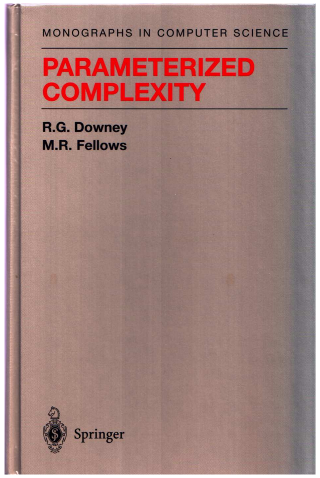
Parametrisierte Algorithmen

Dienstags 10–12 Uhr Seminarraum 4013 (Vorlesung)
Donnerstags 10–12 Uhr Seminarraum 4013 (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.

Einführung

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

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

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

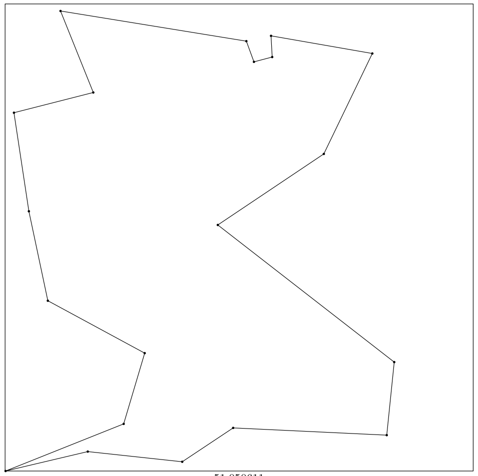
- $L \in NP$
- Jedes Problem in NP läßt sich in polynomieller Zeit auf L reduzieren.

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

NP-vollständige Probleme

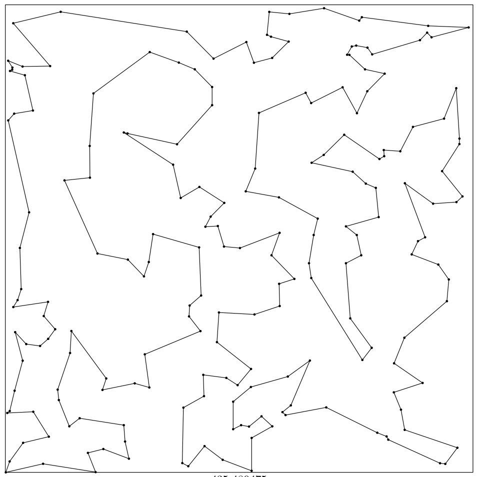
Warum ist SAT (satisfiability) NP-vollständig?
Weil sich eine Berechnung einer nichtdeterministischen Turingmaschine durch ein Schaltnetz simulieren läßt.
Daher kann die erfolgreiche Berechnung einer Turingmaschine auf die Existenz einer erfüllenden Belegung zurückgeführt werden.
Daher gibt es Formeln, 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



