x_{k+1}(inc(x_1, x_2) \wedge \cdots \wedge inc(x_k, x_{k+1}) \wedge x_1 \neq x_2 \ldots)$

Lange Pfade

Mit diesem Satz lassen sich leicht Pfade der Länge k finden:

1. Falls wir einen langen Kreis finden, gibt es natürlich auch einen Pfad der Länge k
2. Sonst verwenden wir die Baumzerlegung und das Theorem von Courcelle:
   $\exists x_1 \ldots \exists x_{k+1}(inc(x_1, x_2) \wedge \cdots \wedge inc(x_k, x_{k+1}) \wedge x_1 \neq x_2 \ldots)$

Frage: Kann man so auch Vertex-Cover lösen?

Ein komplizierterer Satz

Theorem (Bodlaender)

Gegegen sei ein Graph G und Zahlen k, l.
Dann läßt sich in $f(k,l)|G|$ Zeit eines der folgenden in G finden:

1. *Eine Unterteilung eines $2 \times k$-Gitters*
2. *Eine Unterteilung des l-Zirkusgraphen*
3. *Eine Baumzerlegung von G der Baumweite $2(k-1)^2(l-1)+1$.*

Beweis.

Ähnlich mithilfe eines Tiefensuchbaums. □

7-Zirkusgraph

Ein komplizierterer Satz

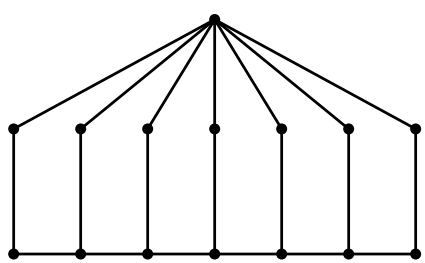
Theorem (Bodlaender)

Gegegen sei ein Graph G und Zahlen k, l.
Dann läßt sich in $f(k,l)|G|$ Zeit eines der folgenden in G finden:

1. *Eine Unterteilung eines $2 \times k$-Gitters*
2. *Eine Unterteilung des l-Zirkusgraphen*
3. *Eine Baumzerlegung von G der Baumweite $2(k-1)^2(l-1)+1$.*

Beweis.

Ähnlich mithilfe eines Tiefensuchbaums. □



7-Zirkusgraph

Ein komplizierterer Satz

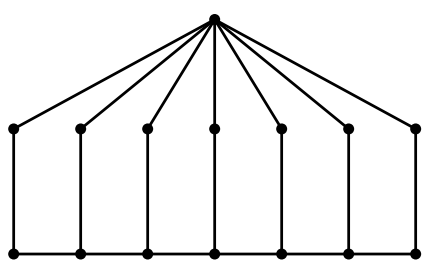
Theorem (Bodlaender)

Gegegen sei ein Graph G und Zahlen k, l.
Dann läßt sich in $f(k,l)|G|$ Zeit eines der folgenden in G finden:

1. *Eine Unterteilung eines $2 \times k$-Gitters*
2. *Eine Unterteilung des l-Zirkusgraphen*
3. *Eine Baumzer* Frage: Kann man so Dominating $2(k-1)^2(l-$ Set lösen?

Beweis.

Ähnlich mithilfe eines Tiefensuchbaums. □



7-Zirkusgraph

Ein komplizierterer Satz

Theorem (Bodlaender)

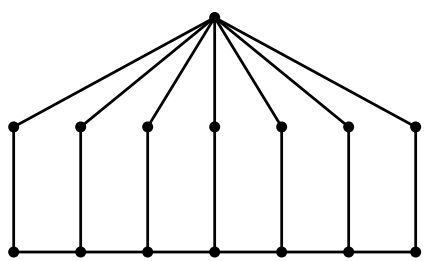
Gegegen sei ein Graph G und Zahlen k, l.
Dann läßt sich in $f(k,l)|G|$ Zeit eines der folgenden in G finden:

1. *Eine Unterteilung eines $2 \times k$-Gitters*
2. *Eine Unterteilung des l-Zirkusgraphen*
3. *Eine Baumzer*
   $2(k-1)^2(l-$

Frage: Kann man so Dominating Set lösen?

Beweis.
Ähnlich mith

Warum ging es bei planaren Graphen?



7-Zirkusgraph

Anwendung 1

Max-Leaf-Spanning-Tree:

Eingabe: Ein Graph G und eine Zahl k

Parameter: k

Frage: Gibt es in G einen Spannbaum mit mindestens k Blättern?

Sowohl ein $2 \times k$-Gitter als auch ein k-Zirkusgraph enthalten einen Baum mit k Blättern.
Also ist Max-Leaf-Spanning-Tree fixed parameter tractable.

Anwendung 1

Max-Leaf-Spanning-Tree:

Eingabe: Ein Graph G und eine Zahl k

Param

Frage: tens k Blättern?

Ist folgendes wahr?
Enthält ein Graph einen Baum mit k Blätterns, dann auch einen Spannbaum mit wenigstens k Blättern.

Sowohl ein $2 \times k$-Gitter als auch ein k-Zirkusgraph enthalten einen Baum mit k Blättern.
Also ist Max-Leaf-Spanning-Tree fixed parameter tractable.

Anwendung 2

Feedback Vertex Set:

Eingabe: Ein Graph G und eine Zahl k

Parameter: k

Frage: Gibt es $\leq k$ Knoten, deren Entfernung G azyklisch macht?

Theorem

Feedback Vertex Set ist fixed parameter tractable.

Anwendung 2

Feedback Vertex Set:

Eingabe: Ein Graph G und eine Zahl k

Parameter: k

Frage: Gibt es $\leq k$ Knoten, deren Entfernung G azyklisch macht?

Theorem

Feedback Vertex Set ist fixed parameter tractable.

Feedback Vertex Set

Theorem

Feedback Vertex Set ist fixed parameter tractable.

Beweis.

Wende Bodlaenders Theorem an.

1. Falls es eine kleine Baumzerlegung gibt: Courcelle
2. Falls es ein $2 \times 3k$-Gitter gibt: Nein
3. Falls es einen $4k$-Zirkusgraphen gibt: Entferne die Spitze und stelle fest, ob jetzt ein Feedback Vertex Set der Größe $k-1$ existiert.

□

Feedback Vertex Set

Theorem

Feedback Vertex Set ist fixed parameter tractable.

Beweis.

Wende Bodlaenders Theorem an.

1. Falls es eine kleine Baumzerlegung gibt: Courcelle
2. Falls es ein $2 \times 3k$-Gitter gibt: Nein
3. Falls es einen $4k$-Zirkusgraphen gibt: Entferne die Spitze und stelle fest, ob jetzt ein Feedback Vertex Set der Größe $k-1$ existiert.

□

Feedback Vertex Set

Theorem

Feedback Vertex Set ist fixed parameter tractable.

Beweis.

Wende Bodlaenders Theorem an.

