Color-Coding und Iterative Compression

Proseminar Parametrisierte Algorithmen

Hanno Jaspers

Prof. Dr. Thomas Schwentick

Technische Universität Dortmund

15. Juli 2008

Longest Path - Beispiel

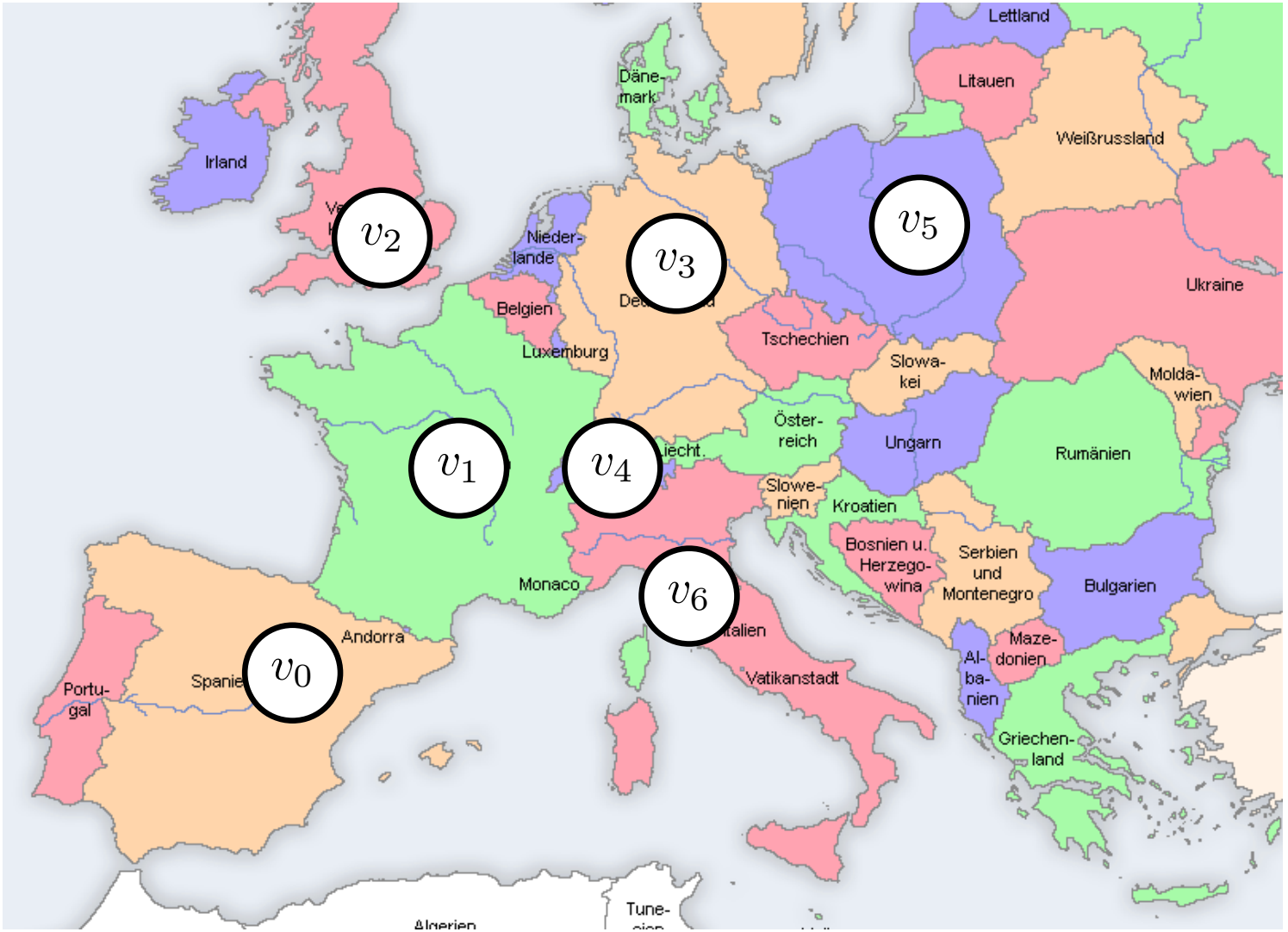
Stellt euch vor...

Longest Path - Beispiel

Stellt euch vor...

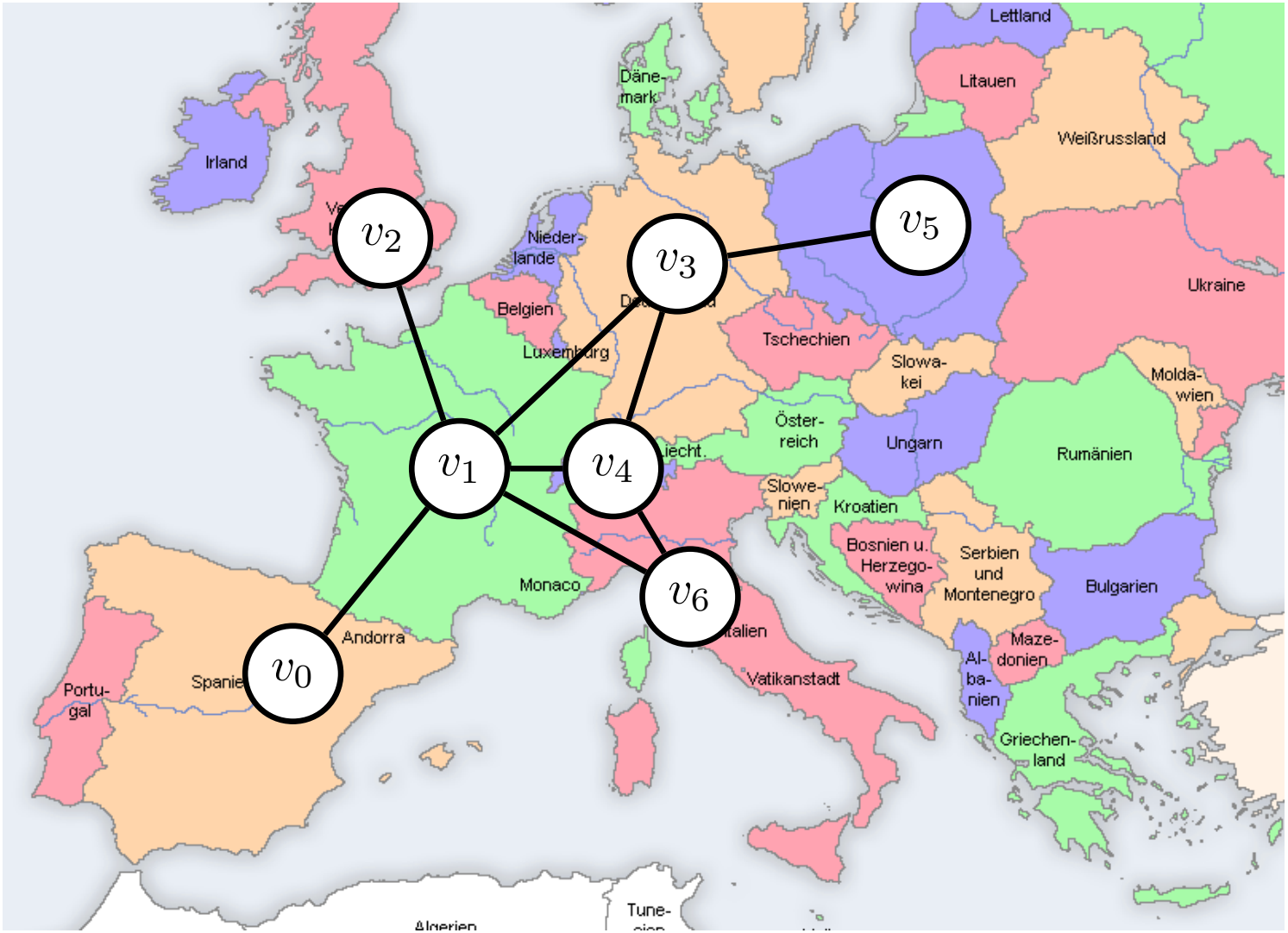
Longest Path - Beispiel

Stellt euch vor...



Longest Path - Beispiel

Stellt euch vor...



Longest Path

Gegeben: Ein Graph $G = (V, E)$.
Gesucht: Der längste einfache Pfad in G.

Longest Path

Gegeben: Ein Graph $G = (V, E)$.
Gesucht: Der längste einfache Pfad in G.

Das Problem:

- Longest Path ist NP-schwierig!

Longest Path

Gegeben: Ein Graph $G = (V, E)$.
Gesucht: Der längste einfache Pfad in G.

Das Problem:

- Longest Path ist NP-schwierig!

k-Longest Path

Gegeben: Ein Graph $G = (V, E)$ und eine positive ganze Zahl k.
Gesucht: Ein einfacher Pfad in G bestehend aus k Knoten und $k - 1$ Kanten.

Longest Path

Gegeben: Ein Graph $G = (V, E)$.
Gesucht: Der längste einfache Pfad in G.

Das Problem:

- Longest Path ist NP-schwierig!

k-Longest Path

Gegeben: Ein Graph $G = (V, E)$ und eine positive ganze Zahl k.
Gesucht: Ein einfacher Pfad in G bestehend aus k Knoten und $k - 1$ Kanten.

Dieses Problem gehört zur Klasse FPT!

Wie könnte ein nicht ganz so naiver Algorithmus aussehen?

Wie könnte ein nicht ganz so naiver Algorithmus aussehen?

Gegeben:

- Graph $G = (V, E)$ und eine natürliche Zahl k.
- Speichere in jedem Knoten eine Menge der Pfade PS, die mit diesem Knoten enden.

Wie könnte ein nicht ganz so naiver Algorithmus aussehen?

Gegeben:

- Graph $G = (V, E)$ und eine natürliche Zahl k.
- Speichere in jedem Knoten eine Menge der Pfade PS, die mit diesem Knoten enden.

Der Algorithmus folgt dem Prinzip der dynamischen Programmierung!

Algorithmus

Algorithmus

1. Starte an einem Knoten s.

Algorithmus

1. Starte an einem Knoten s.
2. Annahme:

Algorithmus

1. Starte an einem Knoten s.
2. Annahme:
   - Pfad der Länge i bereits gefunden.

Algorithmus

1. Starte an einem Knoten s.
2. Annahme:
   - Pfad der Länge i bereits gefunden.
   - Der letzte Knoten ist v, und der Pfad $P \in PS(v)$.

Algorithmus

1. Starte an einem Knoten s.
2. Annahme:
   - Pfad der Länge i bereits gefunden.
   - Der letzte Knoten ist v, und der Pfad $P \in PS(v)$.
3. Pfad der Länge $i+1$ wird gesucht:

Algorithmus

1. Starte an einem Knoten s.
2. Annahme:
   - Pfad der Länge i bereits gefunden.
   - Der letzte Knoten ist v, und der Pfad $P \in PS(v)$.
3. Pfad der Länge $i+1$ wird gesucht:
   - Sei u ein Nachbar von v, also $(u, v) \in E$

Algorithmus

1. Starte an einem Knoten s.
2. Annahme:
   - Pfad der Länge i bereits gefunden.
   - Der letzte Knoten ist v, und der Pfad $P \in PS(v)$.
3. Pfad der Länge $i+1$ wird gesucht:
   - Sei u ein Nachbar von v, also $(u, v) \in E$
   - Wenn $u \notin P$, dann:

Algorithmus

1. Starte an einem Knoten s.
2. Annahme:
   - Pfad der Länge i bereits gefunden.
   - Der letzte Knoten ist v, und der Pfad $P \in PS(v)$.
3. Pfad der Länge $i+1$ wird gesucht:
   - Sei u ein Nachbar von v, also $(u, v) \in E$
   - Wenn $u \notin P$, dann:
     - $P' := P \cup \{u\}$

Algorithmus

1. Starte an einem Knoten s.
2. Annahme:
   - Pfad der Länge i bereits gefunden.
   - Der letzte Knoten ist v, und der Pfad $P \in PS(v)$.
3. Pfad der Länge $i+1$ wird gesucht:
   - Sei u ein Nachbar von v, also $(u, v) \in E$
   - Wenn $u \notin P$, dann:
     - $P' := P \cup \{u\}$
     - $PS(u) := PS(u) \cup P'$

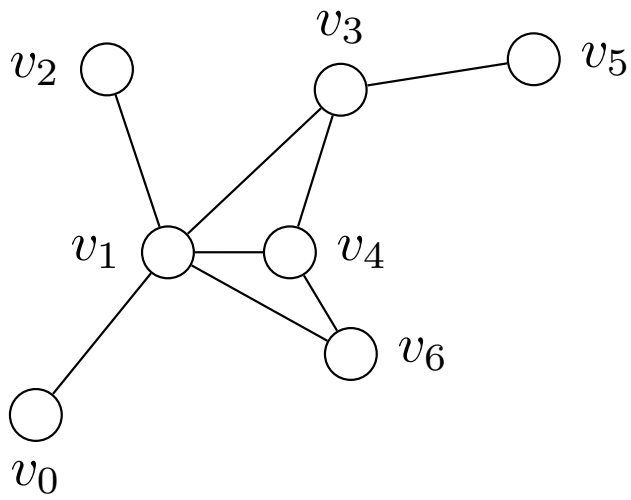
Algorithmus

1. Starte an einem Knoten s.
2. Annahme:
   - Pfad der Länge i bereits gefunden.
   - Der letzte Knoten ist v, und der Pfad $P \in PS(v)$.
3. Pfad der Länge $i+1$ wird gesucht:
   - Sei u ein Nachbar von v, also $(u, v) \in E$
   - Wenn $u \notin P$, dann:
     - $P' := P \cup \{u\}$
     - $PS(u) := PS(u) \cup P'$

$\Rightarrow$ Führe dies für jeden Knoten als Startknoten aus!

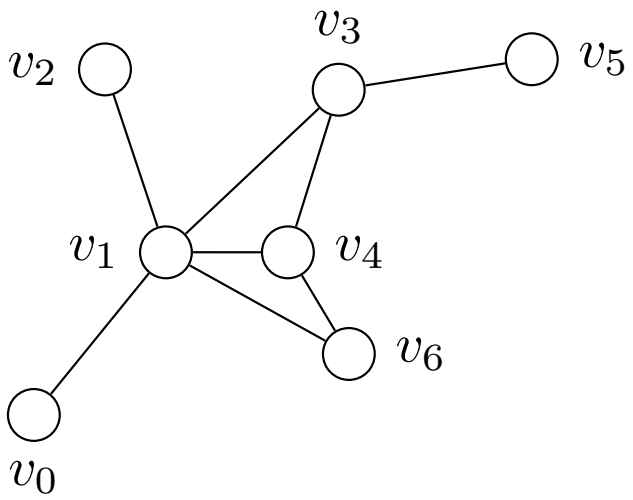
Algorithmus - Beispiel

- Gegeben: Graph $G = (V, E)$.



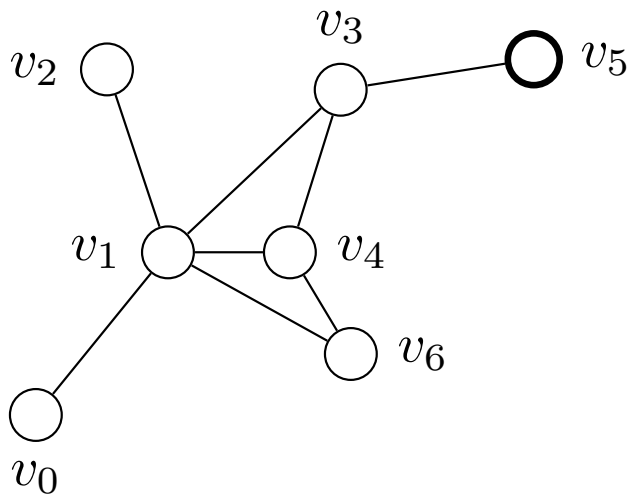
Algorithmus - Beispiel

- Gegeben: Graph $G = (V, E)$.
- Gesucht: Pfad der Länge $k = 5$.



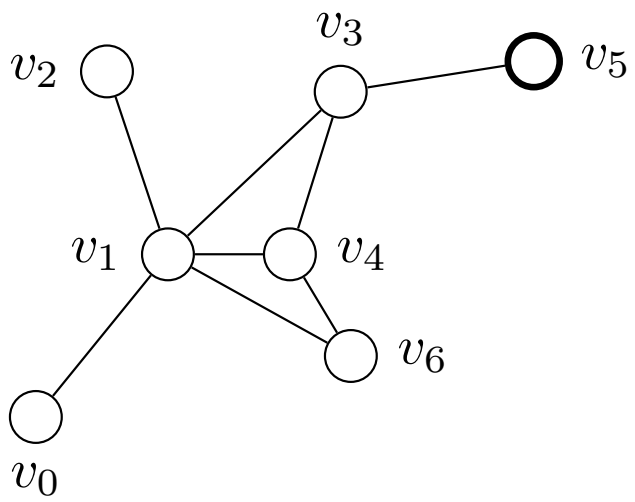
Algorithmus - Beispiel

- Startknoten: $s = v_5$.



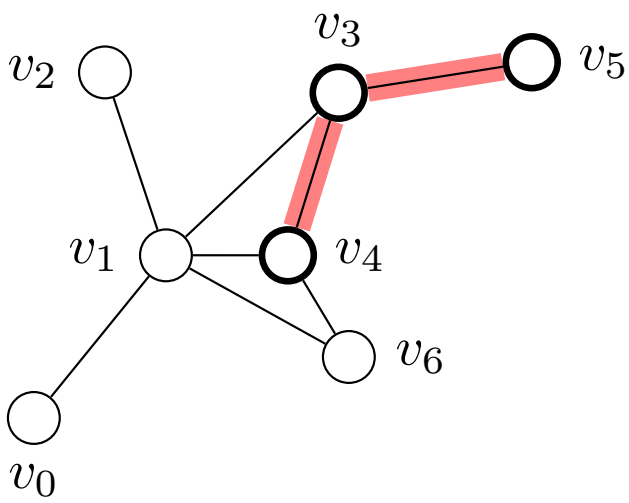
Algorithmus - Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:



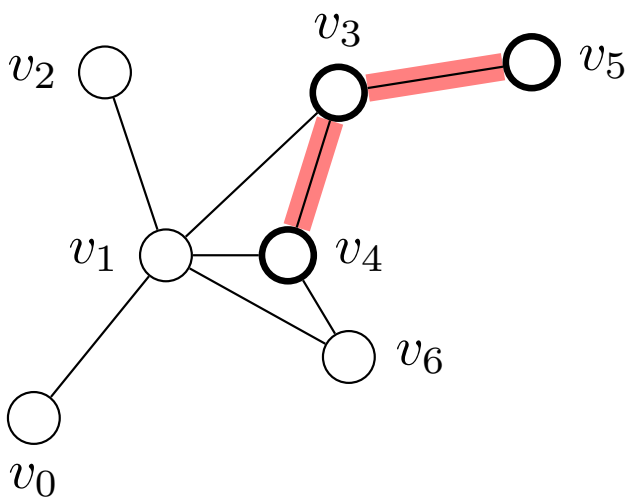
Algorithmus - Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$



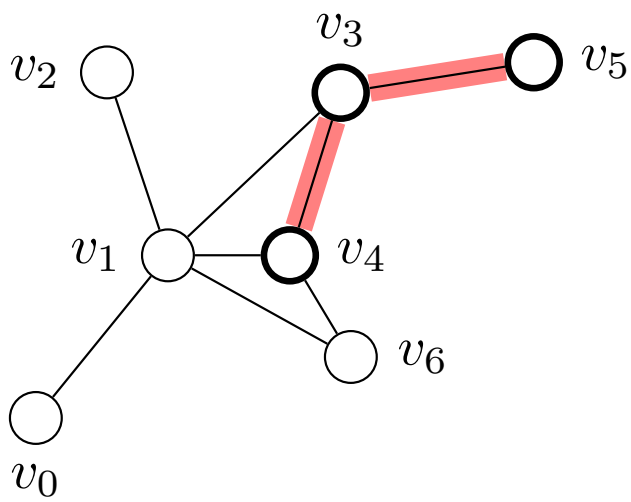
Algorithmus - Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $PS(v_4) = \{\ldots, \underbrace{\{v_5, v_3, v_4\}}_{=:P}, \ldots\}$



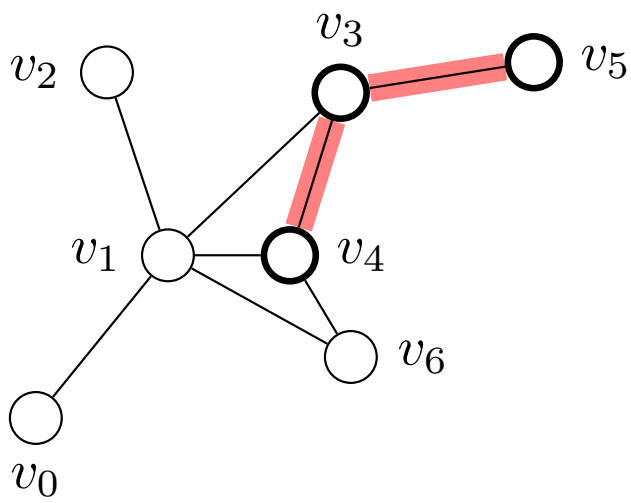
Algorithmus - Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $PS(v_4) = \{\ldots, \underbrace{\{v_5, v_3, v_4\}}_{=:P}, \ldots\}$
- 1. Nachbar von v_4: v_3



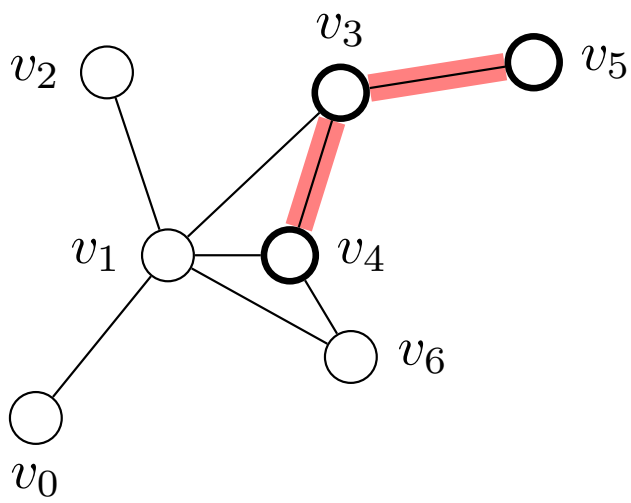
Algorithmus - Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $PS(v_4) = \{\ldots, \underbrace{\{v_5, v_3, v_4\}}_{=:P}, \ldots\}$
- 1. Nachbar von v_4: v_3
  - $v_3 \in P$



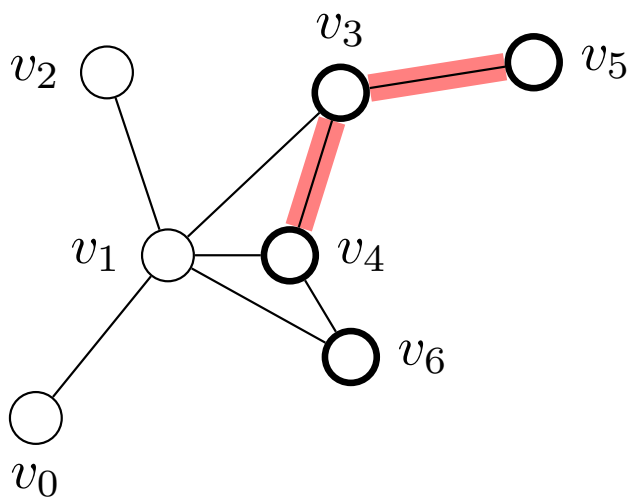
Algorithmus - Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $PS(v_4) = \{\ldots, \underbrace{\{v_5, v_3, v_4\}}_{=:P}, \ldots\}$
- 1. Nachbar von v_4: v_3
  - $v_3 \in P$
  - Nichts weiter passiert!