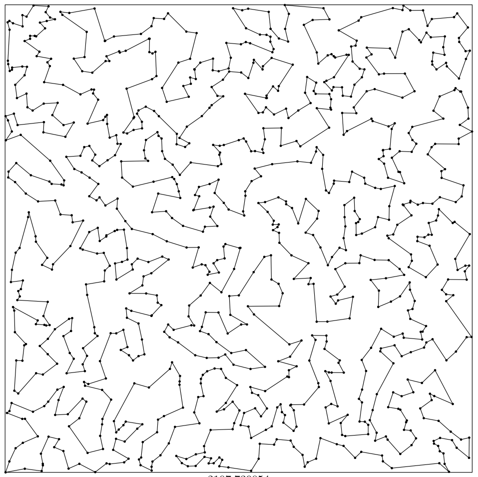
51.050611

Beispiel: TSP



435.489475

Beispiel: TSP



2107.739054

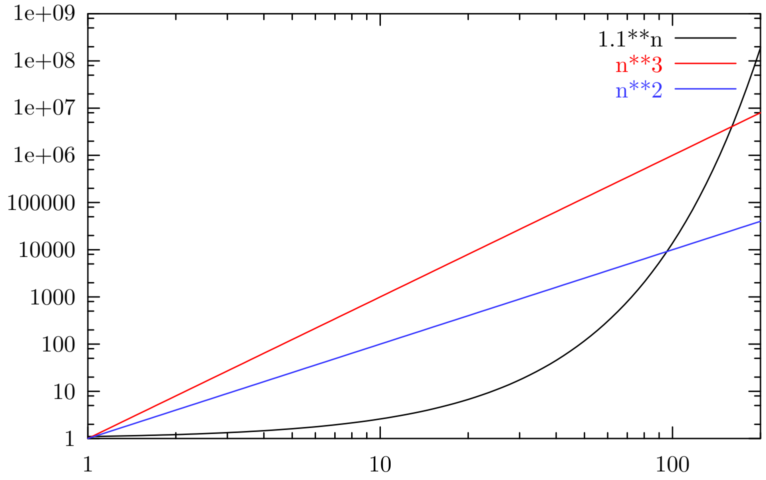
Laufzeiten

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

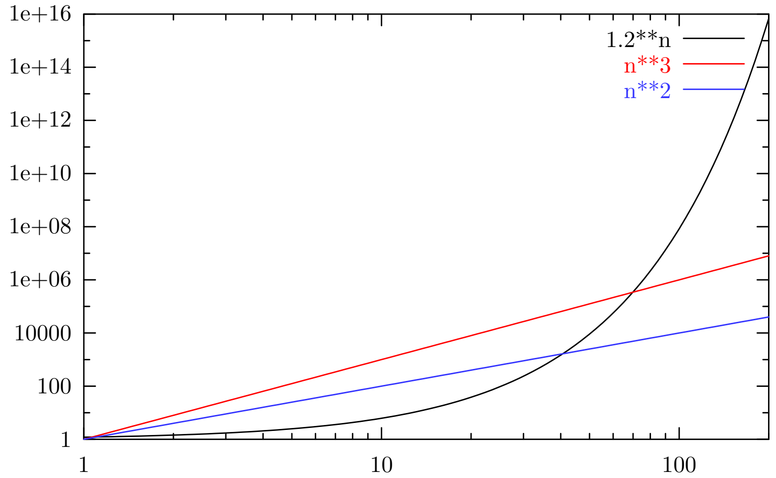
- Greedy-Algorithmen
- Divide-and-Conquer
- Dynamisches Programmieren

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

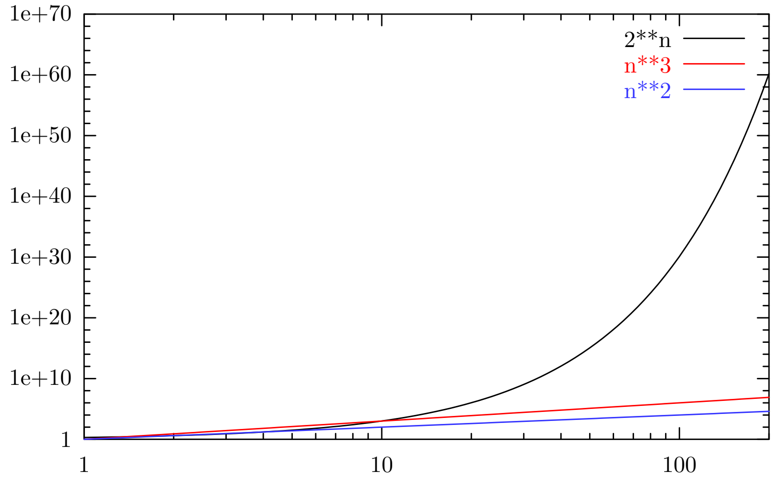
Laufzeitvergleich



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.

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!

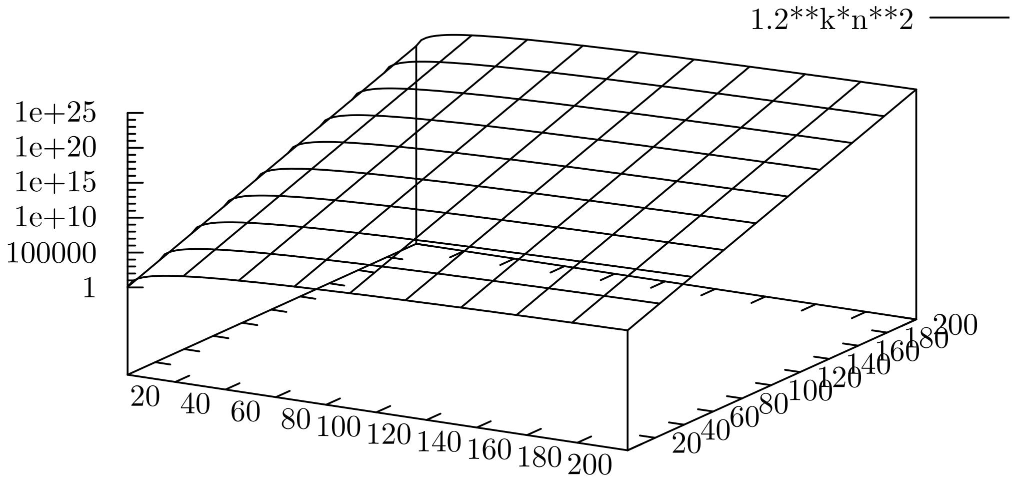
Zentrale Definition

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

$$\boxed{O(f(k)n^c)}$$

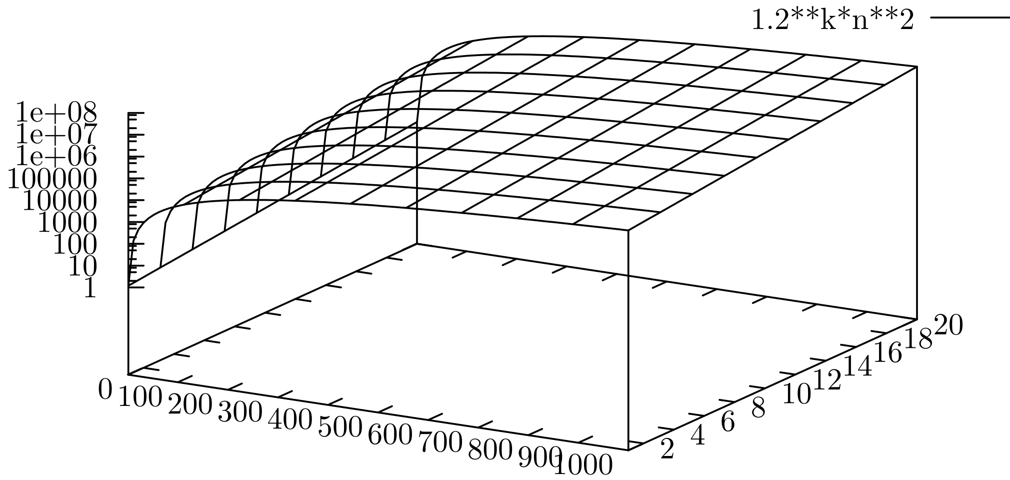
ist.
Dabei ist c eine Konstante und f eine beliebige Funktion.