1. Falls es eine kleine Baumzerlegung gibt: Courcelle
2. Falls es ein $2 \times 3k$-Gitter gibt: Nein
3. Falls es einen $4k$-Zirkusgraphen gibt: Entferne die Spitze und stelle fest, ob jetzt ein Feedback Vertex Set der Größe $k-1$ existiert.

□

Feedback Vertex Set

Theorem

Feedback Vertex Set ist fixed parameter tractable.

Beweis.

Wende Bodlaenders Theorem an.

1. Falls es eine kleine Baumzerlegung gibt: Courcelle
2. Falls es ein $2 \times 3k$-Gitter gibt: Nein
3. Falls es einen $4k$-Zirkusgraphen gibt: Entferne die Spitze und stelle fest, ob jetzt ein Feedback Vertex Set der Größe $k - 1$ existiert.

□

Feedback Vertex Set

Theorem

Feedback Vertex Set ist fixed parameter tractable.

Beweis.

Wende Bodlaenders Theorem an.

1. Falls es eine kleine Baumzerlegung gibt: Courcelle
2. Falls es ein $2 \times 3k$-Gitter gibt: Nein
3. Falls es einen $4k$-Zirkusgraphen gibt: Entferne die Spitze und stelle fest, ob jetzt ein Feedback Vertex Set der Größe $k - 1$ existiert.

□

Feedback Vertex Set

Theorem

Feedback Vertex Set ist fixed parameter tractable.

Beweis.

Wende Bodlaenders Theorem an.

1. Falls es eine kleine Baumzerlegung gibt: Courcelle
2. Falls es ein $2 \times 3k$-Gitter gibt: Nein
3. Falls es einen $4k$-Zirkusgraphen gibt: Entferne die Spitze und stelle fest, ob jetzt ein Feedback Vertex Set der Größe $k - 1$ existiert.

□

Crown Decompositions

Definition

Gegeben sei ein ungerichteter Graph $G = (V, E)$.
Zwei disjunkte Mengen von Knoten (I, H) sind eine Krone, falls

1. $I \neq \emptyset$ eine unabhängige Menge ist
2. $H = N(I)$
3. Es gibt ein Matching zwischen I und H der Größe $|H|$

Konstruktion von Kronen

Folgender Algorithmus konstruiert eine Krone:

1. Finde maximales Matching M_1, nenne die übrigbleibenden Knoten O
2. Finde Matching maximaler Kardinalität M_2 zwischen O und $N(O)$
3. Nenne die Knoten in O aber außerhalb M_2 jetzt I_0
4. **for** $i = 0, \ldots, n$ **do**
   - $H_i := N(I_i)$
   - $I_{i+1} := I_i \cup N_{M_2}(H_i)$
5. **return** (I_n, H_n)

Korrektheit

Theorem

Der Algorithmus ist korrekt: Er konstuiert eine Krone, falls $I_0 \neq \emptyset$.

Beweis.

1. $I_n \subseteq O$ ist unabhängig, weil M_1 maximal
2. $H_n = N(I_n)$ nach Konstruktion
3. Jeder Knoten in H_n ist mit I_n über M_2 gepaart. Wenn nicht, dann gäbe es einen Pfad ungerader Länge, der abwechselnd aus Kanten nicht in bzw. in M_2 besteht.
   Widerspruch zur Maximalität von M_2.

□

Korrektheit

Theorem

Der Algorithmus ist korrekt: Er konstuiert eine Krone, falls $I_0 \neq \emptyset$.

Beweis.

1. $I_n \subseteq O$ ist unabhängig, weil M_1 maximal
2. $H_n = N(I_n)$ nach Konstruktion
3. Jeder Knoten in H_n ist mit I_n über M_2 gepaart. Wenn nicht, dann gäbe es einen Pfad ungerader Länge, der abwechselnd aus Kanten nicht in bzw. in M_2 besteht.
   Widerspruch zur Maximalität von M_2.

□

Korrektheit

Theorem

Der Algorithmus ist korrekt: Er konstuiert eine Krone, falls $I_0 \neq \emptyset$.

Beweis.

1. $I_n \subseteq O$ ist unabhängig, weil M_1 maximal
2. $H_n = N(I_n)$ nach Konstruktion
3. Jeder Knoten in H_n ist mit I_n über M_2 gepaart. Wenn nicht, dann gäbe es einen Pfad ungerader Länge, der abwechselnd aus Kanten nicht in bzw. in M_2 besteht.
   Widerspruch zur Maximalität von M_2.

□

Crown Reductions für Vertex Cover

Falls ein Graph $G = (V, E)$ eine Krone (I, H) besitzt, dann gibt es ein optimales Vertex Cover C mit

- $H \subseteq C$
- $I \cap C = \emptyset$

Falls $|M_1| > k$ oder $|M_2| > k$, dann kann es kein Vertex Cover der Größe k geben.
Andernfalls $|O| \geq n - 2k$.
Höchstens k Knoten in O sind in M_2.
$\Rightarrow |I_0| \geq |O| - k \geq n - 3k$.
$\Rightarrow$ Falls $n > 3k$, dann findet man eine Krone.

Crown Reductions für Vertex Cover

Falls ein Graph $G = (V, E)$ eine Krone (I, H) besitzt, dann gibt es ein optimales Vertex Cover C mit

- $H \subseteq C$
- $I \cap C = \emptyset$

Falls $|M_1| > k$ oder $|M_2| > k$, dann kann es kein Vertex Cover der Größe k geben.

Andernfalls $|O| \geq n - 2k$.

Höchstens k Knoten in O sind in M_2.

$\Rightarrow |I_0| \geq |O| - k \geq n - 3k$.

$\Rightarrow$ Falls $n > 3k$, dann findet man eine Krone.

Crown Reductions für Vertex Cover

Falls ein Graph $G = (V, E)$ eine Krone (I, H) besitzt, dann gibt es ein optimales Vertex Cover C mit

- $H \subseteq C$
- $I \cap C = \emptyset$

Falls $|M_1| > k$ oder $|M_2| > k$, dann kann es kein Vertex Cover der Größe k geben.
Andernfalls $|O| \geq n - 2k$.
Höchstens k Knoten in O sind in M_2.
$\Rightarrow |I_0| \geq |O| - k \geq n - 3k$.
$\Rightarrow$ Falls $n > 3k$, dann findet man eine Krone.

Übersicht

Parametrisierte Komplexitätstheorie

Klassische Komplexitätstheorie:

- Komplexitätsklassen P, NP, etc.
- Sprachen $L \in P$, $L \subseteq \Sigma^*$
- Rahmen reicht nicht für parametrisierte Sicht

Parametrisierte Komplexitätstheorie

Definition

Ein parametrisiertes Problem über dem Alphabet Σ ist eine Menge von Paaren (w, k), wobei $w \in \Sigma^*$ und $k \in \mathbf{N}$.
Es darf keine w und $k \neq k'$ geben mit $(w, k) \in L$ und $(w, k') \in L$, wenn L ein parametrisiertes Problem ist.