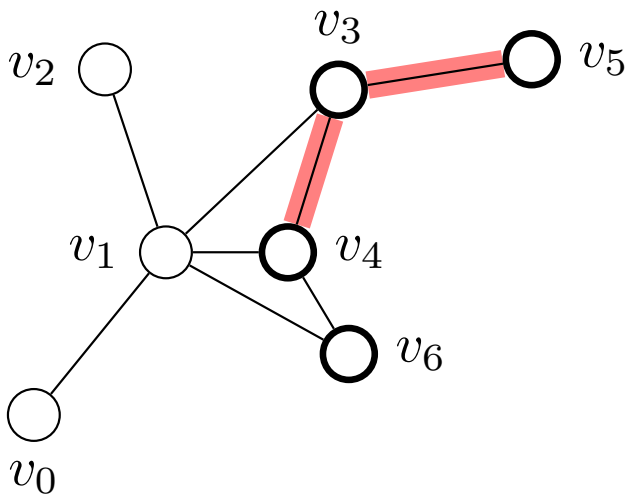
Algorithmus - Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $PS(v_4) = \{\ldots, \underbrace{\{v_5, v_3, v_4\}}_{=:P}, \ldots\}$
- 2. Nachbar von v_4: v_6



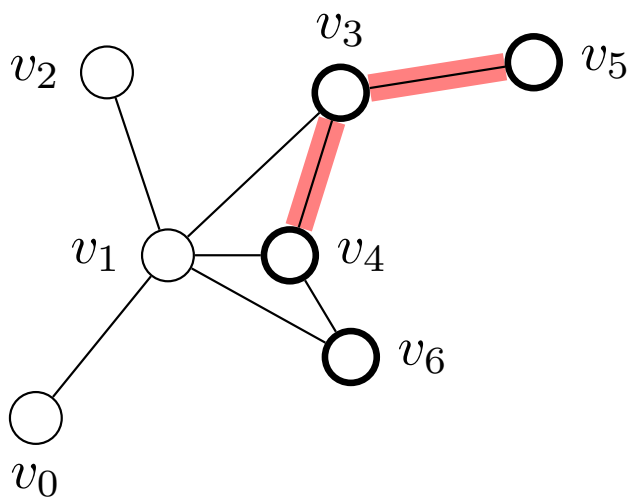
Algorithmus - Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $PS(v_4) = \{\ldots, \underbrace{\{v_5, v_3, v_4\}}_{=:P}, \ldots\}$
- 2. Nachbar von v_4: v_6
  - $v_6 \notin P$



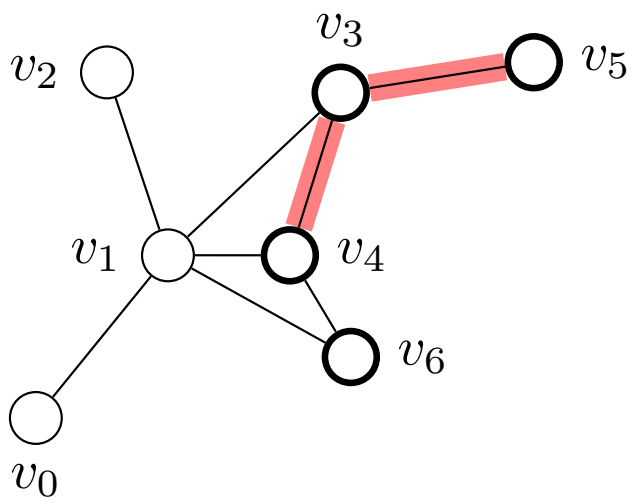
Algorithmus - Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $PS(v_4) = \{\ldots, \underbrace{\{v_5, v_3, v_4\}}_{=:P}, \ldots\}$
- 2. Nachbar von v_4: v_6
  - $v_6 \notin P$
  - $P' := \{v_5, v_3, v_4, v_6\}$



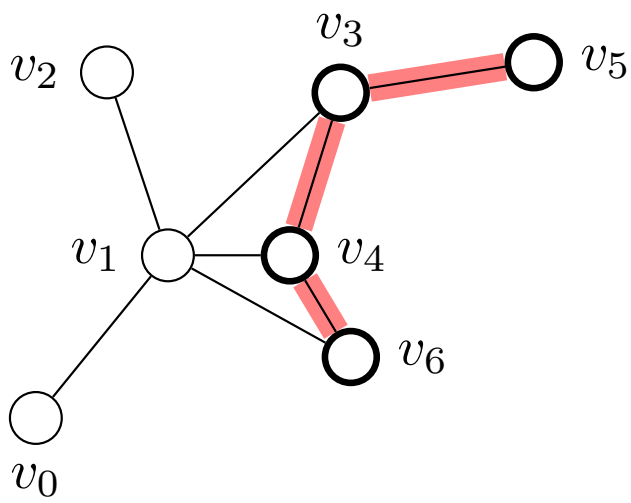
Algorithmus - Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $PS(v_4) = \{\ldots, \underbrace{\{v_5, v_3, v_4\}}_{=:P}, \ldots\}$
- 2. Nachbar von v_4: v_6
  - $v_6 \notin P$
  - $P' := \{v_5, v_3, v_4, v_6\}$
  - $PS(v_6) := \{\ldots, \{v_5, v_3, v_4, v_6\}\}$



Algorithmus - Beispiel

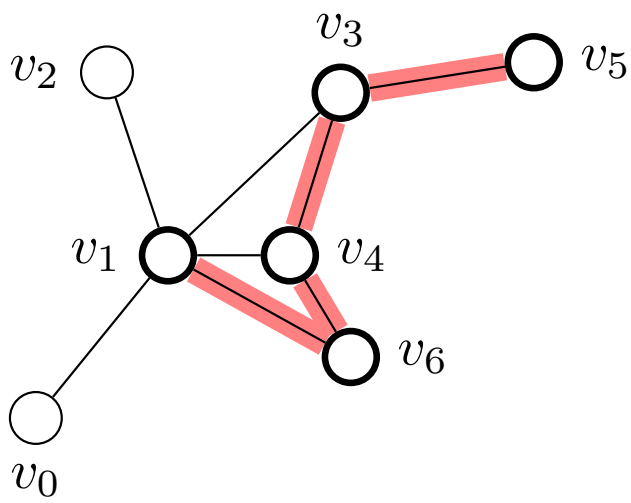
- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $PS(v_4) = \{\ldots, \underbrace{\{v_5, v_3, v_4\}}_{=:P}, \ldots\}$
- 2. Nachbar von v_4: v_6
  - $v_6 \notin P$
  - $P' := \{v_5, v_3, v_4, v_6\}$
  - $PS(v_6) := \{\ldots, \{v_5, v_3, v_4, v_6\}\}$
  - Pfad der Länge $i + 1$ gefunden!



Algorithmus - Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $PS(v_4) = \{\ldots, \underbrace{\{v_5, v_3, v_4\}}_{=:P}, \ldots\}$

Am Ende:



Laufzeit 1/3

Überlegung:

Laufzeit 1/3

Überlegung:

- Angefangen wird an jedem Knoten $v \in V$

Laufzeit 1/3

Überlegung:

- Angefangen wird an jedem Knoten $v \in V$
  - $\Rightarrow$ Pfade der Länge 1 gefunden!

Laufzeit 1/3

Überlegung:

- Angefangen wird an jedem Knoten $v \in V$
  - $\Rightarrow$ Pfade der Länge 1 gefunden!
    - Im i-ten Schritt sind alle Pfade der Länge i gefunden.

Laufzeit 1/3

Überlegung:

- Angefangen wird an jedem Knoten $v \in V$
  - $\Rightarrow$ Pfade der Länge 1 gefunden!
    - Im i-ten Schritt sind alle Pfade der Länge i gefunden.

Wie viele Pfade der Länge i kann es geben?

Laufzeit 1/3

Überlegung:

- Angefangen wird an jedem Knoten $v \in V$
  - $\Rightarrow$ Pfade der Länge 1 gefunden!
    - Im i-ten Schritt sind alle Pfade der Länge i gefunden.

Wie viele Pfade der Länge i kann es geben?

- $\binom{n}{i}$

Laufzeit 1/3

Überlegung:

- Angefangen wird an jedem Knoten $v \in V$
  - $\Rightarrow$ Pfade der Länge 1 gefunden!
    - Im i-ten Schritt sind alle Pfade der Länge i gefunden.

Wie viele Pfade der Länge i kann es geben?

- $\binom{n}{i}$
- Anzahl der Permutationen von i Elementen aus einer n-elementigen Grundmenge (ohne Beachtung der Reihenfolge)!

Laufzeit 2/3

Laufzeit:

Laufzeit 2/3

Laufzeit:

$$O\left(\sum_{i=1}^{k} i \cdot \binom{n}{i} \cdot |E|\right)$$

Laufzeit 2/3

Laufzeit:

$$O\left(\sum_{i=1}^{k} i \cdot \binom{n}{i} \cdot |E|\right)$$

i zum Prüfen, ob Knoten bereits in Knotenmenge vorhanden ist

Laufzeit 2/3

Laufzeit:

$$O\left(\sum_{i=1}^{k} i \cdot \binom{n}{i} \cdot |E|\right)$$

- i zum Prüfen, ob Knoten bereits in Knotenmenge vorhanden ist
- $\binom{n}{i}$ Anzahl der möglichen Pfade

Laufzeit 2/3

Laufzeit:

$$O\left(\sum_{i=1}^{k} i \cdot \binom{n}{i} \cdot |E|\right)$$

- i zum Prüfen, ob Knoten bereits in Knotenmenge vorhanden ist
- $\binom{n}{i}$ Anzahl der möglichen Pfade
- $|E|$ zum Finden der Nachbarn des Knotens

Laufzeit 3/3

Schätzen wir die Laufzeit nun nach oben ab:

Laufzeit 3/3

Schätzen wir die Laufzeit nun nach oben ab:

$$O\left(\sum_{i=1}^{k} i \cdot \binom{n}{i} \cdot |E|\right)$$

Laufzeit 3/3

Schätzen wir die Laufzeit nun nach oben ab:

$$O\left(\sum_{i=1}^{k} i \cdot \binom{n}{i} \cdot |E|\right) = O\left(\sum_{i=1}^{k} k \cdot \binom{n}{i} \cdot |E|\right)$$

Laufzeit 3/3

Schätzen wir die Laufzeit nun nach oben ab:

$$O\left(\sum_{i=1}^{k} i \cdot \binom{n}{i} \cdot |E|\right) = O\left(\sum_{i=1}^{k} k \cdot \binom{n}{i} \cdot |E|\right)$$

$$= O\left(k \cdot |E| \cdot \sum_{i=1}^{k} \binom{n}{i}\right)$$

Laufzeit 3/3

Schätzen wir die Laufzeit nun nach oben ab:

$$
\begin{aligned}
O\left(\sum_{i=1}^{k} i \cdot \binom{n}{i} \cdot |E|\right) &= O\left(\sum_{i=1}^{k} k \cdot \binom{n}{i} \cdot |E|\right) \\
&= O\left(k \cdot |E| \cdot \sum_{i=1}^{k} \binom{n}{i}\right) \\
&= O\left(k \cdot |E| \cdot 2^n\right)
\end{aligned}
$$

Laufzeit 3/3

Schätzen wir die Laufzeit nun nach oben ab:

$$
\begin{aligned}
O\left(\sum_{i=1}^{k} i \cdot \binom{n}{i} \cdot |E|\right) &= O\left(\sum_{i=1}^{k} k \cdot \binom{n}{i} \cdot |E|\right) \\
&= O\left(k \cdot |E| \cdot \sum_{i=1}^{k} \binom{n}{i}\right) \\
&= O\left(k \cdot |E| \cdot 2^n\right) \\
&= 2^{O(n)} \cdot |E|
\end{aligned}
$$

Die Färbung eines Graphen

Die Färbung eines Graphen

- Jedem Knoten eines Graphen wird eine von k Farben zugewiesen.

Die Färbung eines Graphen

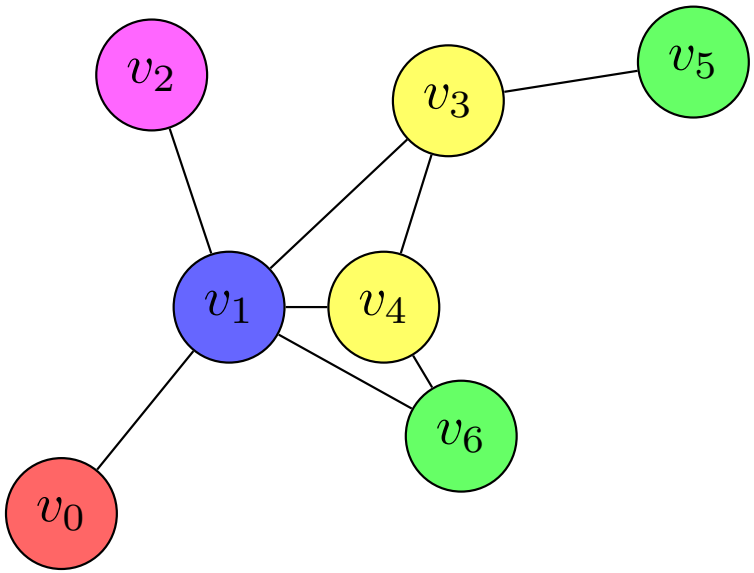
- Jedem Knoten eines Graphen wird eine von k Farben zugewiesen.
- Formal: $C : V \to \{1, \ldots, k\}$.

Die Färbung eines Graphen

- Jedem Knoten eines Graphen wird eine von k Farben zugewiesen.
- Formal: $C : V \to \{1, \ldots, k\}$.
- Hier ein paar Beispiele mit $k = 5$ Farben:

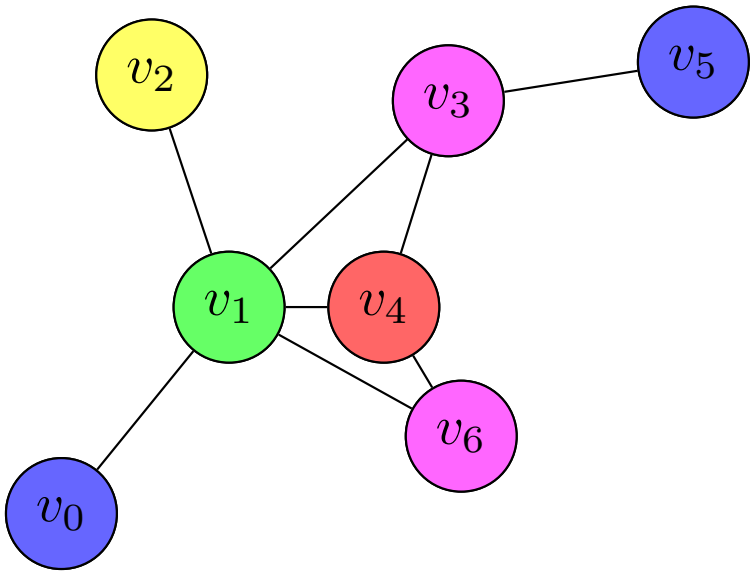
Die Färbung eines Graphen

- Jedem Knoten eines Graphen wird eine von k Farben zugewiesen.
- Formal: $C : V \to \{1, \ldots, k\}$.
- Hier ein paar Beispiele mit $k = 5$ Farben:



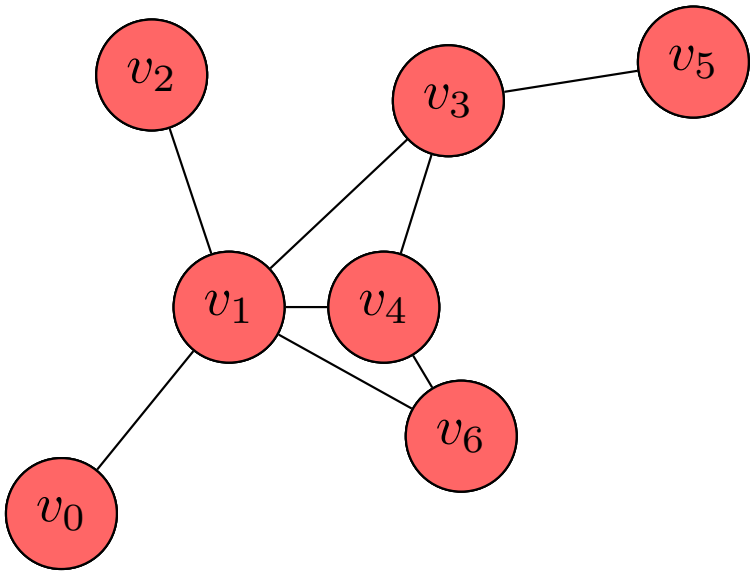
Die Färbung eines Graphen

- Jedem Knoten eines Graphen wird eine von k Farben zugewiesen.
- Formal: $C : V \to \{1, \ldots, k\}$.
- Hier ein paar Beispiele mit $k = 5$ Farben:



Die Färbung eines Graphen

- Jedem Knoten eines Graphen wird eine von k Farben zugewiesen.
- Formal: $C : V \to \{1, \ldots, k\}$.
- Hier ein paar Beispiele mit $k = 5$ Farben:



Um den Algorithmus zu verstehen, müssen folgende Dinge klar sein:

Um den Algorithmus zu verstehen, müssen folgende Dinge klar sein:

- Ein Pfad heißt bunt, wenn alle Knoten des Pfades unterschiedliche Farben haben.

Um den Algorithmus zu verstehen, müssen folgende Dinge klar sein:

- Ein Pfad heißt bunt, wenn alle Knoten des Pfades unterschiedliche Farben haben.
- Ein Pfad kann auch durch den Startknoten und die Menge der Farben der beteiligten Knoten bestimmt werden.

Um den Algorithmus zu verstehen, müssen folgende Dinge klar sein:

- Ein Pfad heißt bunt, wenn alle Knoten des Pfades unterschiedliche Farben haben.
- Ein Pfad kann auch durch den Startknoten und die Menge der Farben der beteiligten Knoten bestimmt werden.
- Analog zum naiven Algorithmus, wo ein Pfad durch den Startknoten und die am Pfad beteiligten Knoten bestimmt wurde.

Um den Algorithmus zu verstehen, müssen folgende Dinge klar sein:

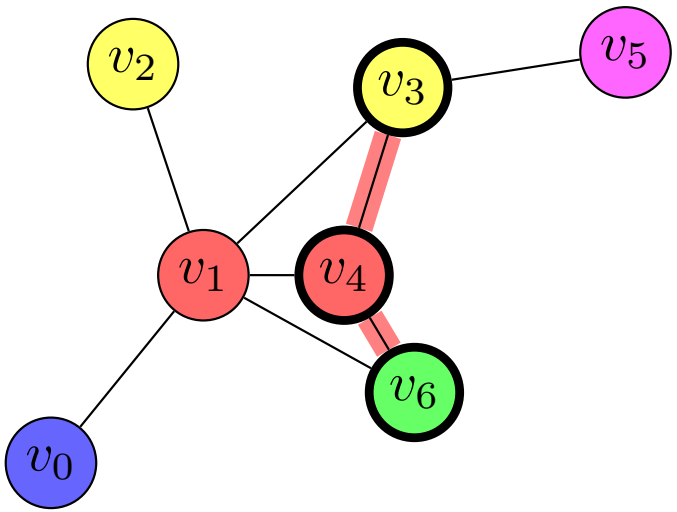
- Ein Pfad heißt bunt, wenn alle Knoten des Pfades unterschiedliche Farben haben.
- Ein Pfad kann auch durch den Startknoten und die Menge der Farben der beteiligten Knoten bestimmt werden.
- Analog zum naiven Algorithmus, wo ein Pfad durch den Startknoten und die am Pfad beteiligten Knoten bestimmt wurde.