Laufzeit eines parametrisierten Algorithmus



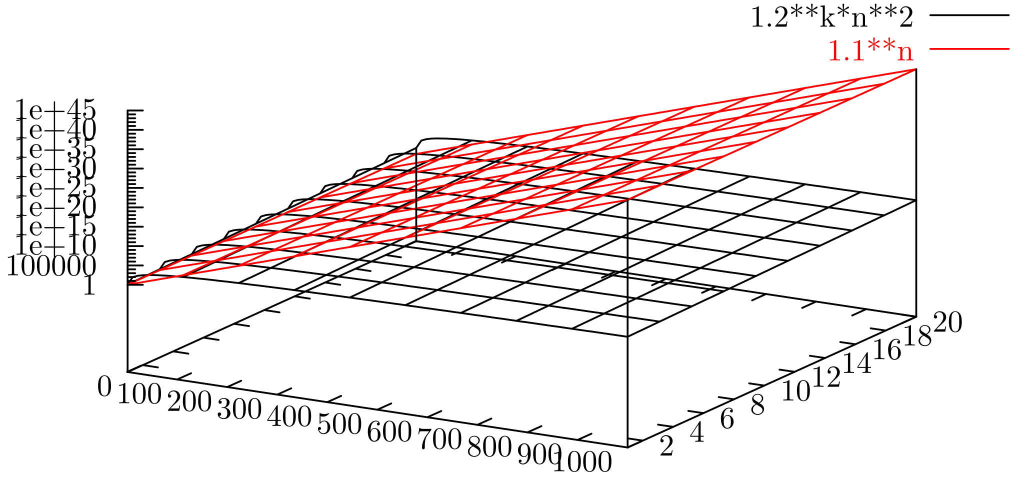
Laufzeit ist $1.2^k n^2$. Der Parameter liegt zwischen 1 und n.

Laufzeit eines parametrisierten Algorithmus



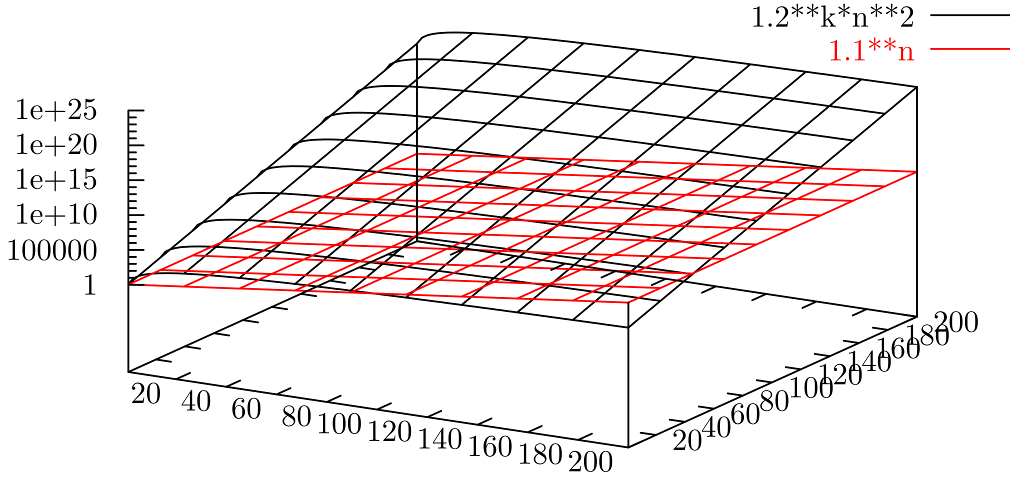
Laufzeit ist $1.2^k n^2$. Der Parameter ist klein.

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

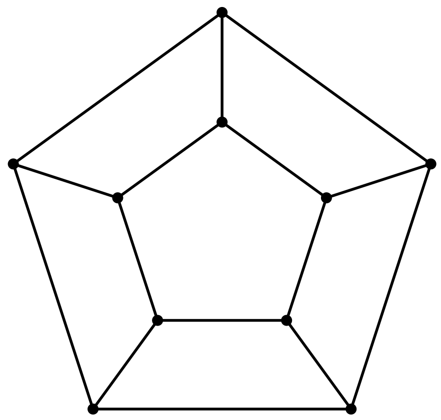


Beispiel: Vertex Cover

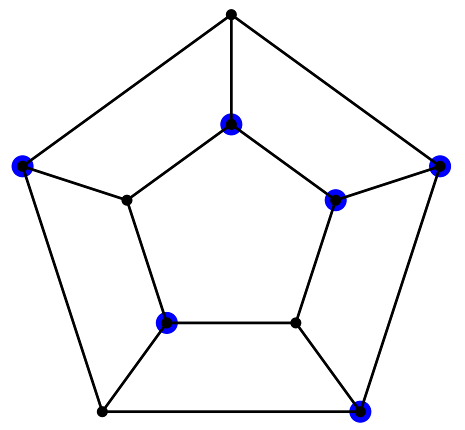
Gegeben: Ein Graph $G = (V, E)$.
Gesucht: Ein minimales Vertex Cover $C \subseteq E$.