Die zweite Bedingung besagt, daß k eine Funktion von w ist.

Parametrisierte Komplexitätstheorie

Wir beschreiben parametrisierte Probleme gerne so:

Eingabe: Ein Graph G und eine Zahl k

Parameter: k

Frage: Hat G eine Clique der Größe k?

Parametrisierte Komplexitätstheorie

Der Parameter kann aber etwas beliebig anderes sein, solange er eine Funktion der Eingabe ist und aus ihr leicht berechnet werden kann.

Eingabe: Ein Graph G und eine Zahl k

Parameter: Der Durchmesser von G

Frage: Hat G eine Clique der Größe k?

Wir können nämlich leicht $(G, \Delta(G))$ aus G berechnen, um dann formal ein parametrisiertes Problem zu erhalten.

Parametrisierte Komplexitätstheorie

Ein Ziel der Komplexitätstheorie ist es, Probleme in leicht und schwer zu klassifizieren.

Die bekanntesten Klassen sind hier P und NP.
Andere sind:

- NC und L
- AC^0 und NC^1
- $EXPTIME$ und $EXPSPACE$
- etc. etc.

Parametrisierte Komplexitätstheorie

In der parametrisierten Komplexitätstheorie sind die leichten Probleme in der Klasse *FPT*.

Definition

Die Klasse *FPT* enthält alle parametrisierten Probleme, die fixed parameter tractable sind.

Das heißt: $L \in FPT$, falls es einen Algorithmus gibt, der $(w, k) \in L$ in höchstens $f(k)|w|^c$ Schritten löst, wobei c eine Konstante und $f : \mathbf{N} \to \mathbf{N}$ eine beliebige Funktion ist.

Parametrisierte Komplexitätstheorie

Ein fundamentales Konzept in der Komplexitätstheorie sind Reduktionen.
Ein wichtiges Beispiel ist die many-one-Reduktion in polynomieller Zeit:
$g : \Sigma^* \to \Sigma^*$ reduziert das Problem L_1 auf L_2, wenn

1. $w \in L_1 \iff g(w) \in L_2$.
2. $g(w)$ kann in $|w|^{O(1)}$ Schritten berechnet werden.

Wichtige Eigenschaft: Falls L_1 auf L_2 reduziert werden kann und $L_1 \notin P$, dann auch $L_2 \notin P$.

"I can't find an efficient algorithm, but neither can all these famous people."

Wichtige Eigenschaft: Falls L_1 auf L_2 reduziert werden kann und $L_1 \notin P$, dann auch $L_2 \notin P$.

Parametrisierte Komplexitätstheorie

Frage: Ist diese Reduktion auch für parametrisierte Probleme brauchbar?

1. $w \in L_1 \iff g(w) \in L_2$.
2. $g(w)$ kann in $|w|^{O(1)}$ Schritten berechnet werden.

Gilt die entsprechende Eigenschaft: Falls L_1 auf L_2 reduziert werden kann und $L_1 \notin FPT$, dann auch $L_2 \notin FPT$.

Parametrisierte Komplexitätstheorie

Die entsprechende Eigenschaft gilt leider nicht:

Wir können (w, k) auf $(w, |w|)$ abbilden!

Falls wir ein Problem auf sich selbst reduzieren, haben wir statt $f(k)|w|^c$ auf einmal $f(|w|)|w|^c$ Schritte zur Verfügung.

So können wir aber jedes berechenbare Problem lösen.

Die bloße Polynomialzeitreduktion ist nicht fein genug.

Parametrisierte Reduktionen

Definition

Ein parametrisiertes Problem L_1 über Σ läßt sich auf ein parametrisiertes Problem L_2 über Γ parametrisiert reduzieren, wenn

- $r, s\colon \mathbf{N} \to \mathbf{N}$ berechenbare Funktionen sind,
- es eine Funktion $g\colon \Sigma^* \times \mathbf{N} \to \Gamma^*$, $(w, k) \mapsto (w', k')$ gibt, die in $r(k)|w|^{O(1)}$ Schritten berechenbar ist und $k' = s(k)$,
- $(w, k) \in L_1$ genau dann, wenn $g(w, k) \in L_2$.

Parametrisierte Reduktionen

Theorem

Falls $L_1 \notin FPT$ und es gibt eine parametrisierte Reduktion von L_1 auf L_2, dann ist $L_2 \notin FPT$.

Beweis.

Nehmen wir $L_2 \in FPT$ an. Wir können $(w', k') = g(w, k)$ in $r(k)|w|^c$ Schritten berechnen. Es gilt $k' = s(k)$ und $|w'| \leq r(k)|w|^c$.
Dann testen wir $(w', k') \in L_2$. Das benötigt $f'(k')|w'|^d \leq f'(s(k))r(k)^d|w|^{cd}$.
Da $(w, k) \in L_1 \iff (w', k') \in L_2$, haben wir so $(w, k) \in L_1$ beantwortet und daher ist $L_1 \in FPT$. $\square$

Parametrisierte Reduktionen

Betrachten wir einige klassische Reduktionen:

- Vertex Cover auf Independent Set
- CNF-SAT auf 3SAT (gewichtet)
- Clique auf Independent Set