Achtung: Der so definierte Pfad muss nicht eindeutig sein!

Um den Algorithmus zu verstehen, müssen folgende Dinge klar sein:

- Ein Pfad heißt bunt, wenn alle Knoten des Pfades unterschiedliche Farben haben.
- Ein Pfad kann auch durch den Startknoten und die Menge der Farben der beteiligten Knoten bestimmt werden.
- Analog zum naiven Algorithmus, wo ein Pfad durch den Startknoten und die am Pfad beteiligten Knoten bestimmt wurde.

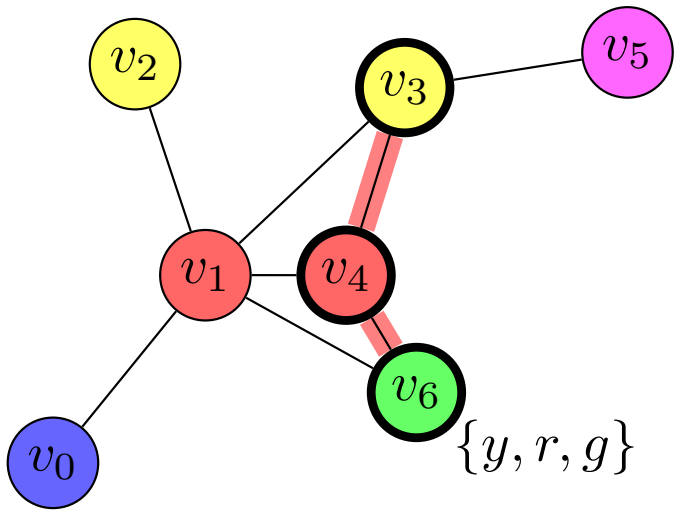
Achtung: Der so definierte Pfad muss nicht eindeutig sein!



Um den Algorithmus zu verstehen, müssen folgende Dinge klar sein:

- Ein Pfad heißt bunt, wenn alle Knoten des Pfades unterschiedliche Farben haben.
- Ein Pfad kann auch durch den Startknoten und die Menge der Farben der beteiligten Knoten bestimmt werden.
- Analog zum naiven Algorithmus, wo ein Pfad durch den Startknoten und die am Pfad beteiligten Knoten bestimmt wurde.

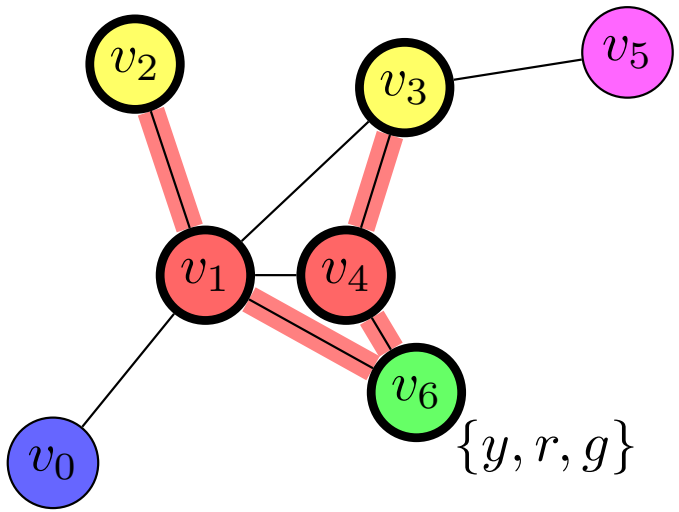
Achtung: Der so definierte Pfad muss nicht eindeutig sein!



Um den Algorithmus zu verstehen, müssen folgende Dinge klar sein:

- Ein Pfad heißt bunt, wenn alle Knoten des Pfades unterschiedliche Farben haben.
- Ein Pfad kann auch durch den Startknoten und die Menge der Farben der beteiligten Knoten bestimmt werden.
- Analog zum naiven Algorithmus, wo ein Pfad durch den Startknoten und die am Pfad beteiligten Knoten bestimmt wurde.

Achtung: Der so definierte Pfad muss nicht eindeutig sein!



Der Algorithmus

Gegeben:

- Ein Graph $G = (V, E)$ und eine Färbung $C : V \to \{1, \ldots, k\}$.
- Für jeden Knoten eine Menge von Farbmengen CS.

Der Algorithmus

Gegeben:

- Ein Graph $G = (V, E)$ und eine Färbung $C : V \to \{1, \ldots, k\}$.
- Für jeden Knoten eine Menge von Farbmengen CS.

Algorithmus:

1. Startknoten: s

Der Algorithmus

Gegeben:

- Ein Graph $G = (V, E)$ und eine Färbung $C : V \to \{1, \ldots, k\}$.
- Für jeden Knoten eine Menge von Farbmengen CS.

Algorithmus:

1. Startknoten: s
2. Annahme:

Der Algorithmus

Gegeben:

- Ein Graph $G = (V, E)$ und eine Färbung $C : V \to \{1, \ldots, k\}$.
- Für jeden Knoten eine Menge von Farbmengen CS.

Algorithmus:

1. Startknoten: s
2. Annahme:
   - Bunter Pfad der Länge i bereits gefunden.

Der Algorithmus

Gegeben:

- Ein Graph $G = (V, E)$ und eine Färbung $C : V \to \{1, \ldots, k\}$.
- Für jeden Knoten eine Menge von Farbmengen CS.

Algorithmus:

1. Startknoten: s
2. Annahme:
   - Bunter Pfad der Länge i bereits gefunden.
   - Der letzte Knoten ist v mit $F \in CS(v)$.

Der Algorithmus

Gegeben:

- Ein Graph $G = (V, E)$ und eine Färbung $C : V \to \{1, \ldots, k\}$.
- Für jeden Knoten eine Menge von Farbmengen CS.

Algorithmus:

1. Startknoten: s
2. Annahme:
   - Bunter Pfad der Länge i bereits gefunden.
   - Der letzte Knoten ist v mit $F \in CS(v)$.
3. Pfad der Länge $i + 1$ wird gesucht:

Der Algorithmus

Gegeben:

- Ein Graph $G = (V, E)$ und eine Färbung $C : V \to \{1, \ldots, k\}$.
- Für jeden Knoten eine Menge von Farbmengen CS.

Algorithmus:

1. Startknoten: s
2. Annahme:
   - Bunter Pfad der Länge i bereits gefunden.
   - Der letzte Knoten ist v mit $F \in CS(v)$.
3. Pfad der Länge $i + 1$ wird gesucht:
   - Sei u ein Nachbar von v, also $(v, u) \in E$.

Der Algorithmus

Gegeben:

- Ein Graph $G = (V, E)$ und eine Färbung $C : V \to \{1, \ldots, k\}$.
- Für jeden Knoten eine Menge von Farbmengen CS.

Algorithmus:

1. Startknoten: s
2. Annahme:
   - Bunter Pfad der Länge i bereits gefunden.
   - Der letzte Knoten ist v mit $F \in CS(v)$.
3. Pfad der Länge $i + 1$ wird gesucht:
   - Sei u ein Nachbar von v, also $(v, u) \in E$.
   - Wenn $C(u) \notin F$, dann

Der Algorithmus

Gegeben:

- Ein Graph $G = (V, E)$ und eine Färbung $C : V \to \{1, \ldots, k\}$.
- Für jeden Knoten eine Menge von Farbmengen CS.

Algorithmus:

1. Startknoten: s
2. Annahme:
   - Bunter Pfad der Länge i bereits gefunden.
   - Der letzte Knoten ist v mit $F \in CS(v)$.
3. Pfad der Länge $i + 1$ wird gesucht:
   - Sei u ein Nachbar von v, also $(v, u) \in E$.
   - Wenn $C(u) \notin F$, dann
     - $F' := F \cup C(u)$

Der Algorithmus

Gegeben:

- Ein Graph $G = (V, E)$ und eine Färbung $C : V \to \{1, \ldots, k\}$.
- Für jeden Knoten eine Menge von Farbmengen CS.

Algorithmus:

1. Startknoten: s
2. Annahme:
   - Bunter Pfad der Länge i bereits gefunden.
   - Der letzte Knoten ist v mit $F \in CS(v)$.
3. Pfad der Länge $i + 1$ wird gesucht:
   - Sei u ein Nachbar von v, also $(v, u) \in E$.
   - Wenn $C(u) \notin F$, dann
     - $F' := F \cup C(u)$
     - $CS(u) := CS(u) \cup F'$

Der Algorithmus

Gegeben:

- Ein Graph $G = (V, E)$ und eine Färbung $C : V \to \{1, \ldots, k\}$.
- Für jeden Knoten eine Menge von Farbmengen CS.

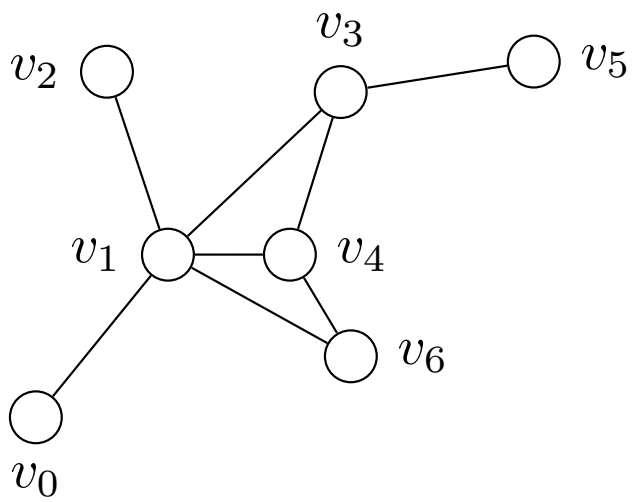
Algorithmus:

1. Startknoten: s
2. Annahme:
   - Bunter Pfad der Länge i bereits gefunden.
   - Der letzte Knoten ist v mit $F \in CS(v)$.
3. Pfad der Länge $i + 1$ wird gesucht:
   - Sei u ein Nachbar von v, also $(v, u) \in E$.
   - Wenn $C(u) \notin F$, dann
     - $F' := F \cup C(u)$
     - $CS(u) := CS(u) \cup F'$

$\Rightarrow$ Dies wird für jeden Knoten als Startknoten wiederholt!

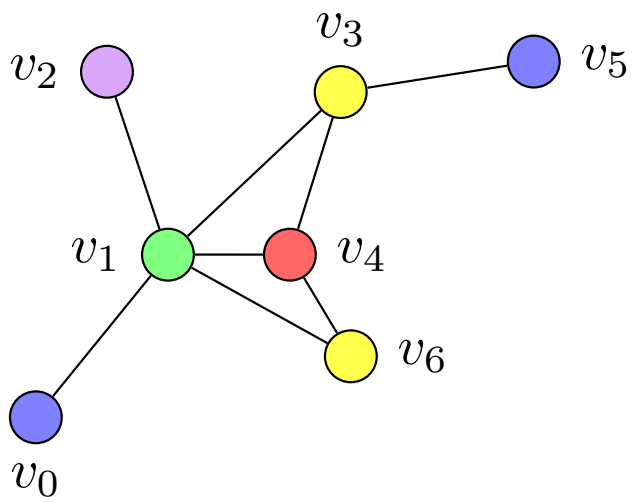
So funktioniert der Algorithmus - Ein Beispiel

- Gegeben: Graph $G = (V, E)$.



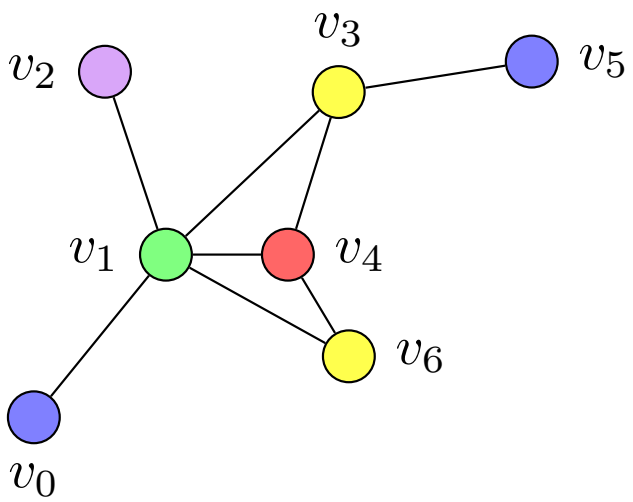
So funktioniert der Algorithmus - Ein Beispiel

- Gegeben: Graph $G = (V, E)$.
- Gegeben: Färbung $C : V \to \{b, g, r, y, p\}$.



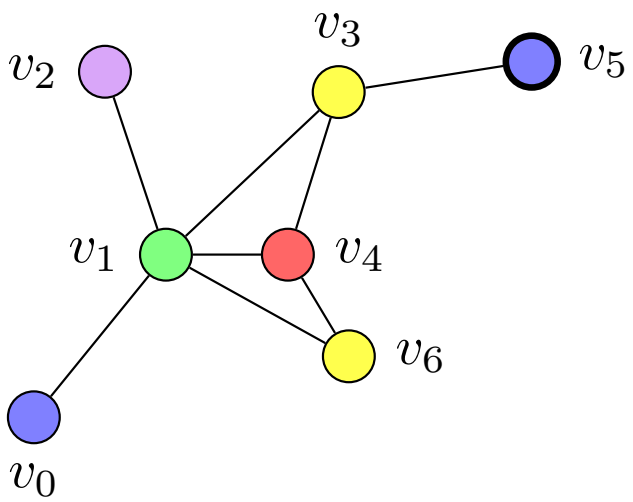
So funktioniert der Algorithmus - Ein Beispiel

- Gegeben: Graph $G = (V, E)$.
- Gegeben: Färbung $C : V \to \{b, g, r, y, p\}$.
- Gesucht: Pfad der Länge $k = 5$.



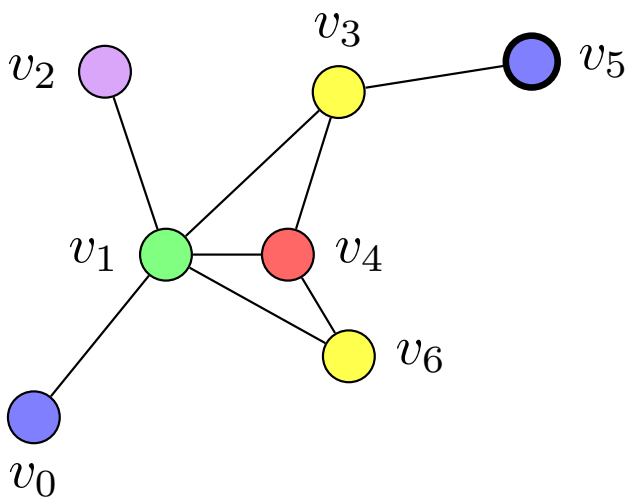
So funktioniert der Algorithmus - Ein Beispiel

- Startknoten: $s = v_5$.



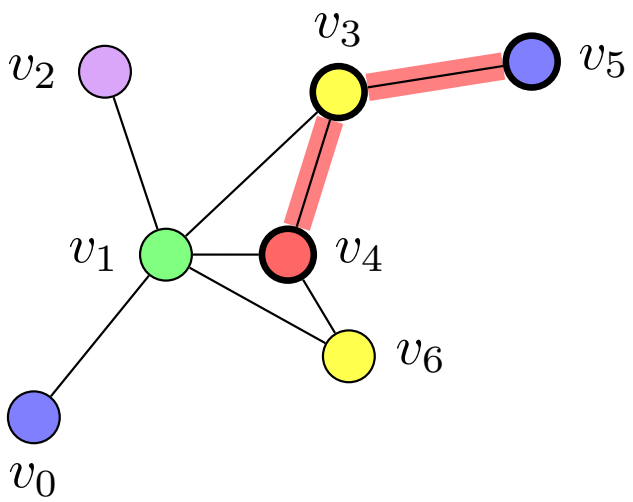
So funktioniert der Algorithmus - Ein Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:



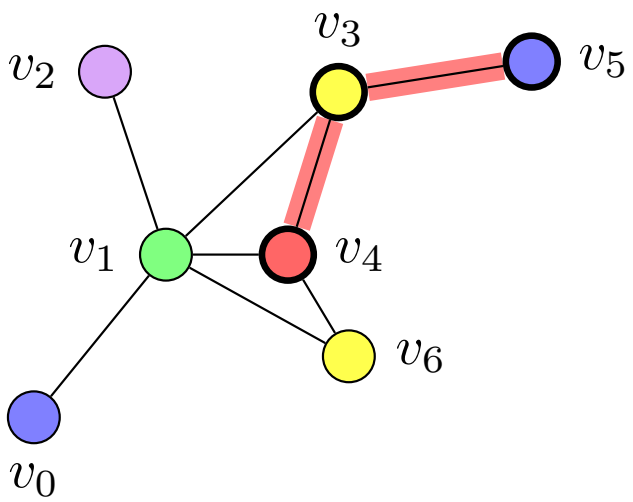
So funktioniert der Algorithmus - Ein Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$



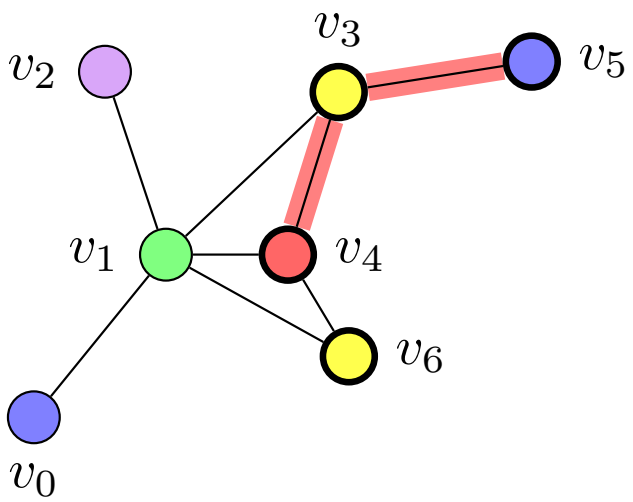
So funktioniert der Algorithmus - Ein Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $CS(v_4) = \{\ldots, \underbrace{\{b, y, r\}}_{=:F}, \ldots\}$



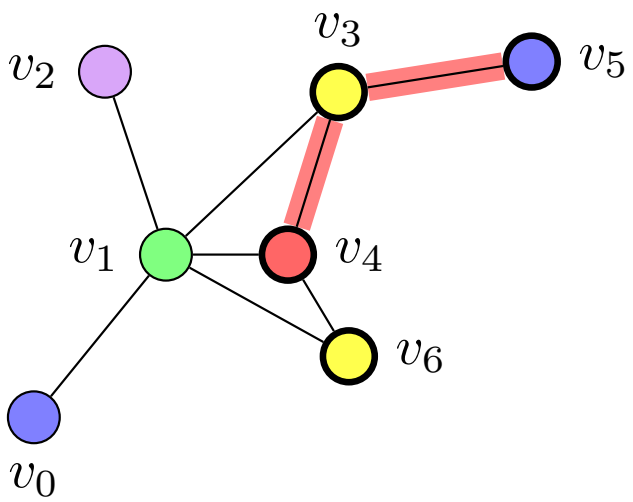
So funktioniert der Algorithmus - Ein Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $CS(v_4) = \{\ldots, \underbrace{\{b, y, r\}}_{=:F}, \ldots\}$
- 1. Nachbar von v_4: v_6



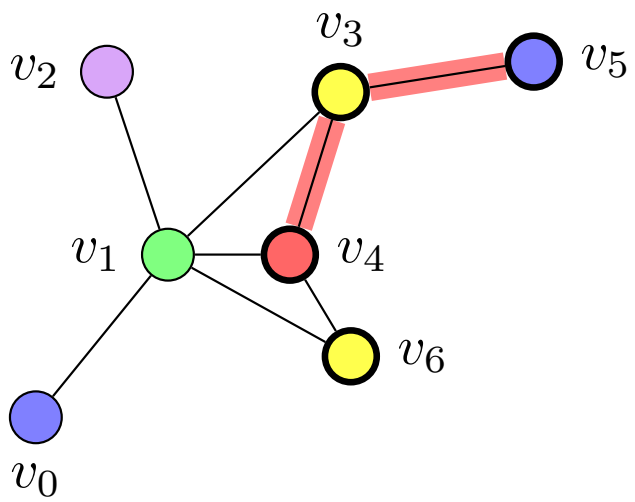
So funktioniert der Algorithmus - Ein Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \rightarrow v_3 \rightarrow v_4$
  - $CS(v_4) = \{\ldots, \underbrace{\{b, y, r\}}_{=:F}, \ldots\}$
- 1. Nachbar von v_4: v_6
  - $C(v_6) = y$ und $y \in F$



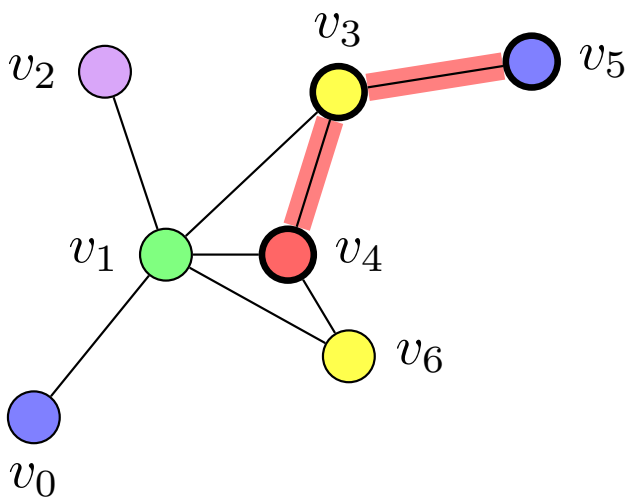
So funktioniert der Algorithmus - Ein Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \rightarrow v_3 \rightarrow v_4$
  - $CS(v_4) = \{\ldots, \underbrace{\{b, y, r\}}_{=:F}, \ldots\}$
- 1. Nachbar von v_4: v_6
  - $C(v_6) = y$ und $y \in F$
  - Nichts weiter passiert!