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

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

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 $O(|E|2^{|V|})$.

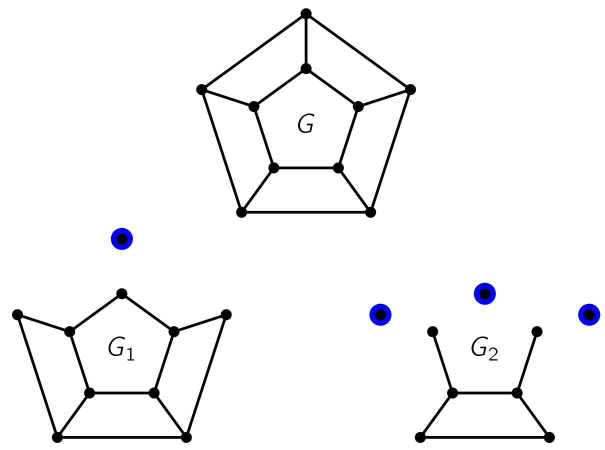
Backtracking

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.

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 $N(v) \cup VC(G_2)$



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:

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

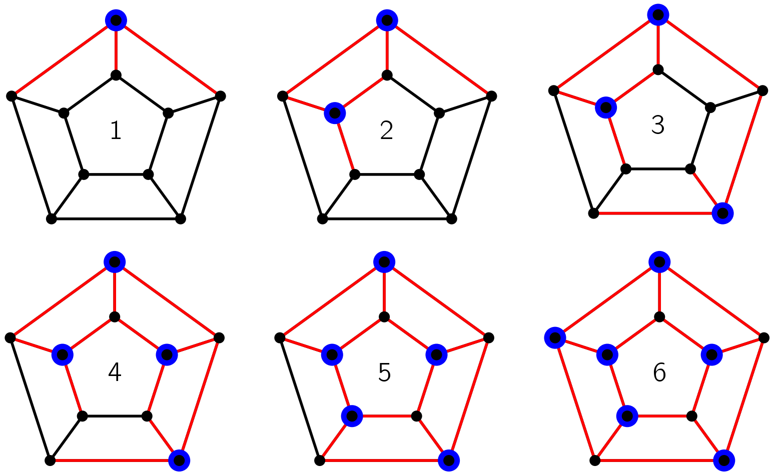
Wieder ergibt sich ein einfacher Algorithmus.

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 $E = \emptyset$ **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)

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.

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:

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

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 $h(s, s_i) \le m$ für alle $1 \le i \le k$.