Klassische Reduktionen gewöhnlich nicht parametrisiert

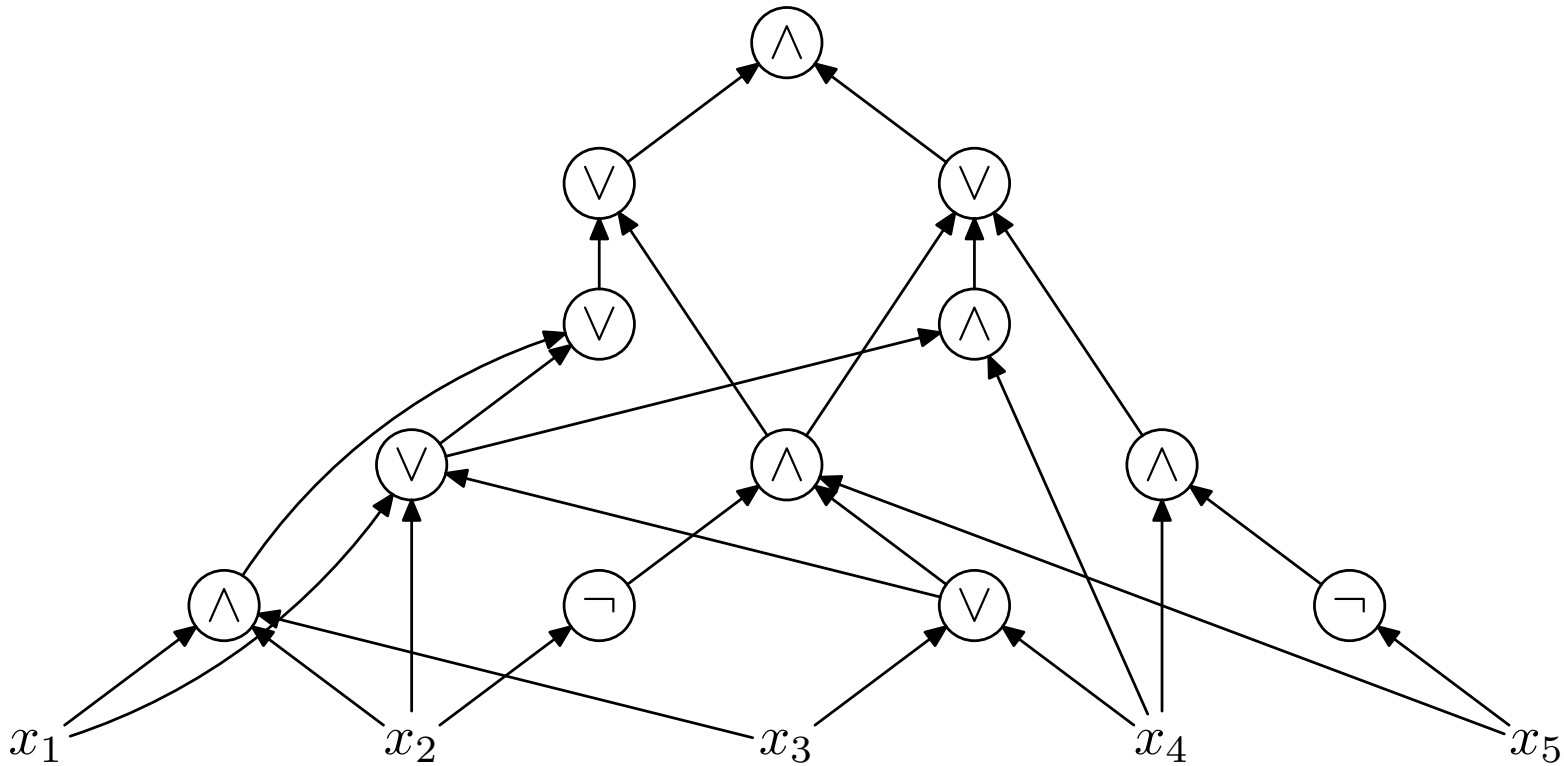
Parametrisierte Komplexitätstheorie

Definition

Ein boolesches Schaltnetz ist ein azyklisher Graph mit:

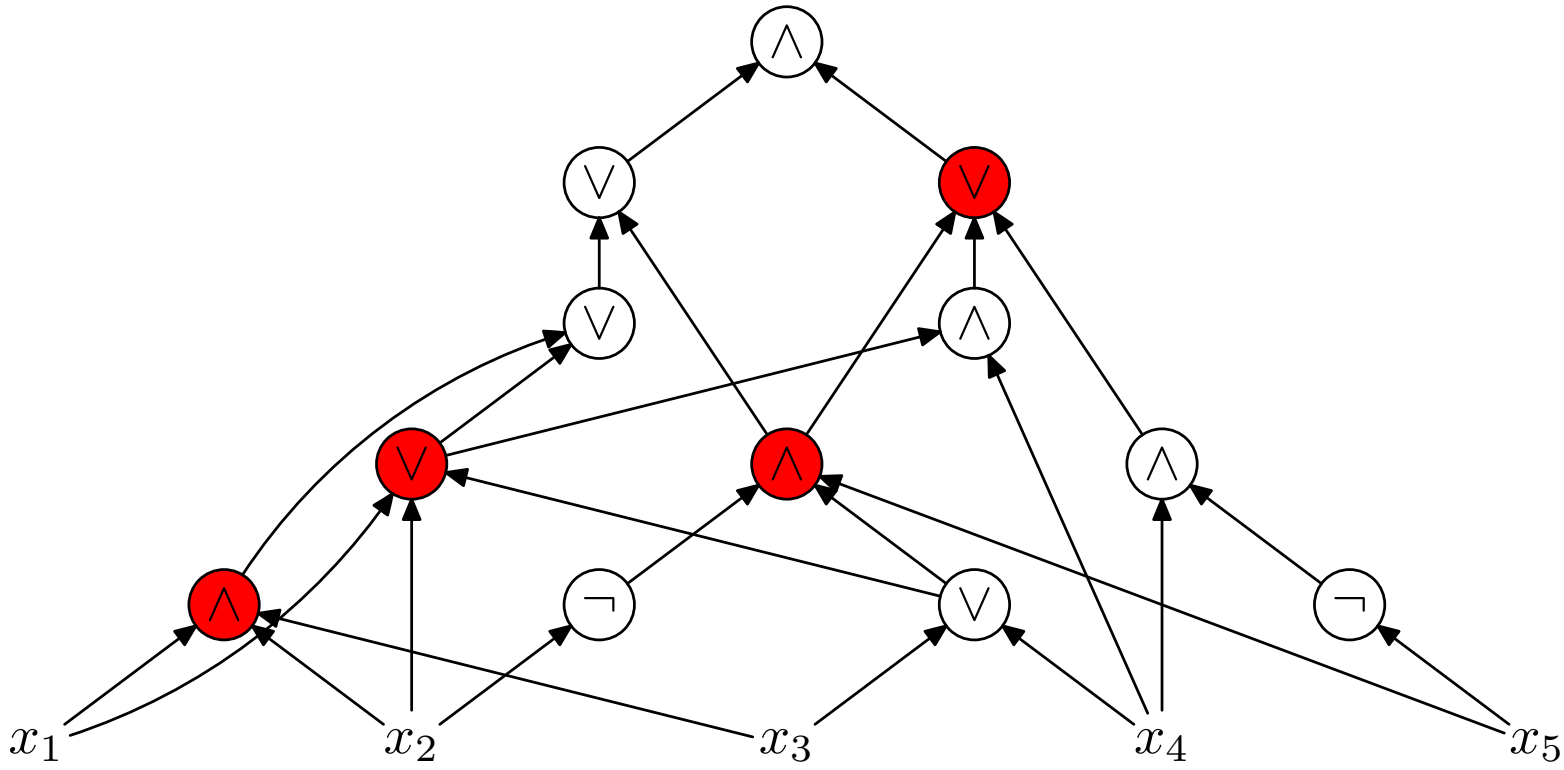
- Es gibt genau einen Knoten mit Ausgangsgrad 0, den Ausgang.
- Jeder Knoten mit Eingangsgrad 0 ist ein Eingang und mit x_i oder $\neg x_i$ beschriftet.
- Die anderen Knoten sind Gatter und sind mit $\wedge$, $\vee$ oder $\neg$ (bei Eingangsgrad 1) beschriftet.