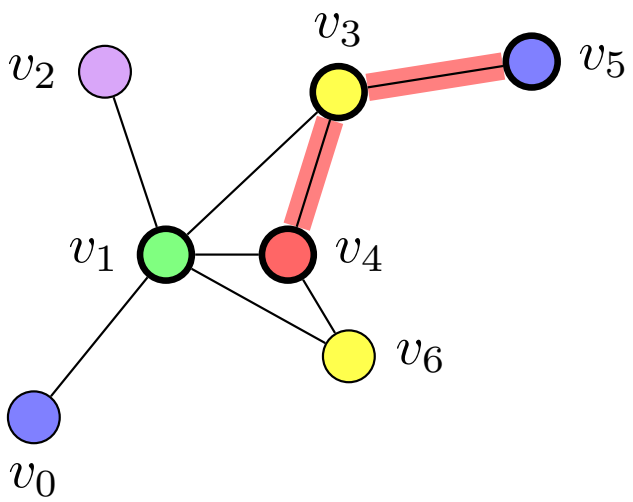
So funktioniert der Algorithmus - Ein Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $CS(v_4) = \{\ldots, \underbrace{\{b, y, r\}}_{=:F}, \ldots\}$
- 2. Nachbar von v_4: v_3
  - ...



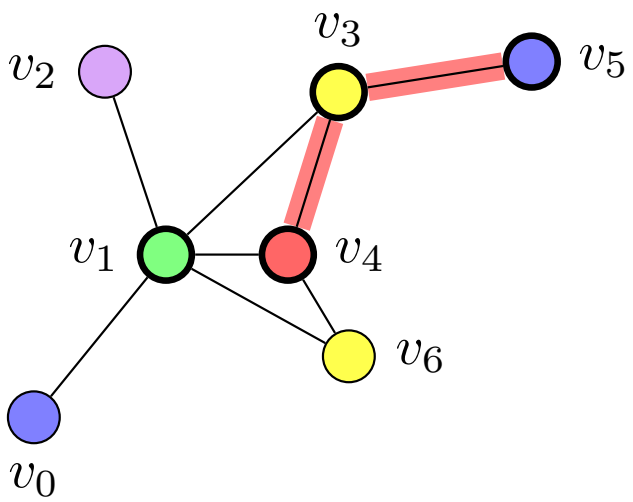
So funktioniert der Algorithmus - Ein Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $CS(v_4) = \{\ldots, \underbrace{\{b, y, r\}}_{=:F}, \ldots\}$
- 3. Nachbar von v_4: v_1



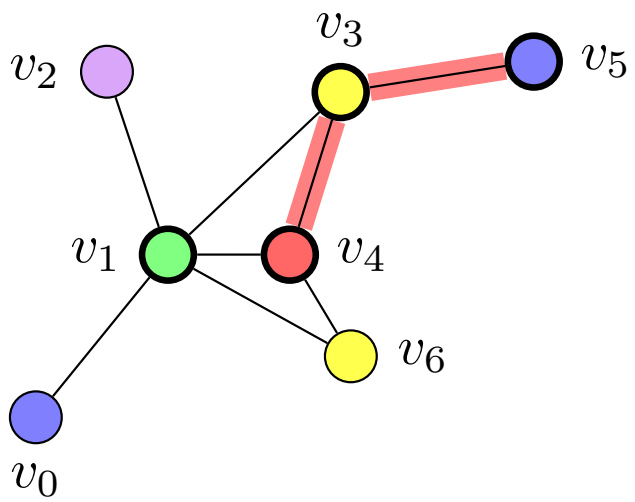
So funktioniert der Algorithmus - Ein Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \longrightarrow v_3 \longrightarrow v_4$
  - $CS(v_4) = \{\ldots, \underbrace{\{b, y, r\}}_{=:F}, \ldots\}$
- 3. Nachbar von v_4: v_1
  - $C(v_1) = g$ und $g \notin F$



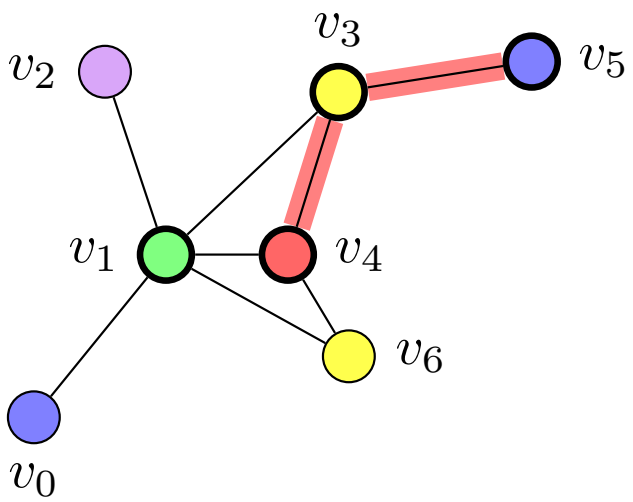
So funktioniert der Algorithmus - Ein Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \rightarrow v_3 \rightarrow v_4$
  - $CS(v_4) = \{\ldots, \underbrace{\{b, y, r\}}_{=:F}, \ldots\}$
- 3. Nachbar von v_4: v_1
  - $C(v_1) = g$ und $g \notin F$
  - $F' := \{b, y, r, g\}$



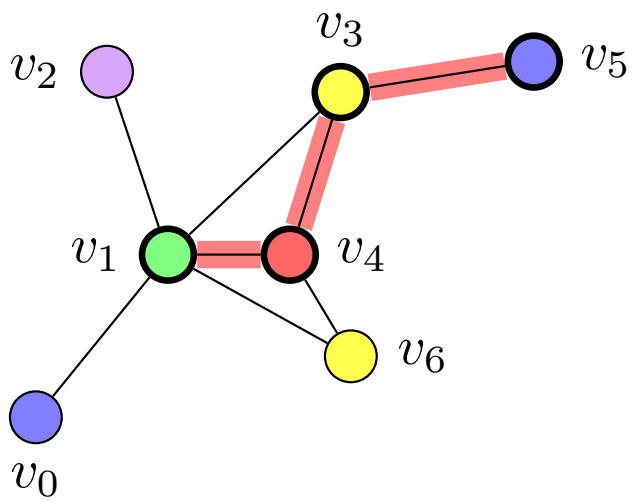
So funktioniert der Algorithmus - Ein Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \rightarrow v_3 \rightarrow v_4$
  - $CS(v_4) = \{\ldots, \underbrace{\{b, y, r\}}_{=:F}, \ldots\}$
- 3. Nachbar von v_4: v_1
  - $C(v_1) = g$ und $g \notin F$
  - $F' := \{b, y, r, g\}$
  - $CS(v_1) := \{\ldots, \{b, y, r, g\}\}$



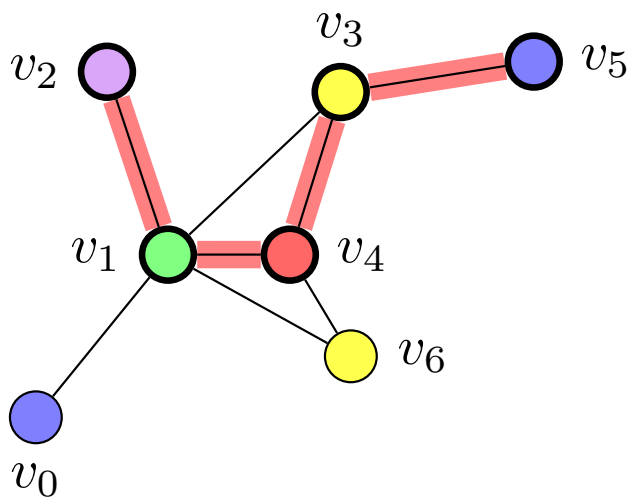
So funktioniert der Algorithmus - Ein Beispiel

- Startknoten: $s = v_5$.
- Pfad der Länge $i = 3$ bereits gefunden:
  - $v_5 \rightarrow v_3 \rightarrow v_4$
  - $CS(v_4) = \{\ldots, \underbrace{\{b, y, r\}}_{=:F}, \ldots\}$
- 3. Nachbar von v_4: v_1
  - $C(v_1) = g$ und $g \notin F$
  - $F' := \{b, y, r, g\}$
  - $CS(v_1) := \{\ldots, \{b, y, r, g\}\}$
  - Pfad der Länge $i + 1$ gefunden!



So funktioniert der Algorithmus - Ein Beispiel

Am Ende:



Laufzeit

Ganz analog zur Analyse ohne Color-Coding:

Laufzeit

Ganz analog zur Analyse ohne Color-Coding:

- Es gibt $\binom{k}{i}$ bunte Pfade der Länge i.

Laufzeit

Ganz analog zur Analyse ohne Color-Coding:

- Es gibt $\binom{k}{i}$ bunte Pfade der Länge i.
- **Laufzeit:**

Laufzeit

Ganz analog zur Analyse ohne Color-Coding:

- Es gibt $\binom{k}{i}$ bunte Pfade der Länge i.
- **Laufzeit:**

$$O\left(\sum_{i=1}^{k} i \cdot \binom{k}{i} \cdot |E|\right) = O\left(k \cdot |E| \cdot 2^k\right)$$

Vergleich mit dem naiven Algorithmus

Was genau ist der Vorteil von Color-Coding?

Vergleich mit dem naiven Algorithmus

Was genau ist der Vorteil von Color-Coding?

- Man betrachte einen Pfad der Länge $i \leq k$:

Vergleich mit dem naiven Algorithmus

Was genau ist der Vorteil von Color-Coding?

- Man betrachte einen Pfad der Länge $i \leq k$:
  - $\binom{n}{i} = O\left(n^i\right)$ mögliche Knotenmengen.

Vergleich mit dem naiven Algorithmus

Was genau ist der Vorteil von Color-Coding?

- Man betrachte einen Pfad der Länge $i \leq k$:
  - $\binom{n}{i} = O\left(n^i\right)$ mögliche Knotenmengen.
  - $\binom{k}{i} = O\left(k^i\right)$ mögliche Farbmengen.

Vergleich mit dem naiven Algorithmus

Was genau ist der Vorteil von Color-Coding?

- Man betrachte einen Pfad der Länge $i \leq k$:
  - $\binom{n}{i} = O\left(n^i\right)$ mögliche Knotenmengen.
  - $\binom{k}{i} = O\left(k^i\right)$ mögliche Farbmengen.

Das führt zu den Laufzeiten:

Vergleich mit dem naiven Algorithmus

Was genau ist der Vorteil von Color-Coding?

- Man betrachte einen Pfad der Länge $i \leq k$:
  - $\binom{n}{i} = O\left(n^i\right)$ mögliche Knotenmengen.
  - $\binom{k}{i} = O\left(k^i\right)$ mögliche Farbmengen.

Das führt zu den Laufzeiten:

ohne Color-Coding: $O\left(k \cdot 2^n \cdot |E|\right)$

Vergleich mit dem naiven Algorithmus

Was genau ist der Vorteil von Color-Coding?

- Man betrachte einen Pfad der Länge $i \leq k$:
  - $\binom{n}{i} = O\left(n^i\right)$ mögliche Knotenmengen.
  - $\binom{k}{i} = O\left(k^i\right)$ mögliche Farbmengen.

Das führt zu den Laufzeiten:

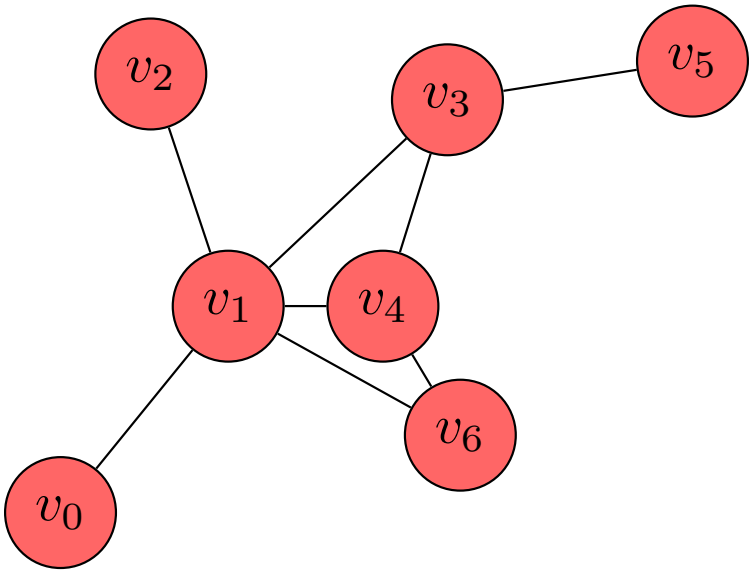
ohne Color-Coding: $O\left(k \cdot 2^n \cdot |E|\right)$

mit Color-Coding: $O\left(k \cdot 2^k \cdot |E|\right)$

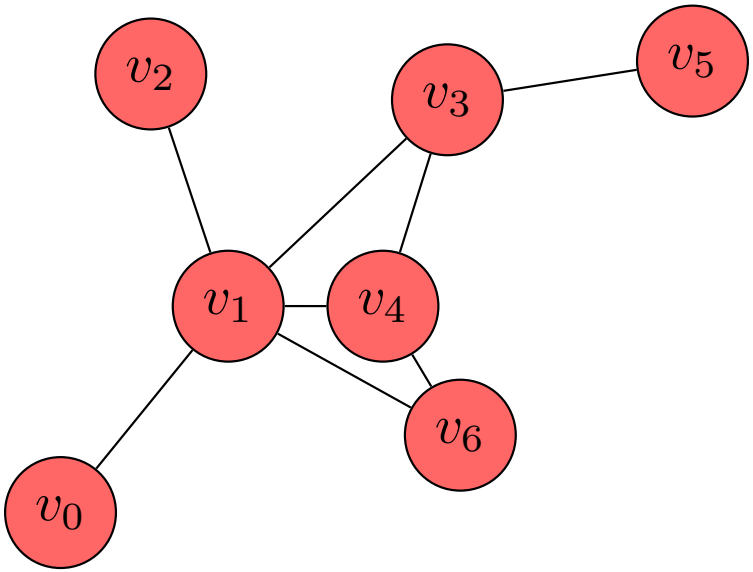
Problem:

Problem: Der Algorithmus war für eine bestimmte Färbung des Graphen.

Problem: Der Algorithmus war für eine bestimmte Färbung des Graphen.



Problem: Der Algorithmus war für eine bestimmte Färbung des Graphen.



Bei einer ungünstigen Färbung wird kein Pfad der Länge k gefunden!

Randomisierter Algorithmus

Mit welcher Wahrscheinlichkeit ist ein einfacher Pfad bunt?

Randomisierter Algorithmus

Mit welcher Wahrscheinlichkeit ist ein einfacher Pfad bunt?

$$
\begin{aligned}
P &= \frac{k!}{k^k} \\
&\approx \sqrt{2\pi k} \cdot \frac{k^k}{k^k} \cdot e^{-k} \\
&\geq e^{-k} && = \frac{1}{2^{O(k)}}
\end{aligned}
$$

Stirling-Formel

$$k! \approx \sqrt{2\pi k} \left(\frac{n}{e}\right)^k$$

Randomisierter Algorithmus

Mit welcher Wahrscheinlichkeit ist ein einfacher Pfad bunt?

$$
\begin{aligned}
P &= \frac{k!}{k^k} \\
&\approx \sqrt{2\pi k} \cdot \frac{k^k}{k^k} \cdot e^{-k} \\
&\geq e^{-k} &= \frac{1}{2^{O(k)}}
\end{aligned}
$$

Stirling-Formel

$$k! \approx \sqrt{2\pi k} \left(\frac{n}{e}\right)^k$$

Randomisierter Algorithmus für k-Longest Path

Wiederhole den Algorithmus mit Color-Coding $e^k = 2^{O(k)}$ Mal mit jeweils neu erzeugten Färbungen.

Deterministischer Algorithmus

Wie können wir einen deterministischen Algorithmus konstruieren?

Deterministischer Algorithmus

Wie können wir einen deterministischen Algorithmus konstruieren?

- Wir brauchen eine Liste von Färbungen.

Deterministischer Algorithmus

Wie können wir einen deterministischen Algorithmus konstruieren?

- Wir brauchen eine Liste von Färbungen.
- Jeder potentielle k-lange Pfad muss durch mind. eine dieser Färbungen bunt werden.

Deterministischer Algorithmus

Wie können wir einen deterministischen Algorithmus konstruieren?

- Wir brauchen eine Liste von Färbungen.
- Jeder potentielle k-lange Pfad muss durch mind. eine dieser Färbungen bunt werden.
  - $\Rightarrow$ Für jede k-große Knotenteilmenge V' muss es mind. eine Färbung geben, sodass jeder Knoten $v \in V'$ eine eindeutige Farbe hat.

Definition: Familie k-perfekter Hash-Funktionen

[1] [6, Schmidt, Siegal - The Spatial Complexity of Oblivious k-Probe Hash Functions]
[2] [1, Alon, Yuster und Zwick - Color-coding]

Definition: Familie k-perfekter Hash-Funktionen

Eine k-perfekte Familie von Hash-Funktionen ist eine Familie H von Funktionen von $\{1, \ldots, n\}$ auf $\{1, \ldots, k\}$, sodass es für jedes $S \subseteq \{1, \ldots, n\}$ mit $|S| = k$ eine Funktion $h \in H$ gibt, die eineindeutig auf S ist.

[1] [6, Schmidt, Siegal - The Spatial Complexity of Oblivious k-Probe Hash Functions]
[2] [1, Alon, Yuster und Zwick - Color-coding]

Definition: Familie k-perfekter Hash-Funktionen

Eine k-perfekte Familie von Hash-Funktionen ist eine Familie H von Funktionen von $\{1, \ldots, n\}$ auf $\{1, \ldots, k\}$, sodass es für jedes $S \subseteq \{1, \ldots, n\}$ mit $|S| = k$ eine Funktion $h \in H$ gibt, die eineindeutig auf S ist.

Es ist möglich, eine solche Familie bestehend aus $2^{O(k)} \cdot log(n)$ Hash-Funktionen zu konstruieren. [1] [2]

[1] [6, Schmidt, Siegal - The Spatial Complexity of Oblivious k-Probe Hash Functions]
[2] [1, Alon, Yuster und Zwick - Color-coding]

Definition: Familie k-perfekter Hash-Funktionen

Eine k-perfekte Familie von Hash-Funktionen ist eine Familie H von Funktionen von $\{1, \ldots, n\}$ auf $\{1, \ldots, k\}$, sodass es für jedes $S \subseteq \{1, \ldots, n\}$ mit $|S| = k$ eine Funktion $h \in H$ gibt, die eineindeutig auf S ist.

Es ist möglich, eine solche Familie bestehend aus $2^{O(k)} \cdot log(n)$ Hash-Funktionen zu konstruieren. [1] [2]

$\Rightarrow$ k-Longest-Path kann deterministisch in $2^{O(k)} \cdot log|V| \cdot |E|$ gelöst werden.

[1] [6, Schmidt, Siegal - The Spatial Complexity of Oblivious k-Probe Hash Functions]
[2] [1, Alon, Yuster und Zwick - Color-coding]

Einschränkung

Color-Coding sieht vielversprechend aus, aber ...

Einschränkung

Color-Coding sieht vielversprechend aus, aber ...

- es ist noch kein praktisch relevanter Einsatz bekannt.

Einschränkung

Color-Coding sieht vielversprechend aus, aber ...

- es ist noch kein praktisch relevanter Einsatz bekannt.
- viele andere Subgraph-Isomorphie Probleme lassen sich nicht per Color-Coding lösen:

Einschränkung

Color-Coding sieht vielversprechend aus, aber ...

- es ist noch kein praktisch relevanter Einsatz bekannt.
- viele andere Subgraph-Isomorphie Probleme lassen sich nicht per Color-Coding lösen:

Clique:

Es müssten Kanten vom neuen Knoten zu allen bereits gewählten Knoten gesucht werden.

Einschränkung

Color-Coding sieht vielversprechend aus, aber ...

- es ist noch kein praktisch relevanter Einsatz bekannt.
- viele andere Subgraph-Isomorphie Probleme lassen sich nicht per Color-Coding lösen:

Clique:

Es müssten Kanten vom neuen Knoten zu allen bereits gewählten Knoten gesucht werden.