Der Parameter sei m

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


```
agcacagtacgcaatagtgtcgcaggt
agctcagtagccaatagagtcccaggt
agatcagttcccaatagagtcgcacgt
agctcagtaaaaaatagagtcgcaggt
agcgcagtacacaatagagtcgcaagt
```



```
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.

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.

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.

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.

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.

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:

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

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.

Theorem

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

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.

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

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

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

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 $O(1.465572^k)$.

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 $(r_1, \dots, r_m)$, 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

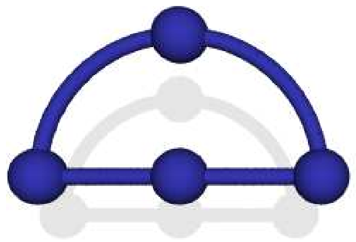
- Um Vertex Cover schneller lösen zu können, führen wir verschiedene Fälle ein.
- Zu jedem Fall wird ein korrekter Suchbaumschritt angegeben mit einem guten Verzweigungsvektor.
- Alle Fälle werden abgedeckt.
- Die größte Verzweigungszahl ergibt eine Schranke für die Größe des gesamten Suchbaums.

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

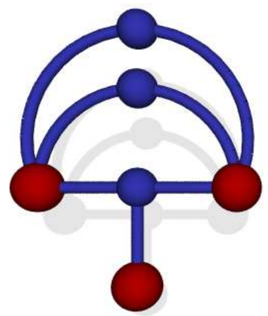
- Spielen große Rolle
- Dreiecke einfach
- 1. Fall: Es gibt Brücken
- 2. Fall: Es gibt keine Brücken

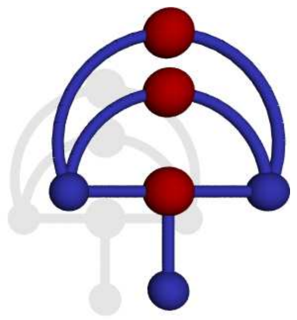
Grad 3

- 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