Parametrisierte Komplexitätstheorie



Ob ein Schaltnetz eine erfüllende Belegung besitzt, ist *NP*-vollständig.

Parametrisierte Komplexitätstheorie



Große Gatter haben Eingangsgrad > 2.

Die Höhe ist die Länge eines längsten Pfads.

Die Weft ist die maximale Anzahl großer Gatter auf einem Pfad.

Parametrisierte Komplexitätstheorie

Definition
Sei $\mathcal{F}(t, h)$ die Menge der Schaltnetze mit Höhe h und Weft t.

Definition
Das gewichtete Erfüllbarkeitsproblem $L_{\mathcal{F}(t,h)}$:

Eingabe: (G, k), wobei $G \in \mathcal{F}(t, h)$
Parameter: k
Frage: Hat G eine erfüllende Belegung mit Gewicht k

Das Gewicht einer Belegung, ist die Anzahl der 1en

Parametrisierte Komplexitätstheorie

Definition

Ein parametrisiertes Problem gehört zur Komplexitätsklasse $W[t]$, wenn es parametrisiert auf $L_{\mathcal{F}(t,h)}$ für ein h reduziert werden kann.

Beispiel

Independent Set gehört zu $W[1]$.
Dominating Set gehört zu $W[2]$.
Frage: Warum?

Parametrisierte Komplexitätstheorie

Definition

Ein Problem L ist $W[t]$-schwer, wenn jedes Problem aus $W[t]$ parametrisiert auf L reduziert werden kann.

Definition

Ein Problem ist $W[t]$-vollständig, wenn es in $W[t]$ ist und gleichzeitig $W[t]$-schwer ist.

Parametrisierte Komplexitätstheorie

Theorem

Sei A ein $W[t]$-vollständiges Problem.
Es lasse sich A parametrisiert auf B reduzieren und $B \in W[t]$.
Dann ist B ebenfalls $W[t]$-vollständig.

Beweis.

Nach Vorraussetzung ist schon $B \in W[t]$.
Da sich jedes Problem in $W[t]$ auf A reduzieren läßt, folgt die $W[t]$-Härte aus der Transitivität parametrisierter Reduktionen. $\square$

Short Turing Machine Acceptance

Folgendes Problem wird sich später als $W[1]$-vollständig herausstellen:

Definition

Short Turing Machine Acceptance:

Eingabe: Eine nichtdeterministische Turingmaschine M, ein Wort w, eine Zahl k.
Parameter: k
Frage: Hat M auf Eingabe w einen akzeptierenden Pfad mit höchstens k Schritten?

Die Klasse $W[1, s]$ und $W[1, 2]$

Wir betrachten nun das gewichtete Erfüllbarkeitsproblem für sehr einfache Schaltnetze.

Definition

Sei $s > 1$ ein Zahl. Mit $\mathcal{F}(s)$ bezeichnen wir die Familie aller Schaltnetze, dessen Ausgang ein UND-Gatter ist, welches mit ODER-Gattern verbunden ist, deren Eingangsgrad höchstens s ist und die wiederum mit Literalen (also Variablen oder negierten Variablen) verbunden sind.

Wir definieren $W[1, s]$ als die Klasse aller Probleme in $W[1]$, die sich auf $L_{\mathcal{F}}(s)$ reduzieren lassen.

Im folgenden werden wir folgendes Theorem beweisen:

Theorem

Short Turing Machine Acceptance $\in W[1,2]$

Zu diesem Behufe benötigen wir eine Reduktion von Short Turing Machine Acceptance auf $L_{\mathcal{F}(2)}$.

Wir müssen M, w, k auf ein Schaltnetz und eine Zahl $k' = f(k)$ abbilden, so daß es eine erfüllende Belegung mit Gewicht k' genau dann gibt, wenn M das Wort w in höchstens k Schritten akzeptiert.

Im folgenden werden wir folgendes Theorem beweisen:

Theorem

Short Turing Machine Acceptance $\in W[1,2]$

Zu diesem Behufe benötigen wir eine Reduktion von Short Turing Machine Acceptance auf $L_{\mathcal{F}(2)}$.

Wir müssen M, w, k auf ein Schaltnetz und eine Zahl $k' = f(k)$ abbilden, so daß es eine erfüllende Belegung mit Gewicht k' genau dann gibt, wenn M das Wort w in höchstens k Schritten akzeptiert.

Wir können uns die Konfigurationen von M numeriert vorstellen und daher mit den Zahlen $1, 2, \ldots$ identifizieren.
Ein Konfiguration umfaßt dabei

- die Position des Schreib-/Lesekopfes,
- und den Zustand.

Falls i eine Konfiguration und a ein Zeichen ist, dann sei $\delta(i, a) = (j, b)$, wenn M beim Lesen von a in die Konfiguration j wechselt und das a durch ein b überschreibt.
Mit der Variablen $C_{t,i,j,a,b}$ wollen wir modellieren, daß M im t-ten Schritt von der Konfiguration i nach j wechselt, dabei ein a liest und durch ein b überschreibt.

Wir können uns die Konfigurationen von M numeriert vorstellen und daher mit den Zahlen $1, 2, \ldots$ identifizieren.
Ein Konfiguration umfaßt dabei

- die Position des Schreib-/Lesekopfes,
- und den Zustand.

Falls i eine Konfiguration und a ein Zeichen ist, dann sei $\delta(i, a) = (j, b)$, wenn M beim Lesen von a in die Konfiguration j wechselt und das a durch ein b überschreibt.
Mit der Variablen $C_{t,i,j,a,b}$ wollen wir modellieren, daß M im t-ten Schritt von der Konfiguration i nach j wechselt, dabei ein a liest und durch ein b überschreibt.

Wir können uns die Konfigurationen von M numeriert vorstellen und daher mit den Zahlen $1, 2, \ldots$ identifizieren.
Ein Konfiguration umfaßt dabei

- die Position des Schreib-/Lesekopfes,
- und den Zustand.

Falls i eine Konfiguration und a ein Zeichen ist, dann sei $\delta(i, a) = (j, b)$, wenn M beim Lesen von a in die Konfiguration j wechselt und das a durch ein b überschreibt.
Mit der Variablen $C_{t,i,j,a,b}$ wollen wir modellieren, daß M im t-ten Schritt von der Konfiguration i nach j wechselt, dabei ein a liest und durch ein b überschreibt.

Wir können uns die Konfigurationen von M numeriert vorstellen und daher mit den Zahlen $1, 2, \ldots$ identifizieren.
Ein Konfiguration umfaßt dabei

- die Position des Schreib-/Lesekopfes,
- und den Zustand.

Falls i eine Konfiguration und a ein Zeichen ist, dann sei $\delta(i, a) = (j, b)$, wenn M beim Lesen von a in die Konfiguration j wechselt und das a durch ein b überschreibt.
Mit der Variablen $C_{t,i,j,a,b}$ wollen wir modellieren, daß M im t-ten Schritt von der Konfiguration i nach j wechselt, dabei ein a liest und durch ein b überschreibt.