- Nur die Farbe der Knoten reicht nicht aus.

Einschränkung

Color-Coding sieht vielversprechend aus, aber ...

- es ist noch kein praktisch relevanter Einsatz bekannt.
- viele andere Subgraph-Isomorphie Probleme lassen sich nicht per Color-Coding lösen:

Clique:

Es müssten Kanten vom neuen Knoten zu allen bereits gewählten Knoten gesucht werden.

- Nur die Farbe der Knoten reicht nicht aus.
- Anscheinend unmöglich, von $\binom{n}{i}$ auf $\binom{k}{i}$ zu kommen.

Zusammenfassung

Color-Coding hilft, parametrisierte Algorithmen für Subgraph-Isomorphie Probleme (z.B. Longest Path) zu entwickeln.

Zusammenfassung

Color-Coding hilft, parametrisierte Algorithmen für Subgraph-Isomorphie Probleme (z.B. Longest Path) zu entwickeln.

- Man kann die kombinatorische Explosion von $\binom{n}{i}$ auf $\binom{k}{i}$ senken.

Zusammenfassung

Color-Coding hilft, parametrisierte Algorithmen für Subgraph-Isomorphie Probleme (z.B. Longest Path) zu entwickeln.

- Man kann die kombinatorische Explosion von $\binom{n}{i}$ auf $\binom{k}{i}$ senken.

Longest Path:

Zusammenfassung

Color-Coding hilft, parametrisierte Algorithmen für Subgraph-Isomorphie Probleme (z.B. Longest Path) zu entwickeln.

- Man kann die kombinatorische Explosion von $\binom{n}{i}$ auf $\binom{k}{i}$ senken.

Longest Path:

naiver Algorithmus: $2^{O(n)} \cdot |E|$

Zusammenfassung

Color-Coding hilft, parametrisierte Algorithmen für Subgraph-Isomorphie Probleme (z.B. Longest Path) zu entwickeln.

- Man kann die kombinatorische Explosion von $\binom{n}{i}$ auf $\binom{k}{i}$ senken.

Longest Path:

naiver Algorithmus:	$2^{O(n)} \cdot \lvert E \rvert$
rand. Algorithmus mit CC:	$2^{O(k)} \cdot \lvert E \rvert$

Zusammenfassung

Color-Coding hilft, parametrisierte Algorithmen für Subgraph-Isomorphie Probleme (z.B. Longest Path) zu entwickeln.

- Man kann die kombinatorische Explosion von $\binom{n}{i}$ auf $\binom{k}{i}$ senken.

Longest Path:

naiver Algorithmus:	$2^{O(n)} \cdot \|E\|$
rand. Algorithmus mit CC:	$2^{O(k)} \cdot \|E\|$
determ. Algorithmus mit CC:	$2^{O(k)} \cdot \|E\| \cdot log\|V\|$

Allgemeines

Iterative Compression . . .

[3] [5, Reed et al. - Finding odd cycle transversals]

Allgemeines

Iterative Compression ...

... wurde erst 2004 veröffentlicht. [3]

[3] [5, Reed et al. - Finding odd cycle transversals]

Allgemeines

Iterative Compression ...

- ... wurde erst 2004 veröffentlicht. [3]
- ... ist für Minimierungsprobleme gedacht.

[3] [5, Reed et al. - Finding odd cycle transversals]

Allgemeines

Iterative Compression ...

- ... wurde erst 2004 veröffentlicht. [3]
- ... ist für Minimierungsprobleme gedacht.
- ... hat bereits sehr gute Ergebnisse bei parametrisierten Algorithmen hervorgebracht.

[3] [5, Reed et al. - Finding odd cycle transversals]

Zentrale Idee:

[a][2, H. Moser - Iterative Compression for Solving Hard Network Problems]

Zentrale Idee:

Berechne eine Lösung für eine Probleminstanz mit Hilfe der Lösung einer kleineren Instanz. [a]

[a] [2, H. Moser - Iterative Compression for Solving Hard Network Problems]

Zentrale Idee:

Berechne eine Lösung für eine Probleminstanz mit Hilfe der Lösung einer kleineren Instanz. [a]

[a] [2, H. Moser - Iterative Compression for Solving Hard Network Problems]

Etwas spezieller:

Gib einen parametrisierten Algorithmus an, der für eine $(k+1)$-große Lösung entweder die k-große Lösung berechnet oder zeigt, dass es eine solche nicht gibt.

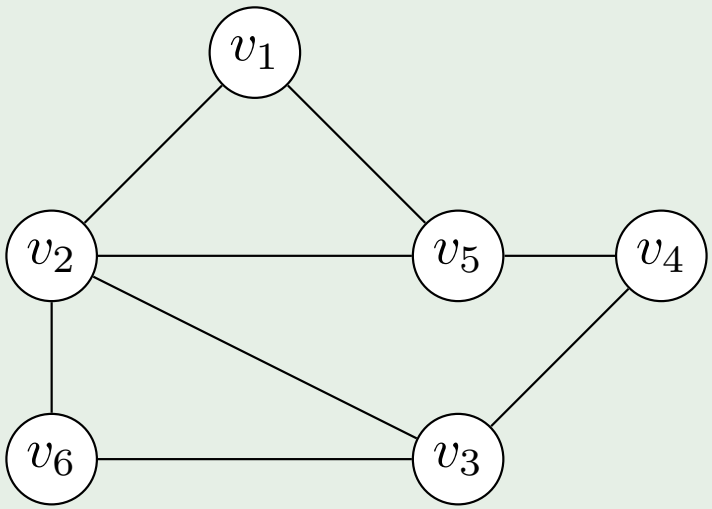
Definition: Vertex Cover

Eingabe: Ein Graph $G = (V, E)$ und eine natürliche Zahl k.
Aufgabe: Finde eine Teilmenge $C \subseteq V$ mit maximal k Knoten, sodass mindestens ein Endpunkt jeder Kante aus E in C ist.

Definition: Vertex Cover

Eingabe: Ein Graph $G = (V, E)$ und eine natürliche Zahl k.
Aufgabe: Finde eine Teilmenge $C \subseteq V$ mit maximal k Knoten, sodass mindestens ein Endpunkt jeder Kante aus E in C ist.

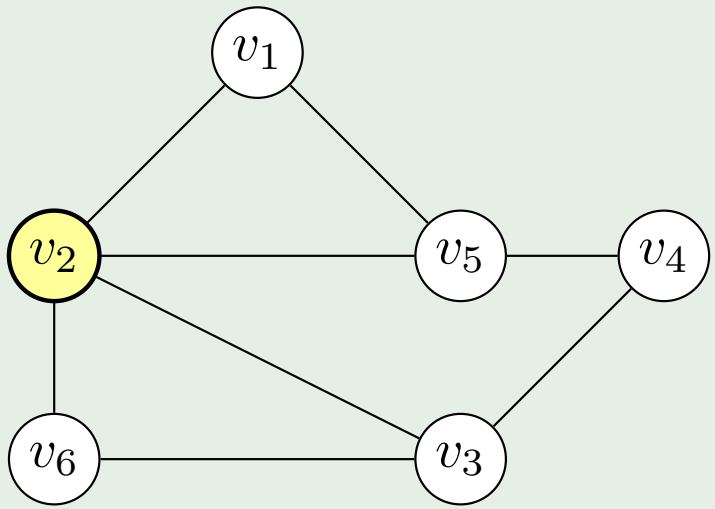
Beispiel



Definition: Vertex Cover

Eingabe: Ein Graph $G = (V, E)$ und eine natürliche Zahl k.
Aufgabe: Finde eine Teilmenge $C \subseteq V$ mit maximal k Knoten, sodass mindestens ein Endpunkt jeder Kante aus E in C ist.

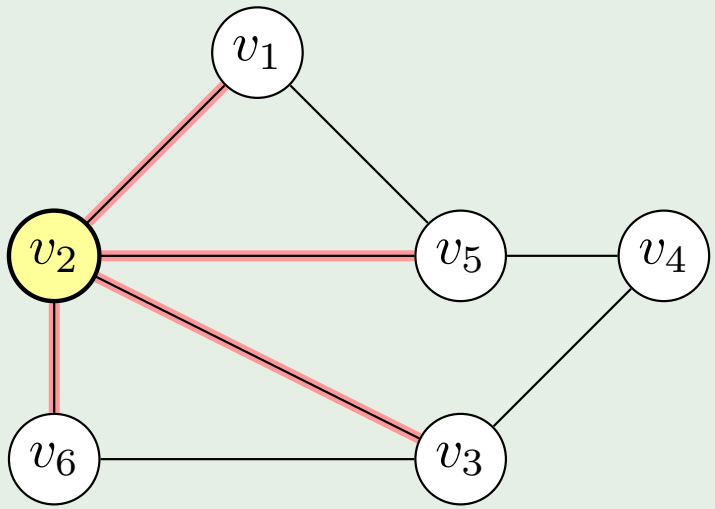
Beispiel



Definition: Vertex Cover

Eingabe: Ein Graph $G = (V, E)$ und eine natürliche Zahl k.
Aufgabe: Finde eine Teilmenge $C \subseteq V$ mit maximal k Knoten, sodass mindestens ein Endpunkt jeder Kante aus E in C ist.

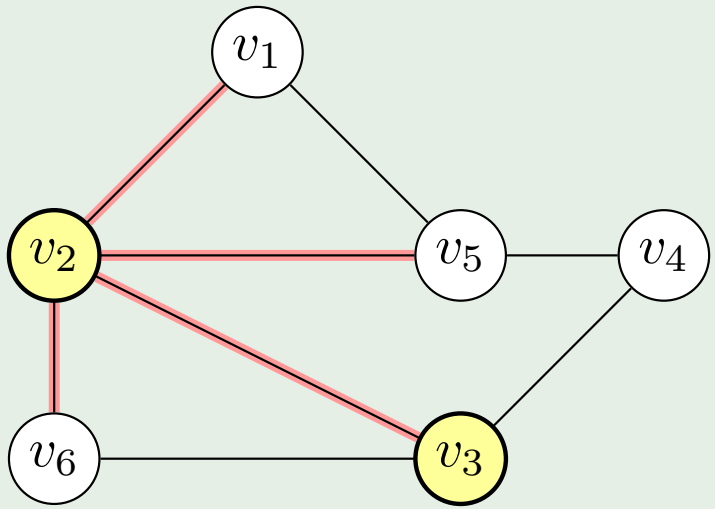
Beispiel



Definition: Vertex Cover

Eingabe: Ein Graph $G = (V, E)$ und eine natürliche Zahl k.
Aufgabe: Finde eine Teilmenge $C \subseteq V$ mit maximal k Knoten, sodass mindestens ein Endpunkt jeder Kante aus E in C ist.

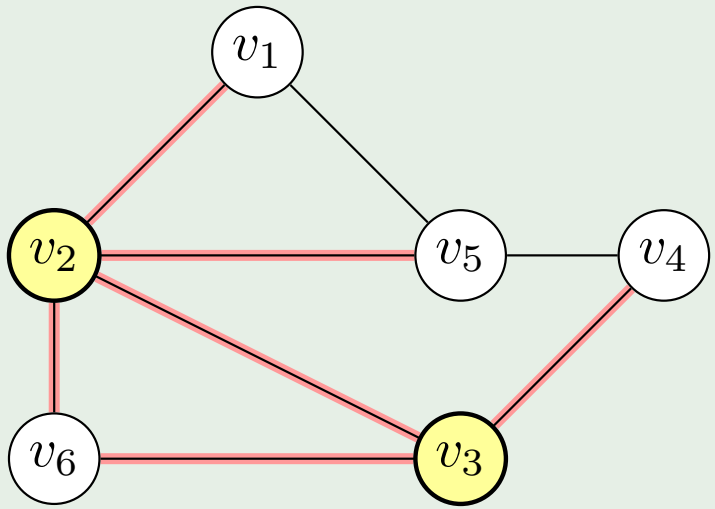
Beispiel



Definition: Vertex Cover

Eingabe: Ein Graph $G = (V, E)$ und eine natürliche Zahl k.
Aufgabe: Finde eine Teilmenge $C \subseteq V$ mit maximal k Knoten, sodass mindestens ein Endpunkt jeder Kante aus E in C ist.

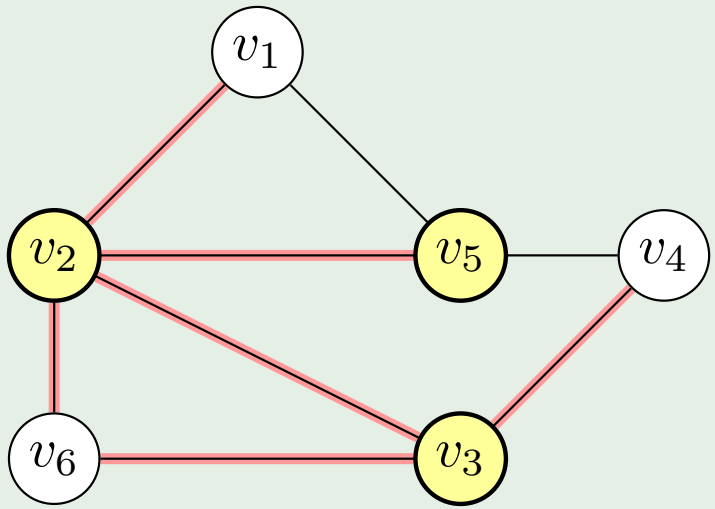
Beispiel



Definition: Vertex Cover

Eingabe: Ein Graph $G = (V, E)$ und eine natürliche Zahl k.
Aufgabe: Finde eine Teilmenge $C \subseteq V$ mit maximal k Knoten, sodass mindestens ein Endpunkt jeder Kante aus E in C ist.

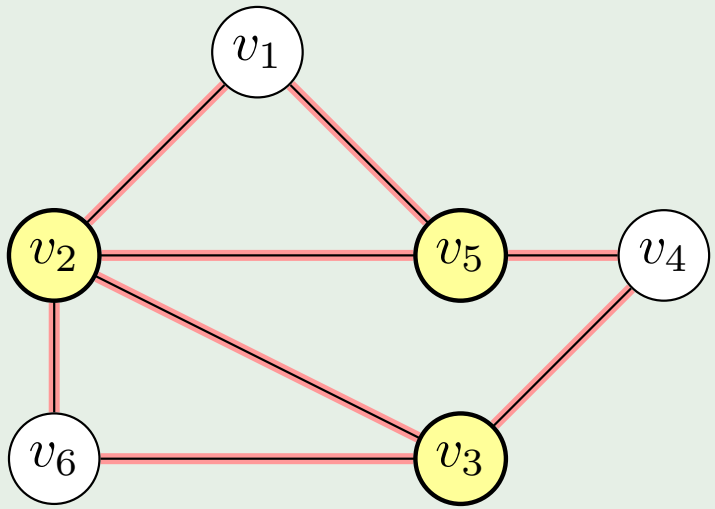
Beispiel



Definition: Vertex Cover

Eingabe: Ein Graph $G = (V, E)$ und eine natürliche Zahl k.
Aufgabe: Finde eine Teilmenge $C \subseteq V$ mit maximal k Knoten, sodass mindestens ein Endpunkt jeder Kante aus E in C ist.

Beispiel



Sei C das *vertex cover (VC)* für G.

Sei C das *vertex cover (VC)* für G.

Initialisierung:

Sei C das *vertex cover (VC)* für G.

Initialisierung:

- Leerer Graph: G'.

Sei C das *vertex cover (VC)* für G.

Initialisierung:

- Leerer Graph: G'.
- Leere Lösung: C.

Sei C das *vertex cover (VC)* für G.

Initialisierung:

- Leerer Graph: G'.
- Leere Lösung: C.

Iteration: Für jeden Knoten v in G:

Sei C das *vertex cover (VC)* für G.

Initialisierung:

- Leerer Graph: G'.
- Leere Lösung: C.

Iteration: Für jeden Knoten v in G:

- Füge v zu G' hinzu.

Sei C das *vertex cover (VC)* für G.

Initialisierung:

- Leerer Graph: G'.
- Leere Lösung: C.

Iteration: Für jeden Knoten v in G:

- Füge v zu G' hinzu.
- Füge v zu C hinzu.

Sei C das *vertex cover (VC)* für G.

Initialisierung:

- Leerer Graph: G'.
- Leere Lösung: C.

Iteration: Für jeden Knoten v in G:

- Füge v zu G' hinzu.
- Füge v zu C hinzu.
- Benutze einen Kompressions-Algorithmus, der C komprimiert.

Sei C das *vertex cover (VC)* für G.

Initialisierung:

- Leerer Graph: G'.
- Leere Lösung: C.

Iteration: Für jeden Knoten v in G:

- Füge v zu G' hinzu.
- Füge v zu C hinzu.
- Benutze einen Kompressions-Algorithmus, der C komprimiert.
- Gilt nun: $|C| > k$, so gibt es keinen VC der Länge k.

Sei C das *vertex cover (VC)* für G.

Initialisierung:

- Leerer Graph: G'.
- Leere Lösung: C.

Iteration: Für jeden Knoten v in G:

- Füge v zu G' hinzu.
- Füge v zu C hinzu.
- Benutze einen Kompressions-Algorithmus, der C komprimiert.
- Gilt nun: $|C| > k$, so gibt es keinen VC der Länge k.

Invariante: C ist VC für G' mit Größe maximal k.

Kompressionsschritt 1/2

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

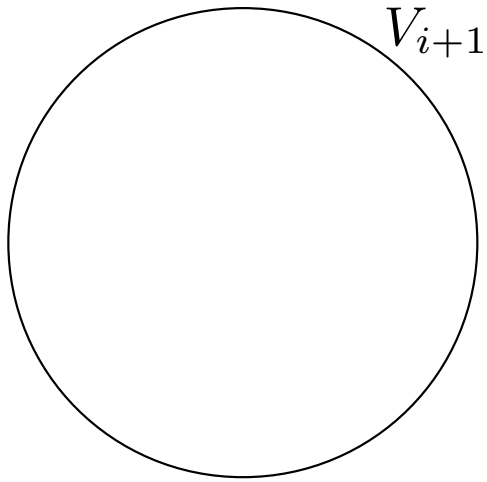
Annahme: Wir haben bereits ein VC C der Größe k für den Teilgraphen $G_i = G[v_1, \ldots, v_i]$.

Kompressionsschritt 1/2

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

Annahme: Wir haben bereits ein VC C der Größe k für den Teilgraphen $G_i = G[v_1, \ldots, v_i]$.

1. Füge v_{i+1} zu C hinzu:
   $C' := C \cup \{v_{i+1}\}$.

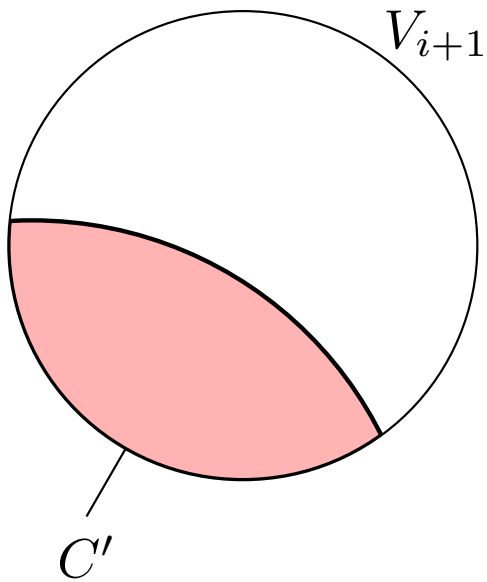


Kompressionsschritt 1/2

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

Annahme: Wir haben bereits ein VC C der Größe k für den Teilgraphen $G_i = G[v_1, \ldots, v_i]$.

1. Füge v_{i+1} zu C hinzu: $C' := C \cup \{v_{i+1}\}$.
2. C' ist $(k+1)$-großes VC für $G_{i+1} = G[v_1, \ldots, v_{i+1}] = (V_{i+1}, E_{i+1})$.

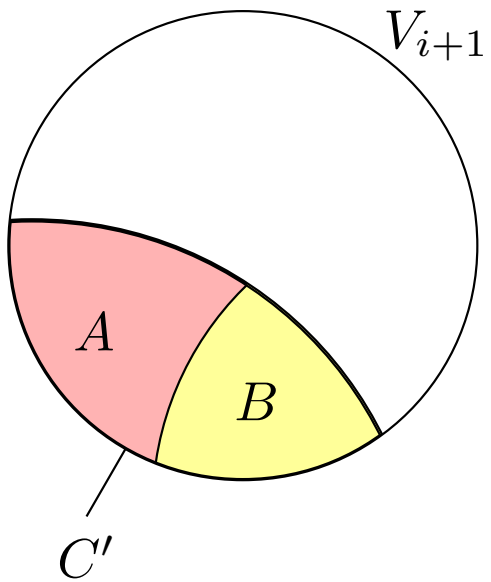


Kompressionsschritt 1/2

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

Annahme: Wir haben bereits ein VC C der Größe k für den Teilgraphen $G_i = G[v_1, \ldots, v_i]$.

1. Füge v_{i+1} zu C hinzu: $C' := C \cup \{v_{i+1}\}$.
2. C' ist $(k+1)$-großes VC für $G_{i+1} = G[v_1, \ldots, v_{i+1}] = (V_{i+1}, E_{i+1})$.
3. Man betrachtet alle 2^{k+1} Partitionen von C' in die Mengen A und B.