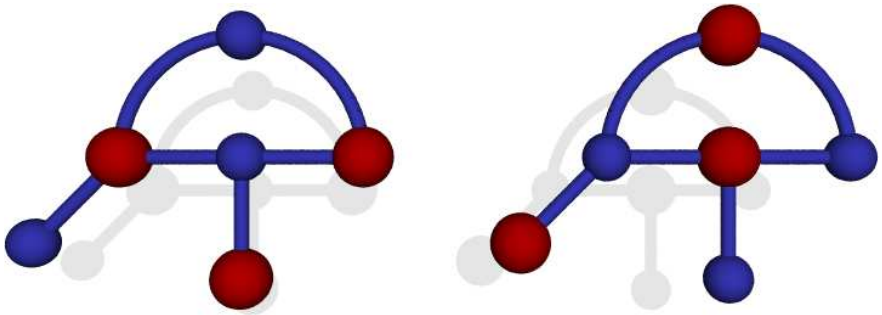
Grad 3 — 2 Brücken





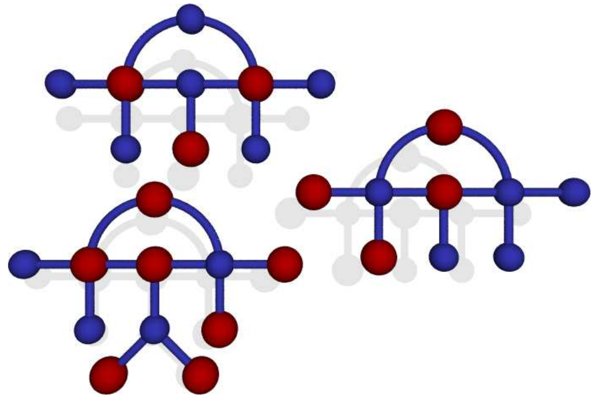
- Es gibt einen Knoten x mit Grad 3
- Er hat mindestens zwei Brücken

Grad 3 — 1 Brücke (a)



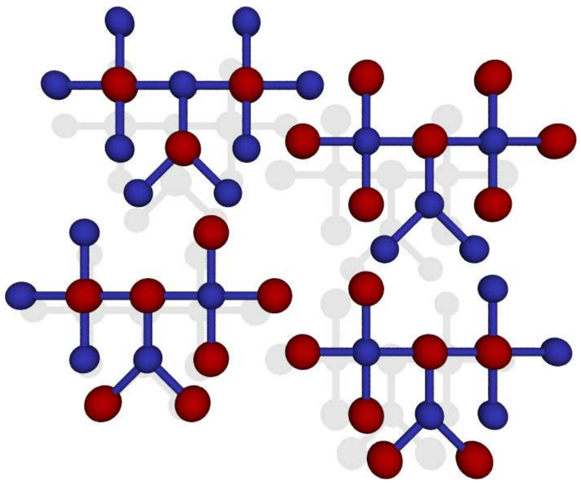
- Es gibt einen Knoten x mit Grad 3
- x hat genau eine Brücke
- x hat einen Brückennachbarn mit ebenfalls Grad 3

Grad 3 — 1 Brücke (b)



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

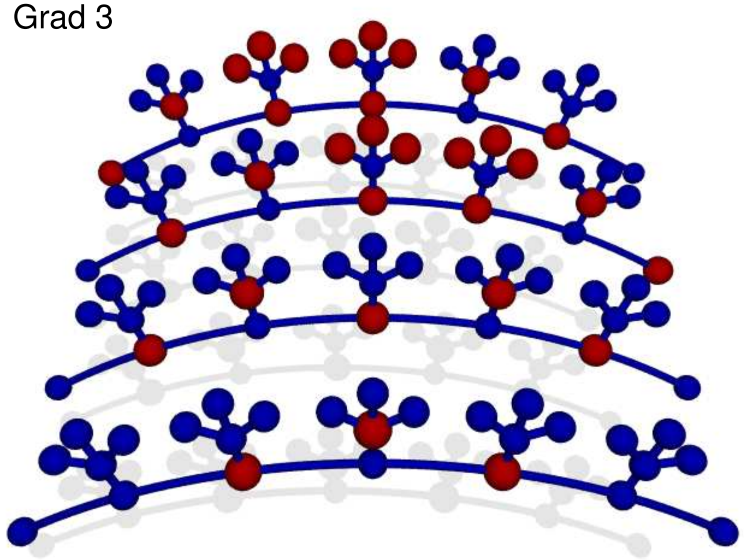
Grad 3 — Keine Brücken (a)



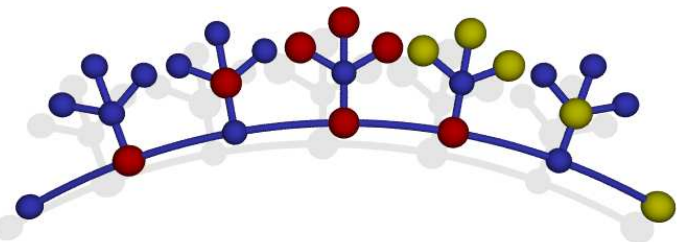
- Es gibt einen Knoten x mit Grad 3
- x hat keine Brücke
- Zwei Nachbarn haben Grad ≥ 4

Grad 3 — Keine Brücken (b)

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



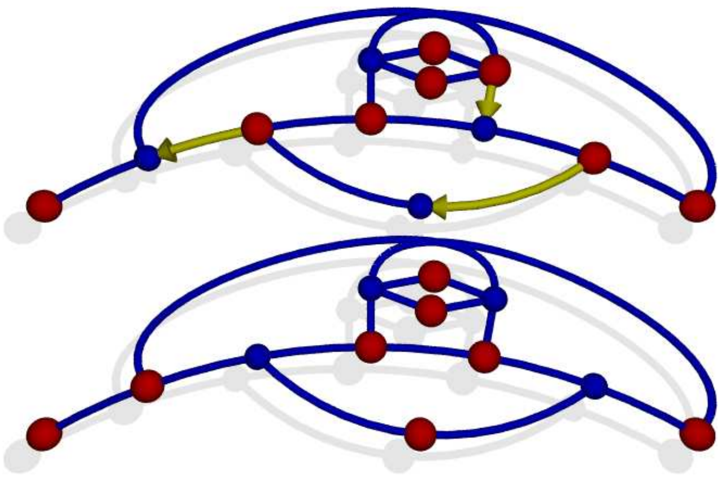
Grad 3 — Keine Brücken (b)



- Gewünscht sind 8 markierte, paarweise verschiedene Knoten
- Rote Knoten sind paarweise verschieden
- Gelbe Knoten sind paarweise verschieden
- ⇒ 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:

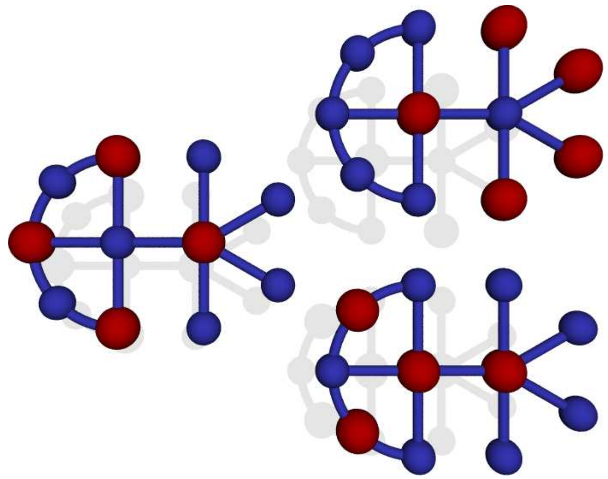


Grad 4

- 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

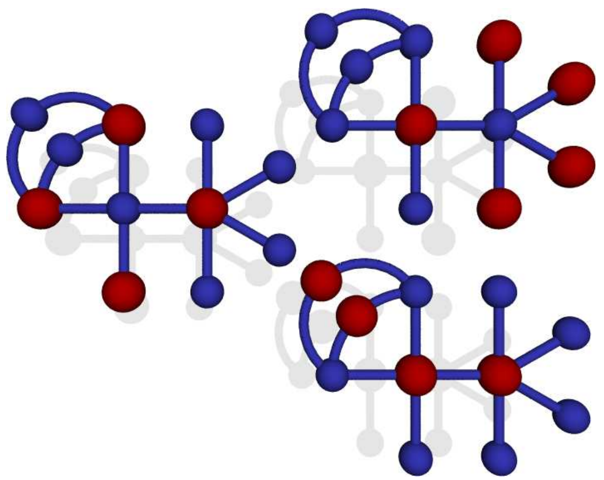
- Es gibt einen Knoten x mit Grad 4, der einen Nachbarn y mit Grad ≥ 5 hat
- 1. Unterfall: x hat ein Dreieck (weggelassen)
- 2. Unterfall: x hat mindestens 2 Brücken, auf denen y nicht liegt
- 3. Unterfall: x hat höchstens 1 Brücke, auf der y nicht liegt

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

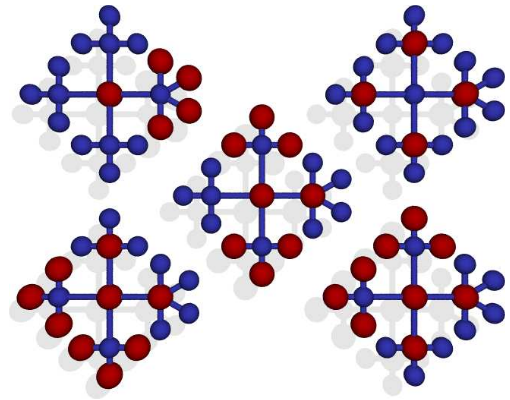
Grad 4 — Doppelbrücke



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

- $(w,k) \in L$ genau dann, wenn $f(w,k) \in L$,
- es eine Funktion $f'\colon \mathbf{N} \to \mathbf{N}$ gibt, so daß $|w'| \le f'(k)$, 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

Dies führt zu folgendem Algorithmus:

- Streiche eine Zeile die eine Untermenge einer anderen Zeile ist, bis es nicht mehr geht.
- Ebenso mit Spalten.
- Gibt es mehr als k Zeilen oder Spalten, dann gibt es keine Lösung.
- Sonst probiere alle Möglichkeiten...

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

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

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
- $x \in I_0 \Rightarrow y, y' \in C_B \Rightarrow y \in C_0, \ldots$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 C_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: Jeder 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

Welche Grapheigenschaften sind vererbbar?

- Bipartite Graphen
- Vollständige Graphen
- Planare Graphen
- Bäume
- Zusammenhängende Graphen
- Graphen mit maximalen 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.

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:

Sei $V = \{v_1, \dots, v_n\}$.

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

Am Ende ist H ein minimaler verbotener induzierter Untergraph.

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

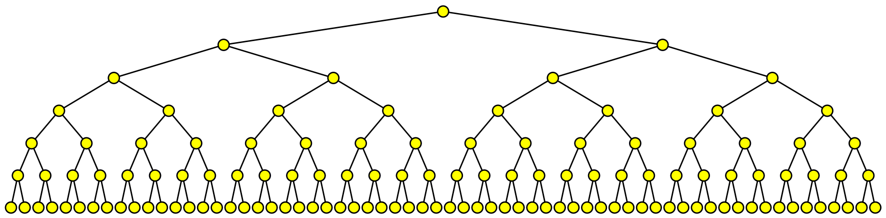
Es gibt nur N Möglichkeiten einen Knoten aus H zu löschen.

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

Gesamtlaufzeit also

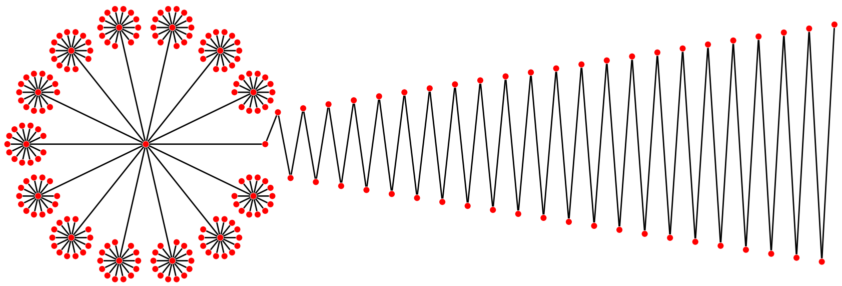
$$O(N^{i+2j+2k}|V|^{N+1}).$$

Interleaving — Suchbäume und Problemkern

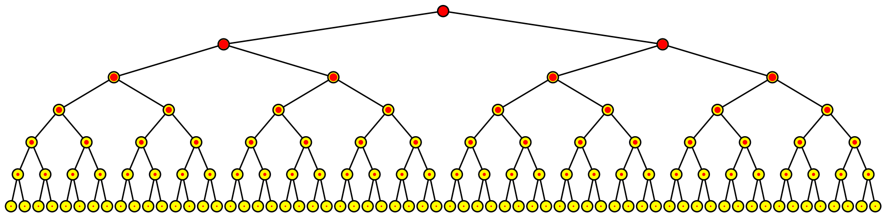


In einem Suchbaum wird der Parameter zu den Blättern hin kleiner, die Größe der Instanz aber nicht unbedingt (sie könnte sogar größer werden).

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.

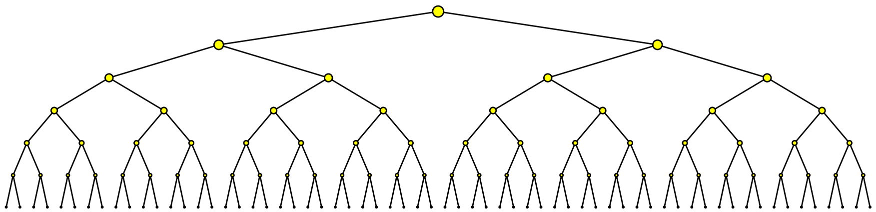


Die Größe dieses Graphen wird nicht kleiner als $n/2$ innerhalb des gesamten Suchbaums eines Vertex Cover–Algorithmus, falls wir nie eine Reduktion auf den Problemkern ausführen.