Mit der Variablen $M_{t,p,a,b}$ wollen wir modellieren, daß am Anfang des t-ten Schritts auf der Position p des Arbeitsbandes das Symbol a steht und in diesem Schritt durch ein b überschrieben wird.
Das nächste Ziel besteht nun darin, eine Formel so anzugeben, daß erfüllende Belegungen Berechnungen der Turingmaschine M widerspiegeln.
Das heißt, es gibt genau dann eine erfüllende Belegung mit $C_{t,i,j,a,b} = 1$, wenn es eine Berechnung gibt, in welcher M im t-ten Schritt von der Konfiguration i nach j wechselt, dabei ein a liest und durch ein b überschreibt.
Jeder mögliche Berechnungspfad soll dabei genau einer erfüllenden Belegung entsprechen.
Ausserdem: Es soll nur Berechnungspfade der Länge k und erfüllende Belegungen mit Gewicht $f(k)$ geben!

Mit der Variablen $M_{t,p,a,b}$ wollen wir modellieren, daß am Anfang des t-ten Schritts auf der Position p des Arbeitsbandes das Symbol a steht und in diesem Schritt durch ein b überschrieben wird.
Das nächste Ziel besteht nun darin, eine Formel so anzugeben, daß erfüllende Belegungen Berechnungen der Turingmaschine M widerspiegeln.
Das heißt, es gibt genau dann eine erfüllende Belegung mit $C_{t,i,j,a,b} = 1$, wenn es eine Berechnung gibt, in welcher M im t-ten Schritt von der Konfiguration i nach j wechselt, dabei ein a liest und durch ein b überschreibt.
Jeder mögliche Berechnungspfad soll dabei genau einer erfüllenden Belegung entsprechen.
Ausserdem: Es soll nur Berechnungspfade der Länge k und erfüllende Belegungen mit Gewicht $f(k)$ geben!

Es gibt sehr viele Bedingungen an unsere Variablen, damit die Modellierung stimmt.
Wir wollen diese durch ein UND von ODERs ausdrücken.
Die Klauseln werden jetzt so gewählt, daß eine falsche Modellierung unmöglich wird, weil sie sofort zu einer unerfüllenden Belegung führt.
Es bleiben dann als erfüllende Belegungen genau diejenigen übrig, welche gegen keine Regel verstossen.

Regel 1:
„Es gibt nichts doppelt, da es ein Berechnungspfad ist“

Genauer: Zum Zeitpunkt t ist M in genau einem Zustand, wechselt zu genau einem anderen, liest dabei genau ein Zeichen und überschreibt dieses durch genau ein (anderes) Zeichen.

Wie läßt sich diese Tatsache durch Klauseln darstellen?

$$C_{t,i,j,a,b} \rightarrow \overline{C_{t,i',j',a',b'}}$$

beziehungsweise

$$\overline{C_{t,i,j,a,b}} \vee \overline{C_{t,i',j',a',b'}}$$

für alle $t, i, j, a, b, i', j', a', b'$ mit $(i, j, a, b) \neq (i', j', a', b')$ und

$$M_{t,p,a,b} \rightarrow \overline{M_{t,p,a',b'}}$$

für alle t, p, a, b, a', b' mit $(a, b) \neq (a', b')$.

Regel 1:
„Es gibt nichts doppelt, da es ein Berechnungspfad ist“

Genauer: Zum Zeitpunkt t ist M in genau einem Zustand, wechselt zu genau einem anderen, liest dabei genau ein Zeichen und überschreibt dieses durch genau ein (anderes) Zeichen.

Wie läßt sich diese Tatsache durch Klauseln darstellen?

$$C_{t,i,j,a,b} \rightarrow \overline{C_{t,i',j',a',b'}}$$

beziehungsweise

$$\overline{C_{t,i,j,a,b}} \vee \overline{C_{t,i',j',a',b'}}$$

für alle $t, i, j, a, b, i', j', a', b'$ mit $(i, j, a, b) \neq (i', j', a', b')$ und

$$M_{t,p,a,b} \rightarrow \overline{M_{t,p,a',b'}}$$

für alle t, p, a, b, a', b' mit $(a, b) \neq (a', b')$.

Regel 1:
„Es gibt nichts doppelt, da es ein Berechnungspfad ist“

Genauer: Zum Zeitpunkt t ist M in genau einem Zustand, wechselt zu genau einem anderen, liest dabei genau ein Zeichen und überschreibt dieses durch genau ein (anderes) Zeichen.

Wie läßt sich diese Tatsache durch Klauseln darstellen?

Frage:
Warum nur eine Möglichkeit? Es geht doch um nichtdeterministische Turingmaschinen?

beziehungsw

$$c_{t,i,j,a,b} \vee c_{t,i',j',a',b'}$$

für alle $t, i, j, a, b, i', j', a', b'$ mit $(i, j, a, b) \neq (i', j', a', b')$ und

$$M_{t,p,a,b} \rightarrow \overline{M_{t,p,a',b'}}$$

für alle t, p, a, b, a', b' mit $(a, b) \neq (a', b')$.

Regel 2:
„M und C muß zusammenpassen.“

Genauer: Wenn ein Zeichen gelesen wird, dann muß es dort auch stehen. Wird ein Zeichen irgendwohin geschrieben, muß es danach dort auch stehen.

Wie läßt sich diese Tatsache durch Klauseln darstellen?

$$C_{t,i,j,a,b} \rightarrow M_{t,p(i),a,b}$$

Für alle t, i, j, a, b, wobei $p(i)$ die Kopfposition der Konfiguration i bezeichnet.

Regel 2:
„M und C muß zusammenpassen.“

Genauer: Wenn ein Zeichen gelesen wird, dann muß es dort auch stehen. Wird ein Zeichen irgendwohin geschrieben, muß es danach dort auch stehen.

Wie läßt sich diese Tatsache durch Klauseln darstellen?

$$C_{t,i,j,a,b} \rightarrow M_{t,p(i),a,b}$$

Für alle t, i, j, a, b, wobei $p(i)$ die Kopfposition der Konfiguration i bezeichnet.

Regel 2:
„M und C muß zusammenpassen.“

Genauer: We dort auch stehen. Wird es danach dort auch ste

Frage:
Benötigen wir hier eine Art Rückrichtung?
„Falls ein Zeichen irgendwo steht, dann muß es auch gelesen werden.“

Wie läßt sich diese Tatsache durch Klauseln darstellen?

$$C_{t,i,j,a,b} \rightarrow M_{t,p(i),a,b}$$

Für alle t, i, j, a, b, wobei $p(i)$ die Kopfposition der Konfiguration i bezeichnet.