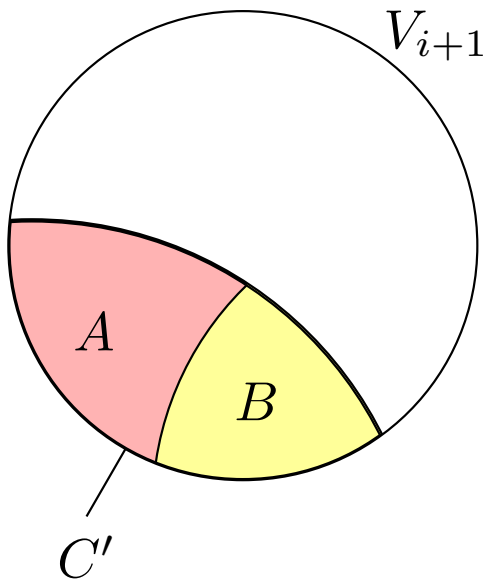


Kompressionsschritt 1/2

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

Annahme: Wir haben bereits ein VC C der Größe k für den Teilgraphen $G_i = G[v_1, \ldots, v_i]$.

1. Füge v_{i+1} zu C hinzu: $C' := C \cup \{v_{i+1}\}$.
2. C' ist $(k+1)$-großes VC für $G_{i+1} = G[v_1, \ldots, v_{i+1}] = (V_{i+1}, E_{i+1})$.
3. Man betrachtet alle 2^{k+1} Partitionen von C' in die Mengen A und B.
   - A soll die Knoten enthalten, die im neuen VC nicht mehr enthalten sind.

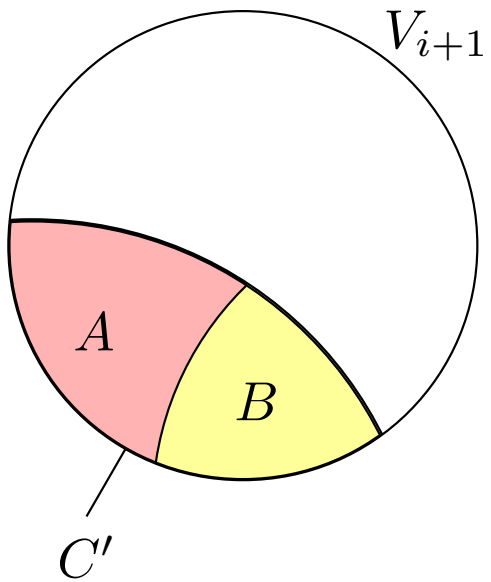


Kompressionsschritt 1/2

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

Annahme: Wir haben bereits ein VC C der Größe k für den Teilgraphen $G_i = G[v_1, \ldots, v_i]$.

1. Füge v_{i+1} zu C hinzu: $C' := C \cup \{v_{i+1}\}$.
2. C' ist $(k+1)$-großes VC für $G_{i+1} = G[v_1, \ldots, v_{i+1}] = (V_{i+1}, E_{i+1})$.
3. Man betrachtet alle 2^{k+1} Partitionen von C' in die Mengen A und B.
   - A soll die Knoten enthalten, die im neuen VC nicht mehr enthalten sind.
   - B soll die Knoten enthalten, die im neuen VC enthalten bleiben.



Kompressionsschritt 2/2

A soll die Knoten enthalten, die im neuen VC nicht mehr enthalten sind.
B soll die Knoten enthalten, die im neuen VC enthalten bleiben.

Kompressionsschritt 2/2

A soll die Knoten enthalten, die im neuen VC nicht mehr enthalten sind.
B soll die Knoten enthalten, die im neuen VC enthalten bleiben.

Für jede Partition werden folgende Schritte durchgeführt:

Kompressionsschritt 2/2

A soll die Knoten enthalten, die im neuen VC nicht mehr enthalten sind.
B soll die Knoten enthalten, die im neuen VC enthalten bleiben.

Für jede Partition werden folgende Schritte durchgeführt:

1. $C'' := \emptyset$

Kompressionsschritt 2/2

A soll die Knoten enthalten, die im neuen VC nicht mehr enthalten sind.
B soll die Knoten enthalten, die im neuen VC enthalten bleiben.

Für jede Partition werden folgende Schritte durchgeführt:

1. $C'' := \emptyset$
2. Für jede noch nicht durch B abgedeckte Kante (also: $\{u, v\} \in E_{i+1}$ mit $u \notin B$ und $v \notin B$), unterscheide drei Fälle:

Kompressionsschritt 2/2

A soll die Knoten enthalten, die im neuen VC nicht mehr enthalten sind.
B soll die Knoten enthalten, die im neuen VC enthalten bleiben.

Für jede Partition werden folgende Schritte durchgeführt:

1. $C'' := \emptyset$
2. Für jede noch nicht durch B abgedeckte Kante (also: $\{u, v\} \in E_{i+1}$ mit $u \notin B$ und $v \notin B$), unterscheide drei Fälle:

   (a) $u \in A$ und $v \in A$: Es gibt keinen VC ohne $\{u, v\}$ und somit ergibt sich aus der aktuellen Partition keine Lösung.

Kompressionsschritt 2/2

A soll die Knoten enthalten, die im neuen VC nicht mehr enthalten sind.
B soll die Knoten enthalten, die im neuen VC enthalten bleiben.

Für jede Partition werden folgende Schritte durchgeführt:

1. $C'' := \emptyset$
2. Für jede noch nicht durch B abgedeckte Kante (also: $\{u, v\} \in E_{i+1}$ mit $u \notin B$ und $v \notin B$), unterscheide drei Fälle:
   (a) $u \in A$ und $v \in A$: Es gibt keinen VC ohne $\{u, v\}$ und somit ergibt sich aus der aktuellen Partition keine Lösung.
   (b) $u \in A$ und $v \in (V_{i+1} \backslash C')$: Setze $C'' := C'' \cup \{v\}$

Kompressionsschritt 2/2

A soll die Knoten enthalten, die im neuen VC nicht mehr enthalten sind.
B soll die Knoten enthalten, die im neuen VC enthalten bleiben.

Für jede Partition werden folgende Schritte durchgeführt:

1. $C'' := \emptyset$
2. Für jede noch nicht durch B abgedeckte Kante (also: $\{u, v\} \in E_{i+1}$ mit $u \notin B$ und $v \notin B$), unterscheide drei Fälle:
   (a) $u \in A$ und $v \in A$: Es gibt keinen VC ohne $\{u, v\}$ und somit ergibt sich aus der aktuellen Partition keine Lösung.
   (b) $u \in A$ und $v \in (V_{i+1} \backslash C')$: Setze $C'' := C'' \cup \{v\}$
   (c) $v \in A$ und $u \in (V_{i+1} \backslash C')$: Setze $C'' := C'' \cup \{u\}$.

Kompressionsschritt 2/2

A soll die Knoten enthalten, die im neuen VC nicht mehr enthalten sind.
B soll die Knoten enthalten, die im neuen VC enthalten bleiben.

Für jede Partition werden folgende Schritte durchgeführt:

1. $C'' := \emptyset$
2. Für jede noch nicht durch B abgedeckte Kante (also: $\{u, v\} \in E_{i+1}$ mit $u \notin B$ und $v \notin B$), unterscheide drei Fälle:
   (a) $u \in A$ und $v \in A$: Es gibt keinen VC ohne $\{u, v\}$ und somit ergibt sich aus der aktuellen Partition keine Lösung.
   (b) $u \in A$ und $v \in (V_{i+1} \backslash C')$: Setze $C'' := C'' \cup \{v\}$
   (c) $v \in A$ und $u \in (V_{i+1} \backslash C')$: Setze $C'' := C'' \cup \{u\}$.
3. Wenn $|B \cup C''| \leq k$, dann gib $B \cup C''$ als Lösung zurück.

Kompressionsschritt 2/2

A soll die Knoten enthalten, die im neuen VC nicht mehr enthalten sind.
B soll die Knoten enthalten, die im neuen VC enthalten bleiben.

Für jede Partition werden folgende Schritte durchgeführt:

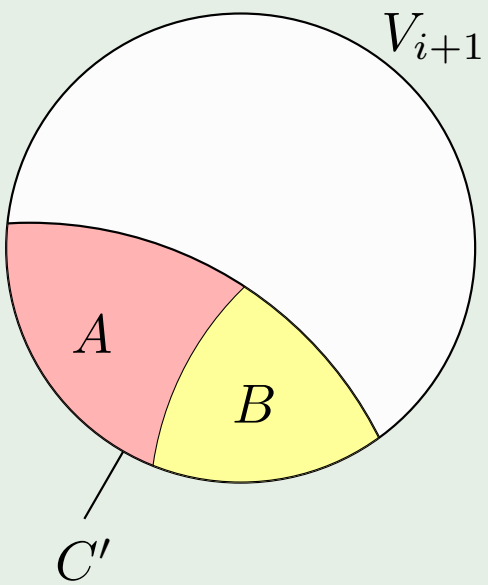
1. $C'' := \emptyset$
2. Für jede noch nicht durch B abgedeckte Kante (also: $\{u, v\} \in E_{i+1}$ mit $u \notin B$ und $v \notin B$), unterscheide drei Fälle:
   (a) $u \in A$ und $v \in A$: Es gibt keinen VC ohne $\{u, v\}$ und somit ergibt sich aus der aktuellen Partition keine Lösung.
   (b) $u \in A$ und $v \in (V_{i+1} \backslash C')$: Setze $C'' := C'' \cup \{v\}$
   (c) $v \in A$ und $u \in (V_{i+1} \backslash C')$: Setze $C'' := C'' \cup \{u\}$.
3. Wenn $|B \cup C''| \leq k$, dann gib $B \cup C''$ als Lösung zurück.

Wenn keine Partition eine Lösung ergibt, so gibt es kein k-großes VC.

Beispiele

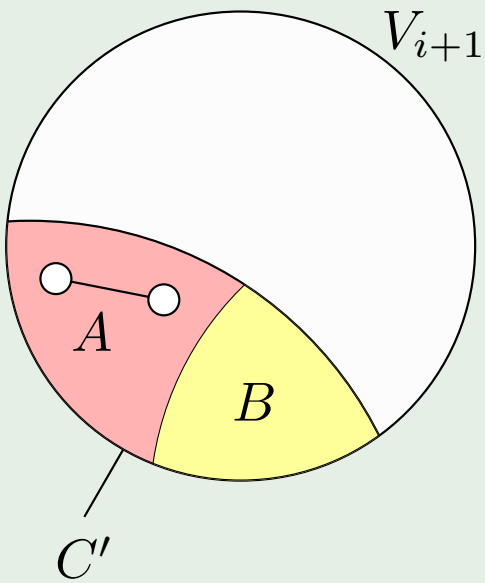
Beispiele

(a) $u \in A$ und $v \in A$



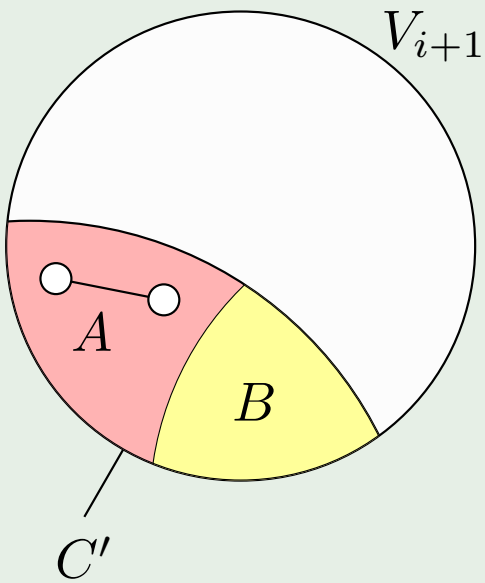
Beispiele

(a) $u \in A$ und $v \in A$



Beispiele

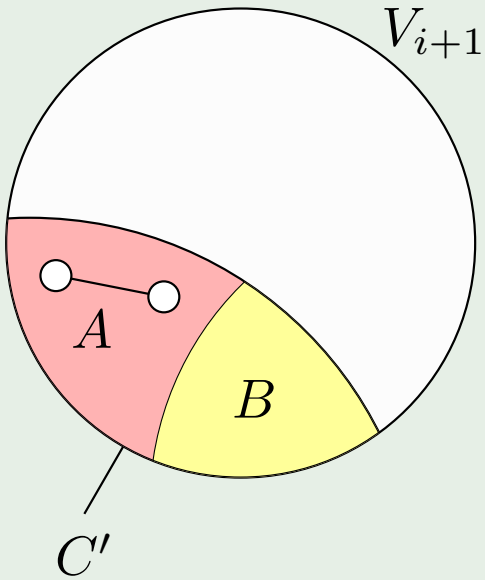
(a) $u \in A$ und $v \in A$



$\Rightarrow$ Partition liefert keine Lösung!

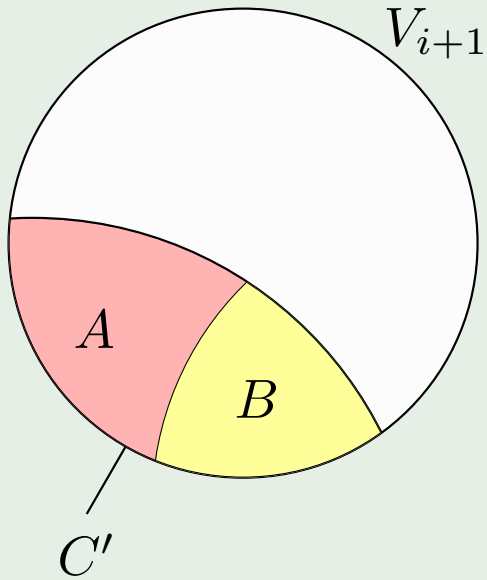
Beispiele

(a) $u \in A$ und $v \in A$



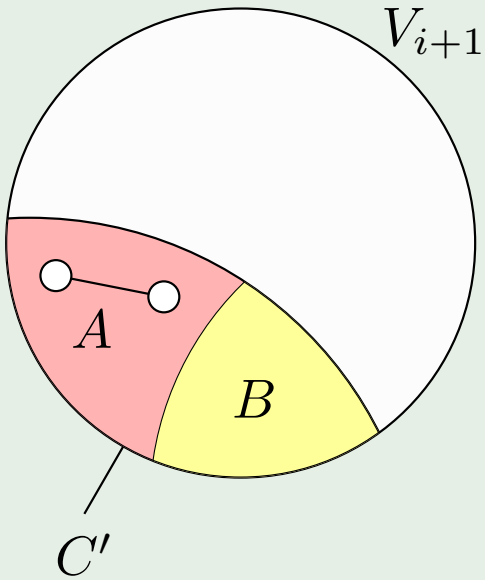
$\Rightarrow$ Partition liefert keine Lösung!

(b) $u \in A$ und $v \in (V_{i+1} \backslash C')$



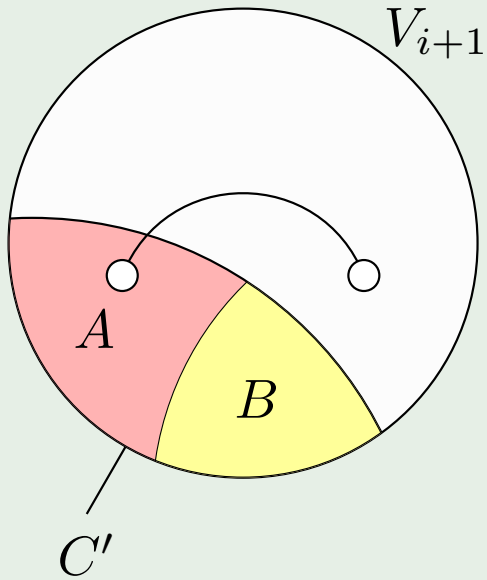
Beispiele

(a) $u \in A$ und $v \in A$



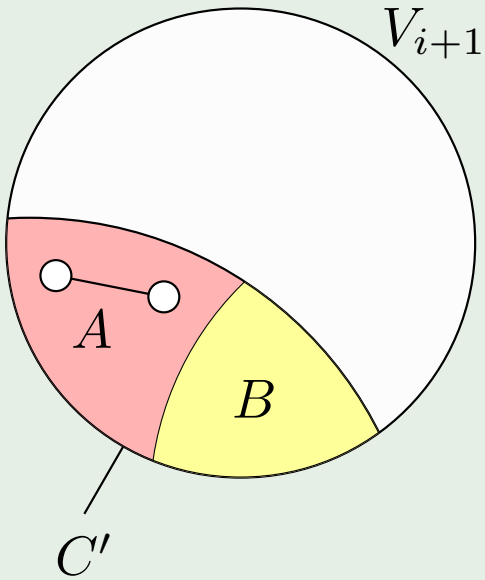
$\Rightarrow$ Partition liefert keine Lösung!

(b) $u \in A$ und $v \in (V_{i+1} \backslash C')$



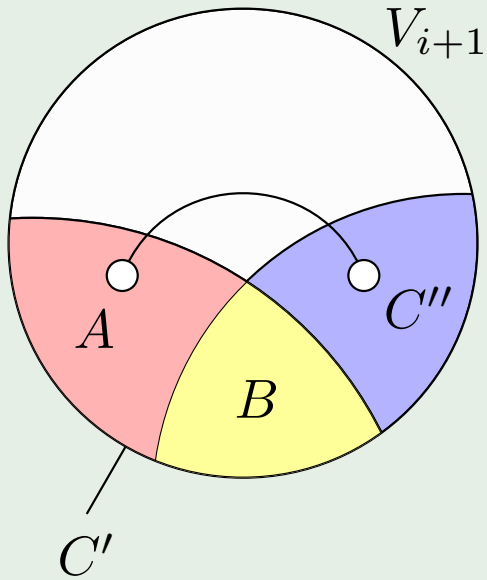
Beispiele

(a) $u \in A$ und $v \in A$



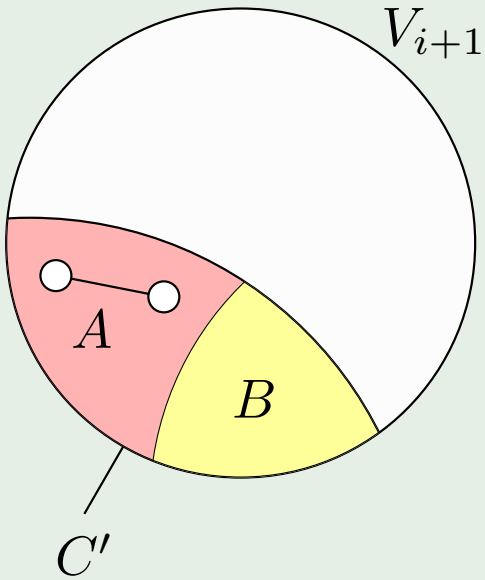
$\Rightarrow$ Partition liefert keine Lösung!

(b) $u \in A$ und $v \in (V_{i+1} \backslash C')$



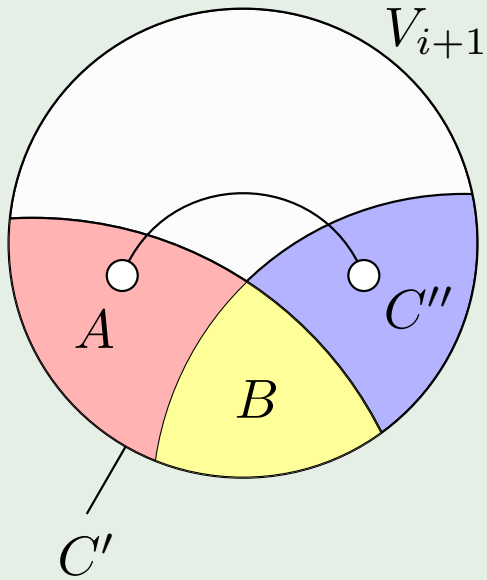
Beispiele

(a) $u \in A$ und $v \in A$



$\Rightarrow$ Partition liefert keine Lösung!

(b) $u \in A$ und $v \in (V_{i+1} \backslash C')$

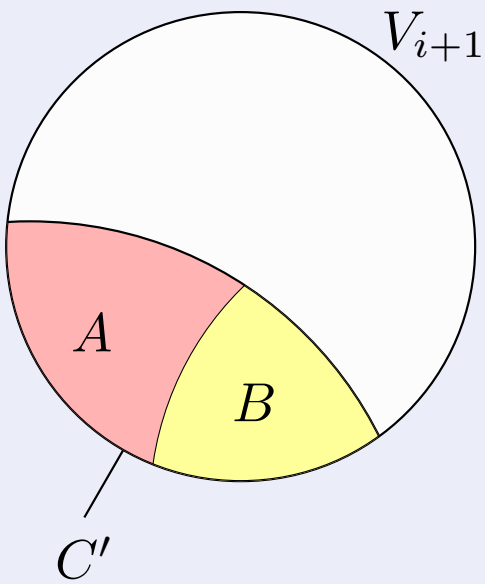


(c) folgt analog!

Waren das alle Fälle?

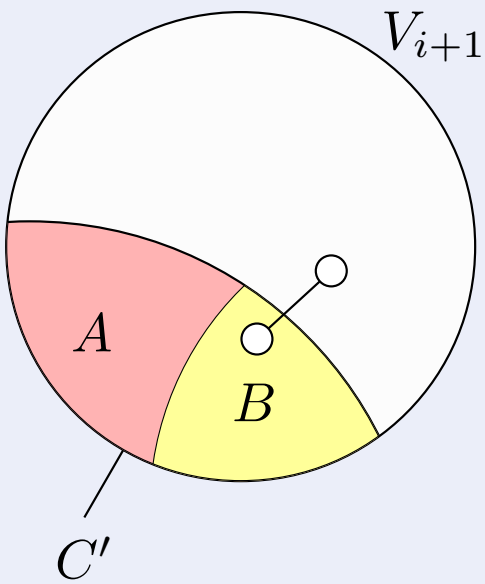
Waren das alle Fälle?