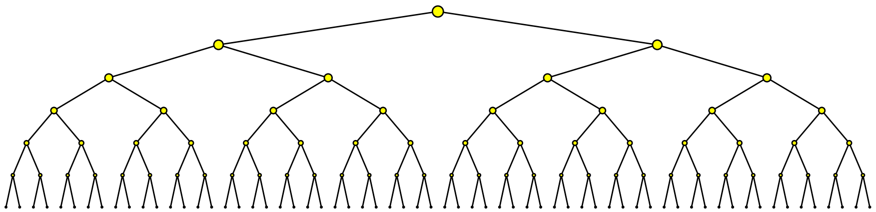
Die Größe des Parameters ist rot gezeigt. Die Rekursion endet, wenn der Parameter klein ist. In den Blättern ist der Parameter durch eine Konstante beschränkt.

Fast alle Knoten sind in der Nähe eines Blatts $\Longrightarrow$ fast alle Knoten haben einen kleinen Parameter.



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.

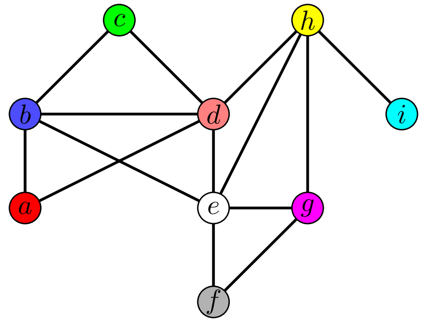
Suchbäume und dynamisches Programmmieren

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.

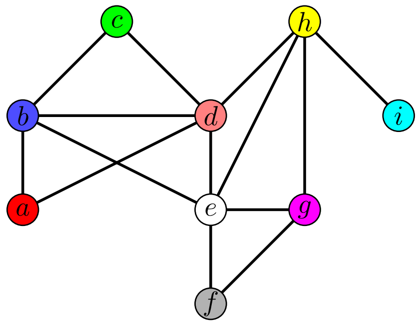
Baumzerlegungen

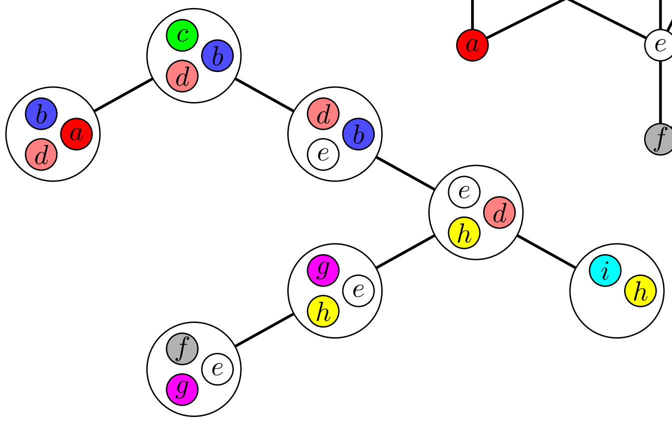
Eine Baumzerlegung eines Graphen G ist ein Baum, dessen Knoten bags genannt werden. Jeder bag ist eine Menge von Knoten aus G.



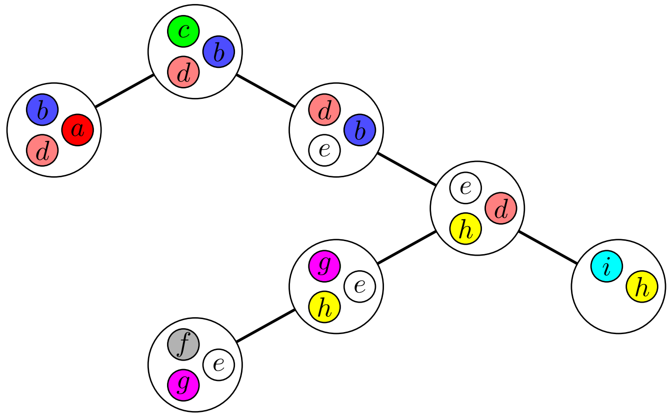
- Jeder Knoten und jede Kante von G muß in einem bag vorkommen.
- Ist ein Knoten in einem bag, dann auch in allen dazwischen

Baumzerlegungen





Baumzerlegungen und Baumweite

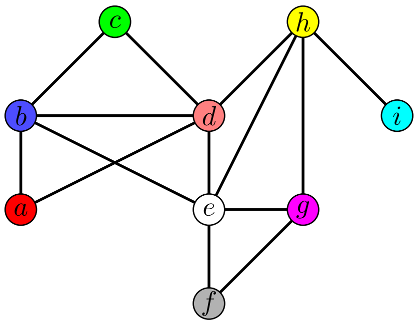


Die Weite einer Baumzerlegung ist die Größe des größten Bags minus 1.

$\Rightarrow$ Hier ist die Baumweite 2.

Baumzerlegungen und Baumweite

Alternative Definition:



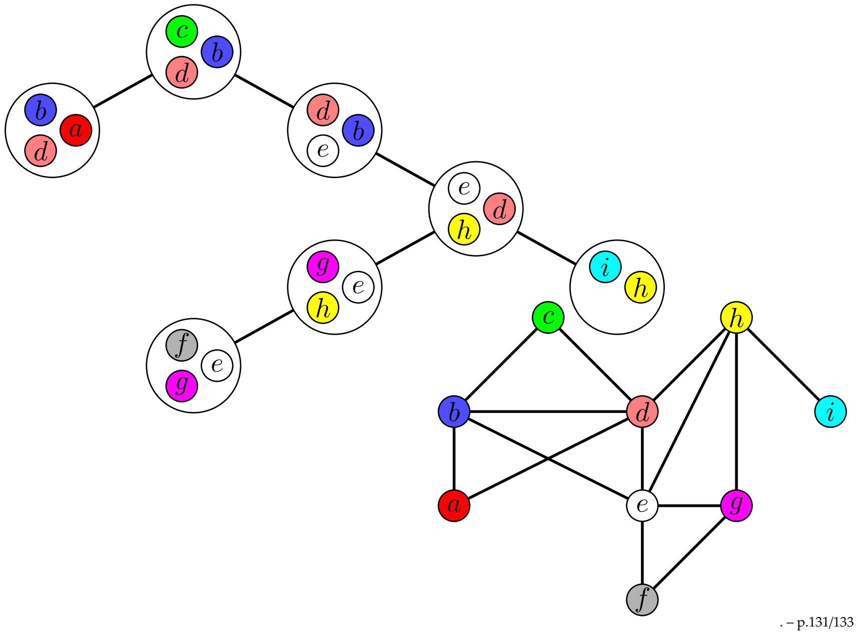
Die Baumweite von G ist die minimale Anzahl von Polizisten, die einen Räuber in G fangen können minus 1.

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.

Beispiel Vertex Cover



Allgemeines Ergebnis

Alle Probleme mit diesen Eigenschaften sind fixed-parameter tractable:

- Given a planar graph $G = (V, E)$ and a number k. Question: Does a set $S \subseteq V$ of size k exist with some property. (E.g. S is a vertex cover.)
- 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 höchsten 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