Regel 3:
„Aufeinanderfolgende Schritte müssen zusammenpassen.“

Genauer: Beende ich einen Schritt in einer bestimmten Konfiguration, dann muß der nächste Schritt dort auch wieder beginnen. Der Inhalt einer Zelle nach dem t-ten und vor dem $t+1$-ten Schritt ist identisch.

Wie läßt sich diese Tatsache durch Klauseln darstellen?

$$C_{t,i,j,a,b} \rightarrow \overline{C_{t+1,i',j',c,d}}$$

Für alle $t, i, j, i', j' a, b, c, d$ mit $i' \neq j$ und

$$M_{t,p,a,b} \rightarrow \overline{M_{t+1,p,c,d}}$$

für alle t, p, a, b, c, d mit $b \neq c$.

Regel 3:
„Aufeinanderfolgende Schritte müssen zusammenpassen.“

Genauer: Beende ich einen Schritt in einer bestimmten Konfiguration, dann muß der nächste Schritt dort auch wieder beginnen. Der Inhalt einer Zelle nach dem t-ten und vor dem $t+1$-ten Schritt ist identisch.

Wie läßt sich diese Tatsache durch Klauseln darstellen?

$$C_{t,i,j,a,b} \rightarrow \overline{C_{t+1,i',j',c,d}}$$

Für alle $t, i, j, i', j' a, b, c, d$ mit $i' \neq j$ und

$$M_{t,p,a,b} \rightarrow \overline{M_{t+1,p,c,d}}$$

für alle t, p, a, b, c, d mit $b \neq c$.

Regel 4: „Die Übergangsrelation der Turingmaschine muß respektiert werden.“

Genauer: Falls $(j, b) \notin \delta(i, a)$, dann dürfen wir nicht von Konfiguration i nach j wechseln und dabei ein a durch ein b ersetzen.

$$\overline{C_{t,i,j,a,b}}$$

für $(j, b) \notin \delta(i, a)$.

Regel 4: „Die Übergangsrelation der Turingmaschine muß respektiert werden.“

Genauer: Falls $(j, b) \notin \delta(i, a)$, dann dürfen wir nicht von Konfiguration i nach j wechseln und dabei ein a durch ein b ersetzen.

$$\overline{C_{t,i,j,a,b}}$$

für $(j, b) \notin \delta(i, a)$.

Regel 5:

„Der Anfang und das Ende müssen stimmen. Der Berechnungspfad muß akzeptierend sein.“

→ Übungsaufgabe

Da alle Regeln gleichzeitig erfüllt werden müssen, können wir sie durch ein riesiges UND verbinde.
Das ergibt eine $\mathcal{F}(2)$-Formel, wie gewünscht.
Es gibt einen akzeptierenden Berechnungspfad der Länge k genau dann, wenn es eine erfüllende Belegung des Gewichts k' gibt.

Regel 5:

„Der Anfang und das Ende müssen stimmen. Der Berechnungspfad muß akzeptierend sein.“

$\longrightarrow$ Übungsaufgabe

Da alle Regeln gleichzeitig erfüllt werden müssen, können wir sie durch ein riesiges UND verbinde.
Das ergibt eine $\mathcal{F}(2)$-Formel, wie gewünscht.
Es gibt einen akzeptierenden Berechnungspfad der Länge k genau dann, wenn es eine erfüllende Belegung des Gewichts k' gibt.

Regel 5:

„Der Anfang und das Ende müssen stimmen. Der Berechnungspfad muß akzeptierend sein.“

→ Übungsaufgabe

Frage:
Wie groß ist k'?

Da all[illegible] den müssen, können wir sie durch [illegible]
Das ergibt eine $\mathcal{F}(2)$-Formel, wie gewünscht.
Es gibt einen akzeptierenden Berechnungspfad der Länge k genau dann, wenn es eine erfüllende Belegung des Gewichts k' gibt.

Zur Erinnerung.

Wir haben gerade folgendes bewiesen:

Theorem
Short Turing Machine Acceptance $\in W[1,2]$

Und mit der kleinen Modifikation:

Theorem
Short Turing Machine Acceptance $\in$ *Antimonoton-W*$[1,2]$

Zur Erinnerung.

Wir haben gerade folgendes bewiesen:

Theorem
Short Turing Machine Acceptance $\in W[1,2]$

Und mit der kleinen Modifikation:

Theorem
Short Turing Machine Acceptance $\in$ *Antimonoton-W*$[1,2]$

Die Klasse Antimonoton-$W[1, s]$

Wir betrachten nun das gewichtete Erfüllbarkeitsproblem für noch einfachere Schaltnetze.

Definition

Sei $s > 1$ ein Zahl. Mit Antimonoton-$\mathcal{F}(s)$ bezeichnen wir die Familie aller Schaltnetze, dessen Ausgang ein UND-Gatter ist, welches mit ODER-Gattern verbunden ist, deren Eingangsgrad höchstens s ist und die wiederum mit negativen Literalen (also negierten Variablen) verbunden sind.

Wir definieren Antimonoton-$W[1, s]$ als die Klasse aller Probleme in $W[1]$, die sich auf $L_{\text{Antimonoton-}\mathcal{F}(s)}$ reduzieren lassen.

Theorem

$L_{\text{Antimonoton-}\mathcal{F}(s)}$ läßt sich parametrisiert auf Short Turing Machine Acceptance reduzieren.

Corollary

$\text{Antimonoton-}W[1, s] \subseteq \text{Antimonoton-}W[1, 2]$

Beweis.

Erzeuge eine Turingmaschine, die folgendes tut:

1. Rate k Variablen auf das Band.
2. Gehe alle Teilmengen der Größe s davon durch.
3. Verifiziere dabei, daß die Teilmenge keine Klausel ausfüllt.

□

Eigentliches Ziel:

$W[1] \subseteq \text{Antimonoton-}W[1, 2]$

Theorem

$L_{\text{Antimonoton-}\mathcal{F}(s)}$ läßt sich parametrisiert auf Short Turing Machine Acceptance reduzieren.

Corollary

Antimonoton-$W[1, s] \subseteq$ Antimonoton-$W[1, 2]$

Beweis.

Erzeuge eine Turingmaschine, die folgendes tut:

1. Rate k Variablen auf das Band.
2. Gehe alle Teilmengen der Größe s davon durch.
3. Verifiziere dabei, daß die Teilmenge keine Klausel ausfüllt.

□

Eigentliches Ziel:

$W[1] \subseteq$ Antimonoton-$W[1, 2]$

Theorem

$L_{\text{Antimonoton-}\mathcal{F}(s)}$ läßt sich parametrisiert auf Short Turing Machine Acceptance reduzieren.

Corollary

Antimonoton-$W[1, s] \subseteq$ Antimonoton-$W[1, 2]$

Beweis.

Erzeuge eine Turingmaschine, die folgendes tut:

1. Rate k Variablen auf das Band.
2. Gehe alle Teilmengen der Größe s davon durch.
3. Verifiziere dabei, daß die Teilmenge keine Klausel ausfüllt.