$\{u, v\} \in E_{i+1}$ mit $u \in B$ oder $v \in B$



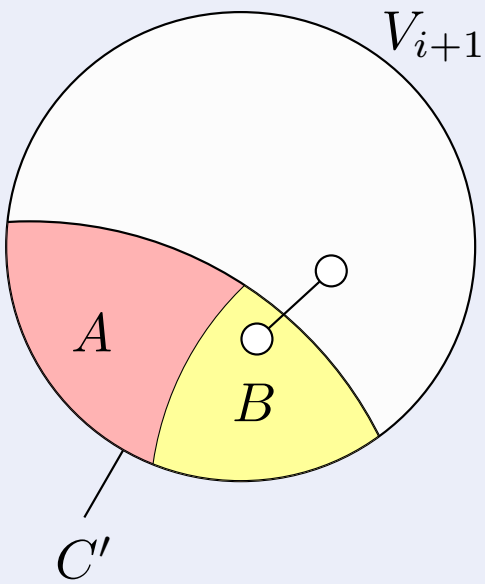
Waren das alle Fälle?

$\{u, v\} \in E_{i+1}$ mit $u \in B$ oder $v \in B$



Waren das alle Fälle?

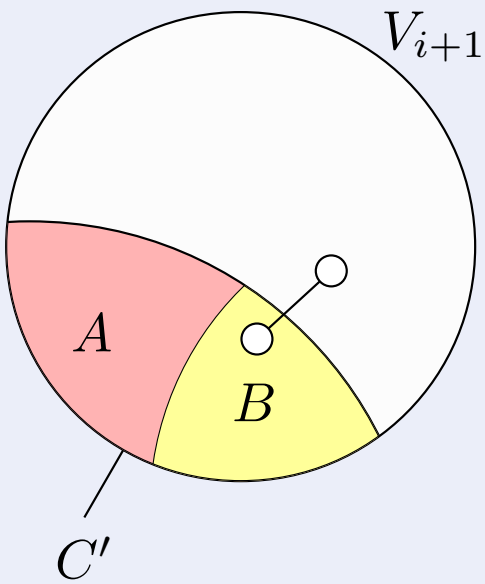
$\{u, v\} \in E_{i+1}$ mit $u \in B$ oder $v \in B$



$\Rightarrow$ Muss nicht betrachtet werden!

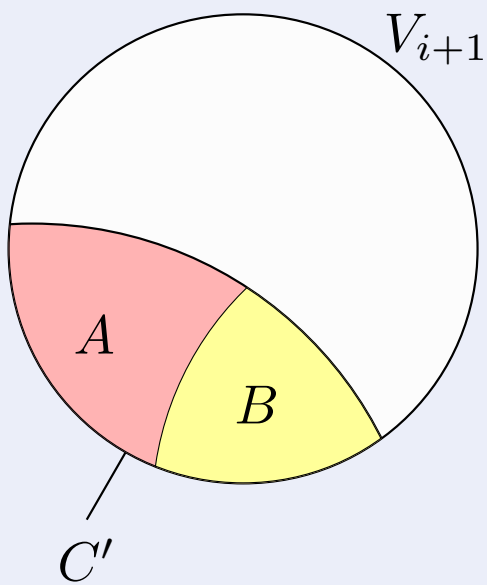
Waren das alle Fälle?

$\{u, v\} \in E_{i+1}$ mit $u \in B$ oder $v \in B$



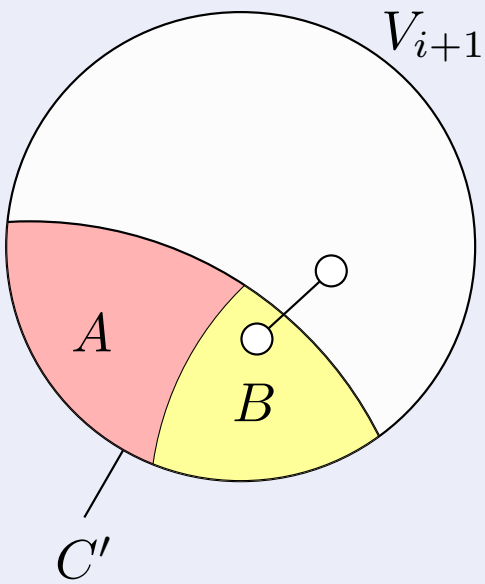
$\Rightarrow$ Muss nicht betrachtet werden!

$\{u, v\} \in E_{i+1}$ mit $u \notin C'$ und $v \notin C'$



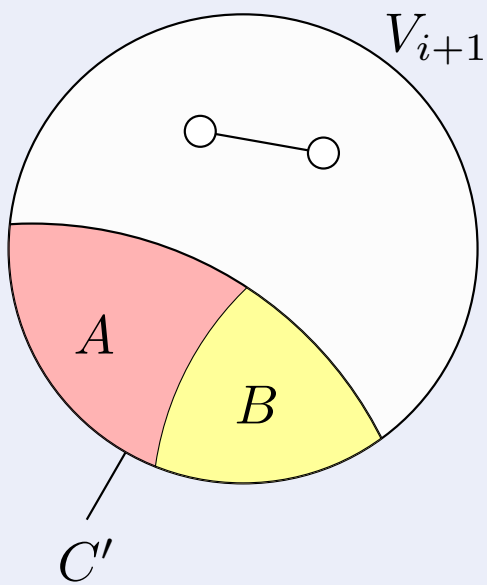
Waren das alle Fälle?

$\{u, v\} \in E_{i+1}$ mit $u \in B$ oder $v \in B$



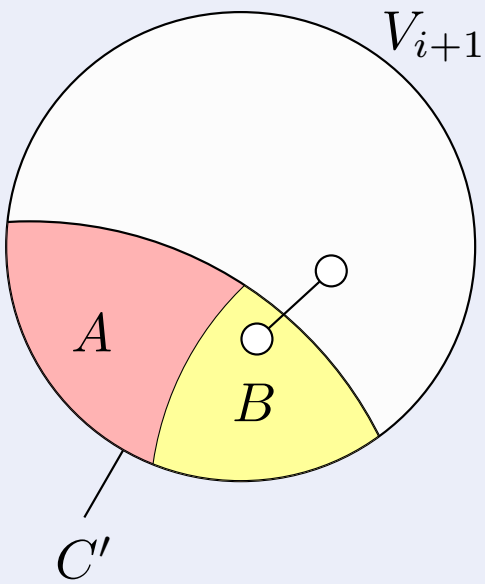
$\Rightarrow$ Muss nicht betrachtet werden!

$\{u, v\} \in E_{i+1}$ mit $u \notin C'$ und $v \notin C'$



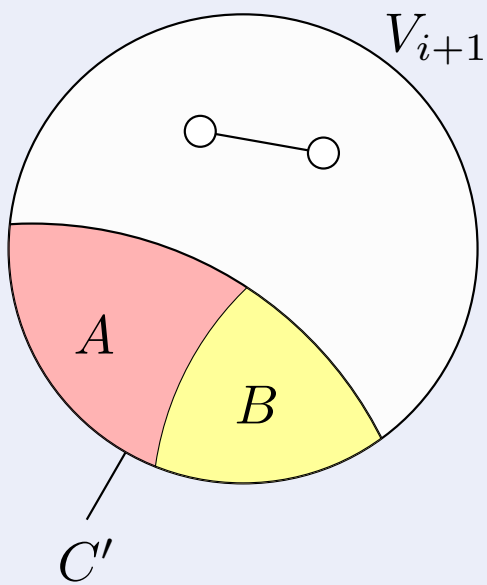
Waren das alle Fälle?

$\{u, v\} \in E_{i+1}$ mit $u \in B$ oder $v \in B$



$\Rightarrow$ Muss nicht betrachtet werden!

$\{u, v\} \in E_{i+1}$ mit $u \notin C'$ und $v \notin C'$



$\Rightarrow$ C' wäre so kein Vertex Cover!

Korrektheit 1/2

Wir stellen fest:

Korrektheit 1/2

Wir stellen fest:

- B enthält die Knoten, die sowohl zum $(k+1)$-großen VC, als auch zum k-großen VC gehören.

Korrektheit 1/2

Wir stellen fest:

- B enthält die Knoten, die sowohl zum $(k+1)$-großen VC, als auch zum k-großen VC gehören.
- Das Austesten aller 2^{k+1} Partitionen von C' stellt sicher, dass solch ein B (und das zugehörige A) gefunden wird.

Korrektheit 2/2

Zu Schritt 2 (Fallunterscheidung):

Korrektheit 2/2

Zu Schritt 2 (Fallunterscheidung):

- **Entscheidender Punkt:** Keine Kante hat beide Endpunkte in $V_{i+1} \backslash C'$, da C' sonst kein VC für $G[V_{i+1}]$ wäre.

Korrektheit 2/2

Zu Schritt 2 (Fallunterscheidung):

- **Entscheidender Punkt:** Keine Kante hat beide Endpunkte in $V_{i+1} \backslash C'$, da C' sonst kein VC für $G[V_{i+1}]$ wäre.
- Es muss sichergestellt werden, dass Kanten, die nicht durch einen Knoten aus B abgedeckt werden, durch einen anderen Knoten abgedeckt werden.

Korrektheit 2/2

Zu Schritt 2 (Fallunterscheidung):

- **Entscheidender Punkt:** Keine Kante hat beide Endpunkte in $V_{i+1} \setminus C'$, da C' sonst kein VC für $G[V_{i+1}]$ wäre.
- Es muss sichergestellt werden, dass Kanten, die nicht durch einen Knoten aus B abgedeckt werden, durch einen anderen Knoten abgedeckt werden.

  zu (a) Hier existiert eine Kante, die weder durch B noch durch einen anderen Knoten abgedeckt wird, daher muss die Partition *falsch* sein.

Korrektheit 2/2

Zu Schritt 2 (Fallunterscheidung):

- **Entscheidender Punkt:** Keine Kante hat beide Endpunkte in $V_{i+1} \setminus C'$, da C' sonst kein VC für $G[V_{i+1}]$ wäre.
- Es muss sichergestellt werden, dass Kanten, die nicht durch einen Knoten aus B abgedeckt werden, durch einen anderen Knoten abgedeckt werden.

  zu (a) Hier existiert eine Kante, die weder durch B noch durch einen anderen Knoten abgedeckt wird, daher muss die Partition *falsch* sein.

  (b) & (c) Ein Endknoten der Kante kann durch einen anderen Knoten abgedeckt werden.

Korrektheit 2/2

Zu Schritt 2 (Fallunterscheidung):

- **Entscheidender Punkt:** Keine Kante hat beide Endpunkte in $V_{i+1} \setminus C'$, da C' sonst kein VC für $G[V_{i+1}]$ wäre.
- Es muss sichergestellt werden, dass Kanten, die nicht durch einen Knoten aus B abgedeckt werden, durch einen anderen Knoten abgedeckt werden.
  - zu (a) Hier existiert eine Kante, die weder durch B noch durch einen anderen Knoten abgedeckt wird, daher muss die Partition *falsch* sein.
  - (b) & (c) Ein Endknoten der Kante kann durch einen anderen Knoten abgedeckt werden.
- Ist schließlich $B \cup C''$ klein genug, so ist es ein VC für G_{i+1}.

Laufzeit

[4] [4, Niedermeier, Guo - Fixed-Parameter Algorithms Slides - p. 156]

Laufzeit

- Die Fallunterscheidung kann in linearer Zeit durchgeführt werden: $O(|E|) = O(m)$.

[4] [4, Niedermeier, Guo - Fixed-Parameter Algorithms Slides - p. 156]

Laufzeit

- Die Fallunterscheidung kann in linearer Zeit durchgeführt werden: $O(|E|) = O(m)$.
- Es werden $O(2^{k+1}) = O(2^k)$ Partitionen getestet.

[4] [4, Niedermeier, Guo - Fixed-Parameter Algorithms Slides - p. 156]

Laufzeit

- Die Fallunterscheidung kann in linearer Zeit durchgeführt werden: $O(|E|) = O(m)$.
- Es werden $O(2^{k+1}) = O(2^k)$ Partitionen getestet.
- Es wird iteriert von G_1 bis zu $G_n = G$: $O(n)$.

[4] [4, Niedermeier, Guo - Fixed-Parameter Algorithms Slides - p. 156]

Laufzeit

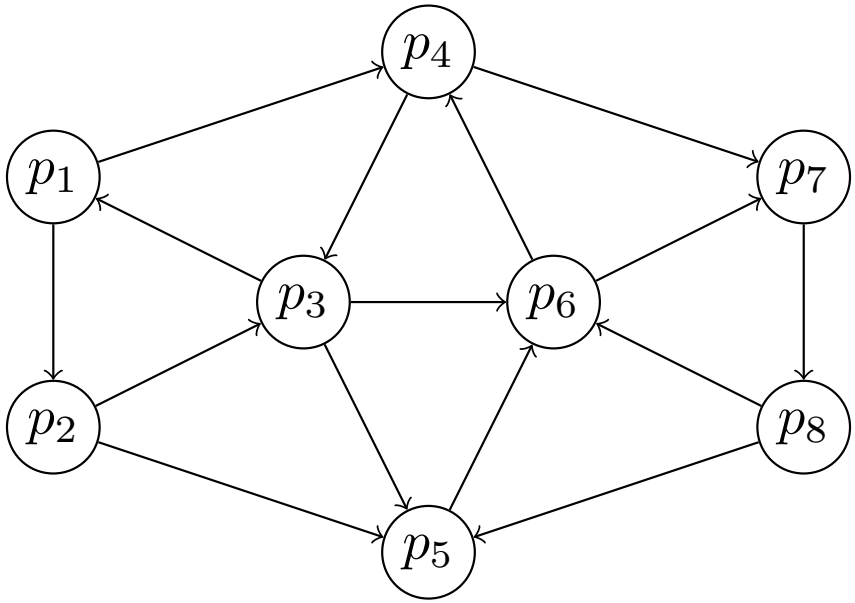
- Die Fallunterscheidung kann in linearer Zeit durchgeführt werden: $O(|E|) = O(m)$.
- Es werden $O(2^{k+1}) = O(2^k)$ Partitionen getestet.
- Es wird iteriert von G_1 bis zu $G_n = G$: $O(n)$.
- **Zusammen:**[4]

$$O(2^k \cdot m \cdot n)$$

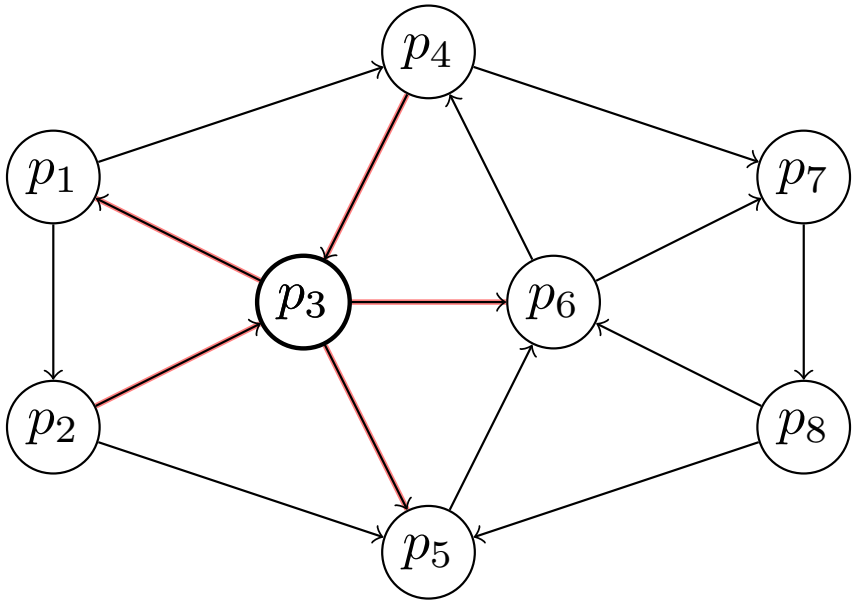
[4] [4, Niedermeier, Guo - Fixed-Parameter Algorithms Slides - p. 156]

Beispiel: Deadlock-Beseitigung

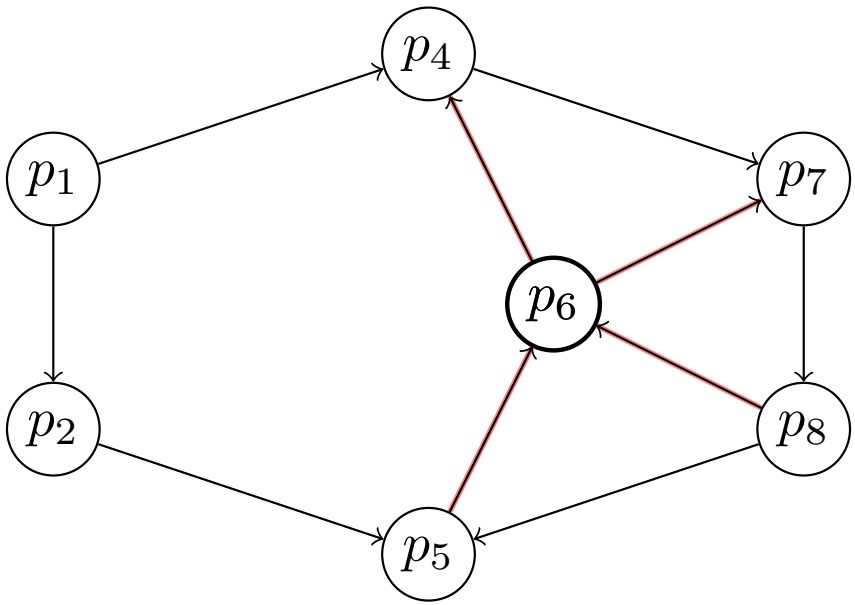
Beispiel: Deadlock-Beseitigung



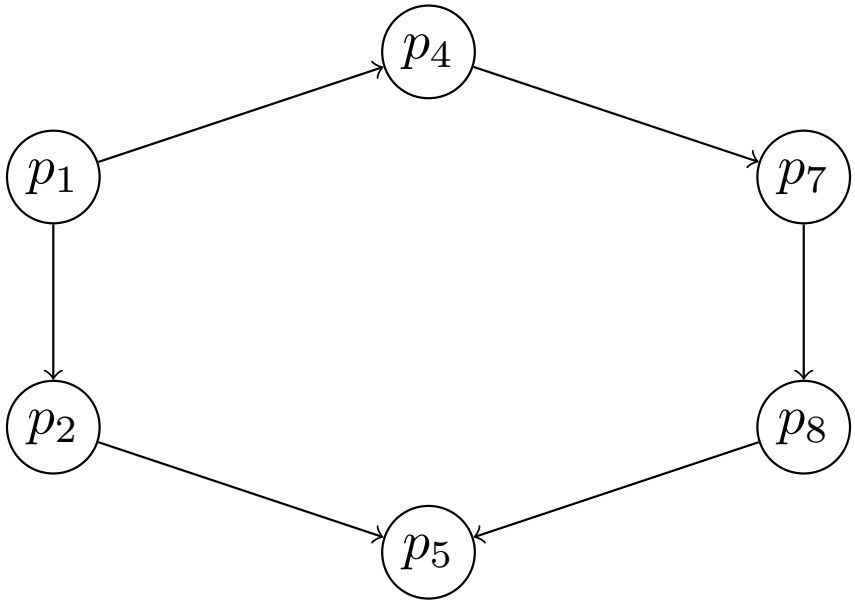
Beispiel: Deadlock-Beseitigung



Beispiel: Deadlock-Beseitigung



Beispiel: Deadlock-Beseitigung



Definition: Feedback Vertex Set

Eingabe: Ein Graph $G = (V, E)$ und eine positive ganze Zahl k.
Aufgabe: Finde eine Teilmenge $X \subseteq V$ mit k oder weniger Knoten, sodass jeder Kreis in G mindestens einen Knoten aus X enthält.

Definition: Feedback Vertex Set

Eingabe: Ein Graph $G = (V, E)$ und eine positive ganze Zahl k.
Aufgabe: Finde eine Teilmenge $X \subseteq V$ mit k oder weniger Knoten, sodass jeder Kreis in G mindestens einen Knoten aus X enthält.

Ergebnis: Der Graph $G[V \setminus X]$ ist kreisfrei.

Sei X das *feedback vertex set (FVS)* für G.

Sei X das *feedback vertex set (FVS)* für G.

Initialisierung:

Sei X das *feedback vertex set (FVS)* für G.

Initialisierung:

- Leerer Graph: G'.

Sei X das *feedback vertex set (FVS)* für G.

Initialisierung:

- Leerer Graph: G'.
- Leere Lösung: X.

Sei X das *feedback vertex set (FVS)* für G.

Initialisierung:

- Leerer Graph: G'.
- Leere Lösung: X.

Iteration: Für jeden Knoten v in G:

Sei X das *feedback vertex set (FVS)* für G.

Initialisierung:

- Leerer Graph: G'.
- Leere Lösung: X.

Iteration: Für jeden Knoten v in G:

- Füge v zu G' hinzu.

Sei X das *feedback vertex set (FVS)* für G.