□

Eigentliches Ziel:

$W[1] \subseteq$ Antimonoton-$W[1, 2]$

Theorem

$L_{\text{Antimonoton-}\mathcal{F}(s)}$ läßt sich parametrisiert auf Short Turing Machine Acceptance reduzieren.

Corollary

Antimonoton-$W[1, s] \subseteq$ Antimonoton-$W[1, 2]$

Beweis.

Erzeuge eine [illegible] tut:

Frage:
Wie ist die Laufzeit der Turingmaschine?

1. Rate k Variablen auf das Band.
2. Gehe alle Teilmengen der Größe s davon durch.
3. Verifiziere dabei, daß die Teilmenge keine Klausel ausfüllt.

□

Eigentliches Ziel:

$W[1] \subseteq$ Antimonoton-$W[1, 2]$

Die Klasse $W[1, 1, s]$

Wir betrachten jetzt das gewichtete Erfüllbarkeitsproblem für ganz spezielle, aber einfache Schaltnetze.

Definition

Sei $s > 1$ ein Zahl. Mit $\mathcal{F}(1, 1, s)$ bezeichnen wir die Familie aller Schaltnetze, dessen Ausgang ein ODER-Gatter ist, welches mit UND-Gattern verbunden ist, die wieder mit ODER-Gattern verbunden sind, deren Eingangsgrad höchstens s ist und die wiederum mit Eingängen oder negierten Eingängen verbunden sind.

Wir definieren $W[1, 1, s]$ als die Klasse aller Probleme in $W[1]$, die sich auf $L_{\mathcal{F}(1,1,s)}$ reduzieren lassen.

Vereinfachen von Weft-1-Schaltnetzen

Theorem

Gegeben sei ein Schaltnetz mit Weft 1 und Höhe h.
Dann läßt sich in polynomieller Zeit ein äquivalentes Schaltnetz in $\mathcal{F}(1, 1, s)$ *konstruieren, wobei s nur von h abhängt.*

Beweis.

- DNF und CNF
- de Morgan
- Kombination gleicher Gattertypen
- Distributivgesetz

□

Vereinfachen von Weft-1-Schaltnetzen

Theorem

Gegeben sei ein Schaltnetz mit Weft 1 und Höhe h.
Dann läßt sich in polynomieller Zeit ein äquivalentes Schaltnetz in $\mathcal{F}(1, 1, s)$ *konstruieren, wobei s nur von h abhängt.*

Beweis.

- DNF und CNF
- de Morgan
- Kombination gleicher Gattertypen
- Distributivgesetz

□

$L_{\mathcal{F}(1,1,s)}$

Ziel: $L_{\mathcal{F}(1,1,s)}$ auf STMA reduzieren.

Zwischenschritt:
$L_{\mathcal{F}(1,1,s)}$ auf je eine TM M_i für alle Unterschaltnetze des oberen Odergatters reduzieren.

Eine TM rät, welche M_i zur Simulation verwendet wird.

Noch offen:
$L_{\mathcal{F}(1,s)}$ auf STMA reduzieren.

$L_{\mathcal{F}(1,1,s)}$

Ziel: $L_{\mathcal{F}(1,1,s)}$ auf STMA reduzieren.

Zwischenschritt:
$L_{\mathcal{F}(1,1,s)}$ auf je eine TM M_i für alle Unterschaltnetze des oberen Odergatters reduzieren.

Eine TM rät, welche M_i zur Simulation verwendet wird.

Noch offen:
$L_{\mathcal{F}(1,s)}$ auf STMA reduzieren.

Reduktion von $L_{\mathcal{F}(1,s)}$ auf STMA:
Konstruiere eine TM, die eine Belegung auf das Band rät und dann zwei Zahlen berechnet:

- $A =$ Anzahl der Klauseln, die durch negierte Variablen erfüllt werden.
- $M =$ Anzahl der Klauseln, die nicht durch negierte, sondern nur durch positive Variablen erfüllt werden.

Belegung ist erfüllend gdw. $A + M =$ Anzahl der Klauseln.

Reduktion von $L_{\mathcal{F}(1,s)}$ auf STMA:
Konstruiere eine TM, die eine Belegung auf das Band rät und dann zwei Zahlen berechnet:

- $A =$ Anzahl der Klauseln, die durch negierte Variablen erfüllt werden.
- $M =$ Anzahl der Klauseln, die nicht durch negierte, sondern nur durch positive Variablen erfüllt werden.

Belegung ist erfüllend gdw. $A + M =$ Anzahl der Klauseln.

Reduktion von $L_{\mathcal{F}(1,s)}$ auf STMA:
Konstruiere eine TM, die eine Belegung auf das Band rät und dann zwei Zahlen berechnet:

- $A =$ Anzahl der Klauseln, die durch negierte Variablen erfüllt werden.
- $M =$ Anzahl der Klauseln, die nicht durch negierte, sondern nur durch positive Variablen erfüllt werden.

Belegung ist erfüllend gdw. $A + M =$ Anzahl der Klauseln.

Reduktion von $L_{\mathcal{F}(1,s)}$ auf STMA:
Konstruiere eine TM, die eine Belegung auf das Band rät und dann zwei Zahlen berechnet:

- $A =$ Anzahl der Klauseln, die durch negierte Variablen erfüllt werden.
- $M =$ Anzahl der Klauseln, die nicht durch negierte, sondern nur durch positive Variablen erfüllt werden.

Belegung ist erfüllend gdw. $A + M =$ Anzahl der Klauseln.

A kann wie im antimonotonen Fall berechnet werden.

Wie berechnen wir aber M?

Wir definieren $M(S, T)$ als die Anzahl der Klauseln, die genau die Variablen in T als negative Literale besitzen und durch positive Variablen aus S erfüllt werden.
Dann ist

$$M = \sum_{S,T \subseteq P, |S|,|T| \leq s} (-1)^{|S|+1} M(S, T),$$

wenn P die Variablen auf dem Band sind.

A kann wie im antimonotonen Fall berechnet werden.

Wie berechnen wir aber M?

Wir definieren $M(S, T)$ als die Anzahl der Klauseln, die genau die Variablen in T als negative Literale besitzen und durch positive Variablen aus S erfüllt werden.

Dann ist

$$M = \sum_{S,T \subseteq P, |S|,|T| \leq s} (-1)^{|S|+1} M(S, T),$$

wenn P die Variablen auf dem Band sind.

A kann wie im antimonotonen Fall berechnet werden.

Wie berechnen wir aber M?

Wir definieren $M(S, T)$ als die Anzahl der Klauseln, die genau die Variablen in T als negative Literale besitzen und durch positive Variablen aus S erfüllt werden.
Dann ist

$$M = \sum_{S,T \subseteq P, |S|, |T| \leq s} (-1)^{|S|+1} M(S, T),$$

wenn P die Variablen auf dem Band sind.