Initialisierung:

- Leerer Graph: G'.
- Leere Lösung: X.

Iteration: Für jeden Knoten v in G:

- Füge v zu G' hinzu.
- Füge v zu X hinzu.

Sei X das *feedback vertex set (FVS)* für G.

Initialisierung:

- Leerer Graph: G'.
- Leere Lösung: X.

Iteration: Für jeden Knoten v in G:

- Füge v zu G' hinzu.
- Füge v zu X hinzu.
- Benutze einen Kompressions-Algorithmus, der X komprimiert.

Sei X das *feedback vertex set (FVS)* für G.

Initialisierung:

- Leerer Graph: G'.
- Leere Lösung: X.

Iteration: Für jeden Knoten v in G:

- Füge v zu G' hinzu.
- Füge v zu X hinzu.
- Benutze einen Kompressions-Algorithmus, der X komprimiert.
- Gilt nun: $|X| > k$, so gibt es keinen FVS der Länge k.

Sei X das *feedback vertex set (FVS)* für G.

Initialisierung:

- Leerer Graph: G'.
- Leere Lösung: X.

Iteration: Für jeden Knoten v in G:

- Füge v zu G' hinzu.
- Füge v zu X hinzu.
- Benutze einen Kompressions-Algorithmus, der X komprimiert.
- Gilt nun: $|X| > k$, so gibt es keinen FVS der Länge k.

Invariante: X ist FVS für G' mit Größe maximal k.

Kompressionsschritt

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

Annahme: Wir haben bereits ein FVS X_i der Größe k für den Teilgraphen $G_i = G[v_1, \ldots, v_i]$.

Kompressionsschritt

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

Annahme: Wir haben bereits ein FVS X_i der Größe k für den Teilgraphen $G_i = G[v_1, \ldots, v_i]$.

1. Füge v_{i+1} zu X_i hinzu: $X := X_i \cup \{v_{i+1}\}$.

Kompressionsschritt

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

Annahme: Wir haben bereits ein FVS X_i der Größe k für den Teilgraphen $G_i = G[v_1, \ldots, v_i]$.

1. Füge v_{i+1} zu X_i hinzu: $X := X_i \cup \{v_{i+1}\}$.
2. X ist $(k+1)$-großes FVS für $G_{i+1} = G[v_1, \ldots, v_{i+1}] = (V_{i+1}, E_{i+1})$.

Kompressionsschritt

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

Annahme: Wir haben bereits ein FVS X_i der Größe k für den Teilgraphen $G_i = G[v_1, \ldots, v_i]$.

1. Füge v_{i+1} zu X_i hinzu: $X := X_i \cup \{v_{i+1}\}$.
2. X ist $(k+1)$-großes FVS für $G_{i+1} = G[v_1, \ldots, v_{i+1}] = (V_{i+1}, E_{i+1})$.
3. Man betrachtet alle 2^{k+1} Partitionen von X in die Mengen A und B.

Kompressionsschritt

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

Annahme: Wir haben bereits ein FVS X_i der Größe k für den Teilgraphen $G_i = G[v_1, \ldots, v_i]$.

1. Füge v_{i+1} zu X_i hinzu: $X := X_i \cup \{v_{i+1}\}$.
2. X ist $(k+1)$-großes FVS für $G_{i+1} = G[v_1, \ldots, v_{i+1}] = (V_{i+1}, E_{i+1})$.
3. Man betrachtet alle 2^{k+1} Partitionen von X in die Mengen A und B.
   - A soll die Knoten enthalten, die im neuen FVS X' nicht mehr enthalten sind.

Kompressionsschritt

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

Annahme: Wir haben bereits ein FVS X_i der Größe k für den Teilgraphen $G_i = G[v_1, \ldots, v_i]$.

1. Füge v_{i+1} zu X_i hinzu: $X := X_i \cup \{v_{i+1}\}$.
2. X ist $(k+1)$-großes FVS für $G_{i+1} = G[v_1, \ldots, v_{i+1}] = (V_{i+1}, E_{i+1})$.
3. Man betrachtet alle 2^{k+1} Partitionen von X in die Mengen A und B.
   - A soll die Knoten enthalten, die im neuen FVS X' nicht mehr enthalten sind.
   - B soll die Knoten enthalten, die im neuen FVS X' enthalten bleiben.

Kompressionsschritt

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

Annahme: Wir haben bereits ein FVS X_i der Größe k für den Teilgraphen $G_i = G[v_1, \ldots, v_i]$.

1. Füge v_{i+1} zu X_i hinzu: $X := X_i \cup \{v_{i+1}\}$.
2. X ist $(k+1)$-großes FVS für $G_{i+1} = G[v_1, \ldots, v_{i+1}] = (V_{i+1}, E_{i+1})$.
3. Man betrachtet alle 2^{k+1} Partitionen von X in die Mengen A und B.
   - A soll die Knoten enthalten, die im neuen FVS X' nicht mehr enthalten sind.
   - B soll die Knoten enthalten, die im neuen FVS X' enthalten bleiben.

$\Rightarrow$ Bis hierher wie bei Vertex Cover!

Kompressionsschritt

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

Annahme: Wir haben bereits ein FVS X_i der Größe k für den Teilgraphen $G_i = G[v_1, \ldots, v_i]$.

1. Füge v_{i+1} zu X_i hinzu: $X := X_i \cup \{v_{i+1}\}$.
2. X ist $(k+1)$-großes FVS für $G_{i+1} = G[v_1, \ldots, v_{i+1}] = (V_{i+1}, E_{i+1})$.
3. Man betrachtet alle 2^{k+1} Partitionen von X in die Mengen A und B.
   - A soll die Knoten enthalten, die im neuen FVS X' nicht mehr enthalten sind.
   - B soll die Knoten enthalten, die im neuen FVS X' enthalten bleiben.

$\Rightarrow$ Bis hierher wie bei Vertex Cover!

Im folgenden sei $V_{i+1} = V$.

Idee:

Idee:

Vervollständige B mit Knoten aus einer Menge $V' \subseteq V \backslash X$ zu einem k-großen FVS X'.

Idee:

Vervollständige B mit Knoten aus einer Menge $V' \subseteq V \backslash X$ zu einem k-großen FVS X'.

$\Rightarrow$ Benutze Reduktionsregeln, um V' so klein wie möglich zu halten!

Idee:

Vervollständige B mit Knoten aus einer Menge $V' \subseteq V \backslash X$ zu einem k-großen FVS X'.

- $\Rightarrow$ Benutze Reduktionsregeln, um V' so klein wie möglich zu halten!
- $\Rightarrow$ Wenn V' in Bezug zu A zu groß ist, dann gibt es keine Lösung (für diese Partition)!

Idee:

Vervollständige B mit Knoten aus einer Menge $V' \subseteq V \backslash X$ zu einem k-großen FVS X'.

$\Rightarrow$ Benutze Reduktionsregeln, um V' so klein wie möglich zu halten!

$\Rightarrow$ Wenn V' in Bezug zu A zu groß ist, dann gibt es keine Lösung (für diese Partition)!

Beobachtungen:

Idee:

Vervollständige B mit Knoten aus einer Menge $V' \subseteq V \backslash X$ zu einem k-großen FVS X'.

$\Rightarrow$ Benutze Reduktionsregeln, um V' so klein wie möglich zu halten!

$\Rightarrow$ Wenn V' in Bezug zu A zu groß ist, dann gibt es keine Lösung (für diese Partition)!

Beobachtungen:

- A darf keine Kreise enthalten, sonst gäbe es kein FVS ohne Knoten aus A!

Idee:

Vervollständige B mit Knoten aus einer Menge $V' \subseteq V \backslash X$ zu einem k-großen FVS X'.

$\Rightarrow$ Benutze Reduktionsregeln, um V' so klein wie möglich zu halten!

$\Rightarrow$ Wenn V' in Bezug zu A zu groß ist, dann gibt es keine Lösung (für diese Partition)!

Beobachtungen:

- A darf keine Kreise enthalten, sonst gäbe es kein FVS ohne Knoten aus A!
- A ist ein FVS für den Teilgraph $G_{i+1}[V \backslash B]$.

Idee:

Vervollständige B mit Knoten aus einer Menge $V' \subseteq V \backslash X$ zu einem k-großen FVS X'.

$\Rightarrow$ Benutze Reduktionsregeln, um V' so klein wie möglich zu halten!

$\Rightarrow$ Wenn V' in Bezug zu A zu groß ist, dann gibt es keine Lösung (für diese Partition)!

Beobachtungen:

- A darf keine Kreise enthalten, sonst gäbe es kein FVS ohne Knoten aus A!
- A ist ein FVS für den Teilgraph $G_{i+1}[V \backslash B]$.

Vorbereitung zur Reduktion:

Idee:

Vervollständige B mit Knoten aus einer Menge $V' \subseteq V \backslash X$ zu einem k-großen FVS X'.

$\Rightarrow$ Benutze Reduktionsregeln, um V' so klein wie möglich zu halten!

$\Rightarrow$ Wenn V' in Bezug zu A zu groß ist, dann gibt es keine Lösung (für diese Partition)!

Beobachtungen:

- A darf keine Kreise enthalten, sonst gäbe es kein FVS ohne Knoten aus A!
- A ist ein FVS für den Teilgraph $G_{i+1}[V \backslash B]$.

Vorbereitung zur Reduktion:

- Entferne aus G_{i+1} die Knoten der Menge B.

Idee:

Vervollständige B mit Knoten aus einer Menge $V' \subseteq V \backslash X$ zu einem k-großen FVS X'.

$\Rightarrow$ Benutze Reduktionsregeln, um V' so klein wie möglich zu halten!

$\Rightarrow$ Wenn V' in Bezug zu A zu groß ist, dann gibt es keine Lösung (für diese Partition)!

Beobachtungen:

- A darf keine Kreise enthalten, sonst gäbe es kein FVS ohne Knoten aus A!
- A ist ein FVS für den Teilgraph $G_{i+1}[V \backslash B]$.

Vorbereitung zur Reduktion:

- Entferne aus G_{i+1} die Knoten der Menge B.
  $\Rightarrow$ Diese sind bereits im neuen FVS X'!

Idee:

Vervollständige B mit Knoten aus einer Menge $V' \subseteq V \backslash X$ zu einem k-großen FVS X'.

$\Rightarrow$ Benutze Reduktionsregeln, um V' so klein wie möglich zu halten!

$\Rightarrow$ Wenn V' in Bezug zu A zu groß ist, dann gibt es keine Lösung (für diese Partition)!

Beobachtungen:

- A darf keine Kreise enthalten, sonst gäbe es kein FVS ohne Knoten aus A!
- A ist ein FVS für den Teilgraph $G_{i+1}[V \backslash B]$.

Vorbereitung zur Reduktion:

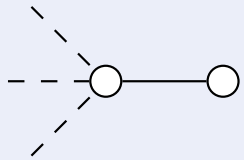
- Entferne aus G_{i+1} die Knoten der Menge B.
  $\Rightarrow$ Diese sind bereits im neuen FVS X'!
- Wende die Reduktionsregeln auf die Knoten $V \backslash X$ an.

Reduktionsregeln

Reduktionsregeln

Reduktionsregel 1:

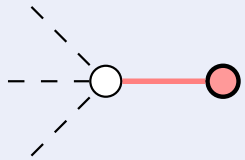
Entferne alle Knoten v mit $deg(v) = 1$.



Reduktionsregeln

Reduktionsregel 1:

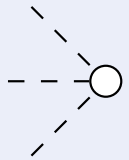
Entferne alle Knoten v mit $deg(v) = 1$.



Reduktionsregeln

Reduktionsregel 1:

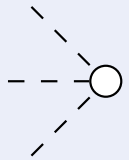
Entferne alle Knoten v mit $deg(v) = 1$.



Reduktionsregeln

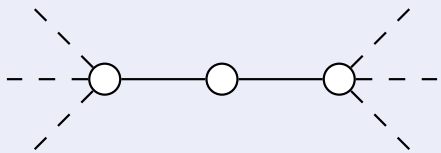
Reduktionsregel 1:

Entferne alle Knoten v mit $deg(v) = 1$.



Reduktionsregel 2:

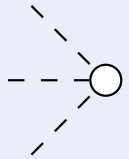
Überbrücke alle Knoten v mit $deg(v) = 2$.



Reduktionsregeln

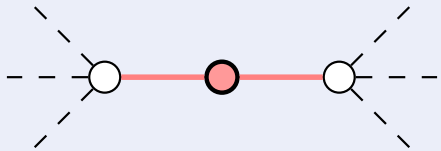
Reduktionsregel 1:

Entferne alle Knoten v mit $deg(v) = 1$.



Reduktionsregel 2:

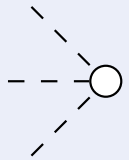
Überbrücke alle Knoten v mit $deg(v) = 2$.



Reduktionsregeln

Reduktionsregel 1:

Entferne alle Knoten v mit $deg(v) = 1$.



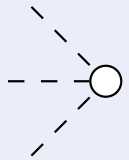
Reduktionsregel 2:

Überbrücke alle Knoten v mit $deg(v) = 2$.

Reduktionsregeln

Reduktionsregel 1:

Entferne alle Knoten v mit $deg(v) = 1$.



Reduktionsregel 2:

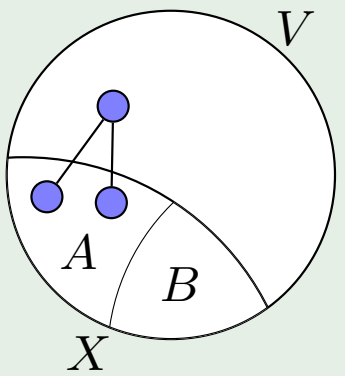
Überbrücke alle Knoten v mit $deg(v) = 2$.

Achtung: Für Regel 2 gibt es zwei Ausnahmen!

Ausnahme 1:

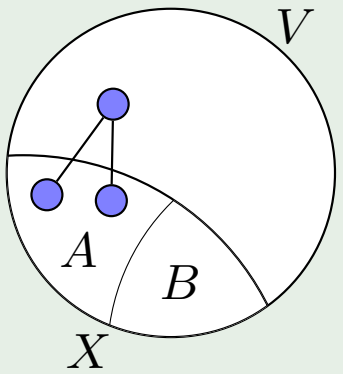
Ausnahme 1:

Überbrücke keine Knoten mit Grad 2, die beide Nachbarn in A haben.



Ausnahme 1:

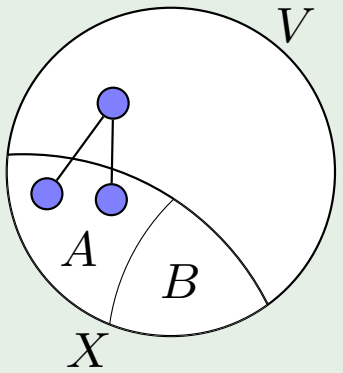
Überbrücke keine Knoten mit Grad 2, die beide Nachbarn in A haben.



Ausnahme 2:

Ausnahme 1:

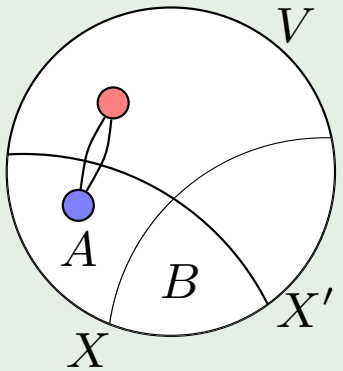
Überbrücke keine Knoten mit Grad 2, die beide Nachbarn in A haben.



Ausnahme 2:

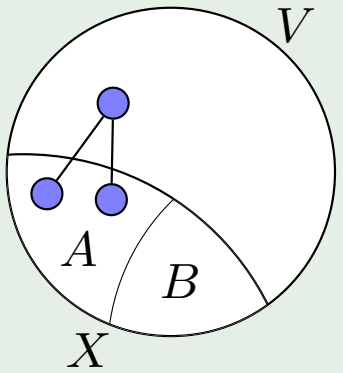
Während der Reduktion entsteht ein Kreis der Länge 2:

- Einer der Knoten muss in A liegen.
- Entferne den anderen Knoten und füge ihn zu X' hinzu!



Ausnahme 1:

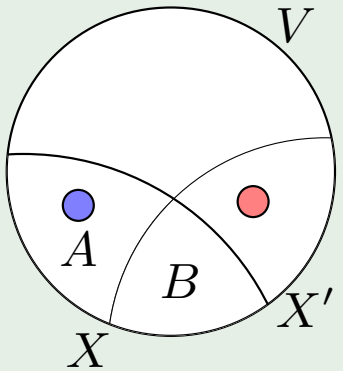
Überbrücke keine Knoten mit Grad 2, die beide Nachbarn in A haben.



Ausnahme 2:

Während der Reduktion entsteht ein Kreis der Länge 2:

- Einer der Knoten muss in A liegen.
- Entferne den anderen Knoten und füge ihn zu X' hinzu!



Beschränkung von $|V'|$

Beschränkung von $|V'|$

Idee:

Beschränkung von $|V'|$

Idee:

- Teile V' in drei Mengen ein.

Beschränkung von $|V'|$

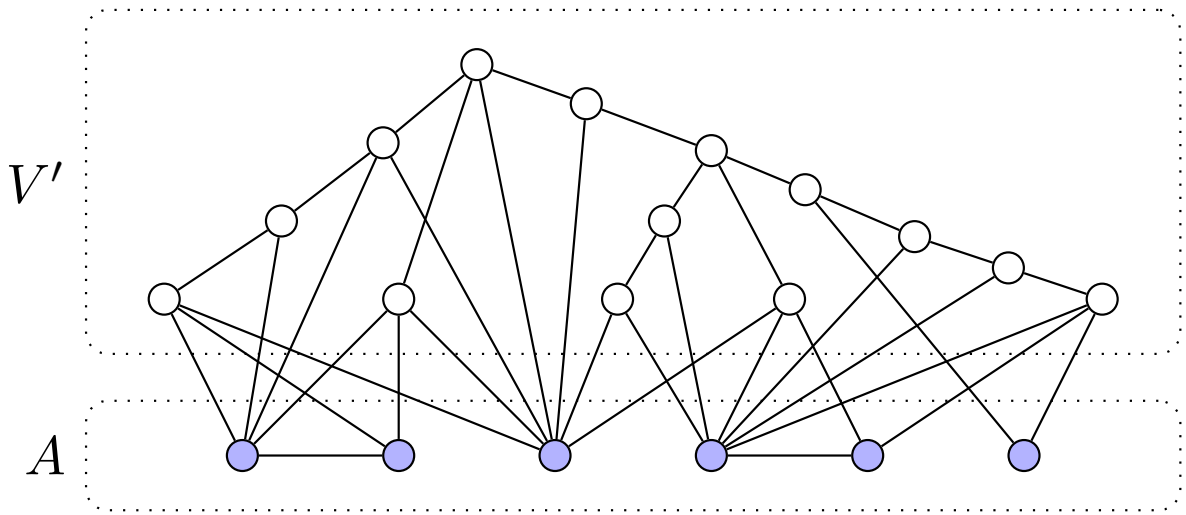
Idee:

- Teile V' in drei Mengen ein.
- Beschränke die Größe jeder Menge einzeln.

Beschränkung von $|V'|$

Idee:

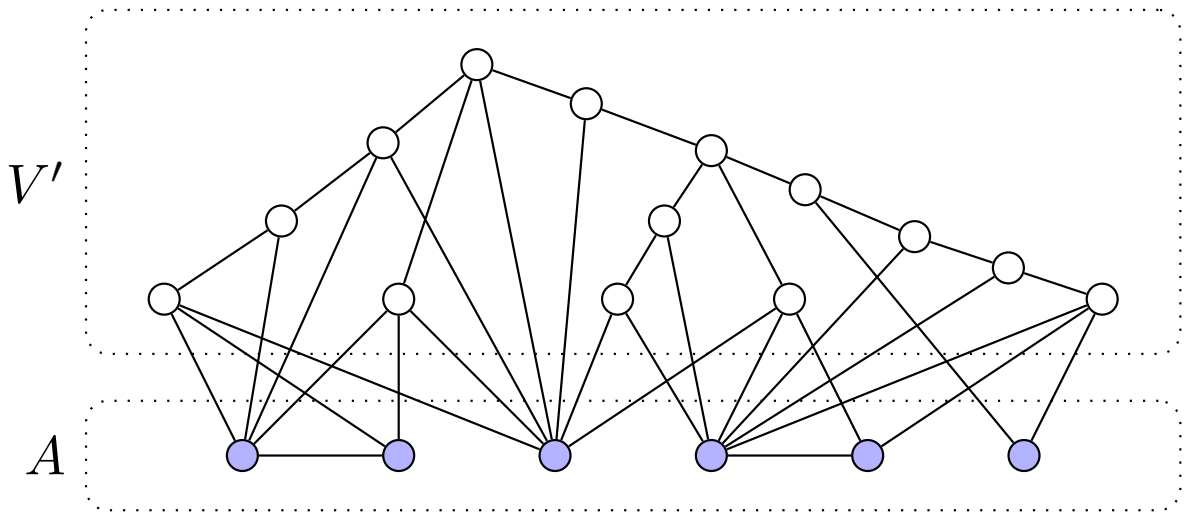
- Teile V' in drei Mengen ein.
- Beschränke die Größe jeder Menge einzeln.



Beschränkung von $|V'|$

Idee:

- Teile V' in drei Mengen ein.
- Beschränke die Größe jeder Menge einzeln.

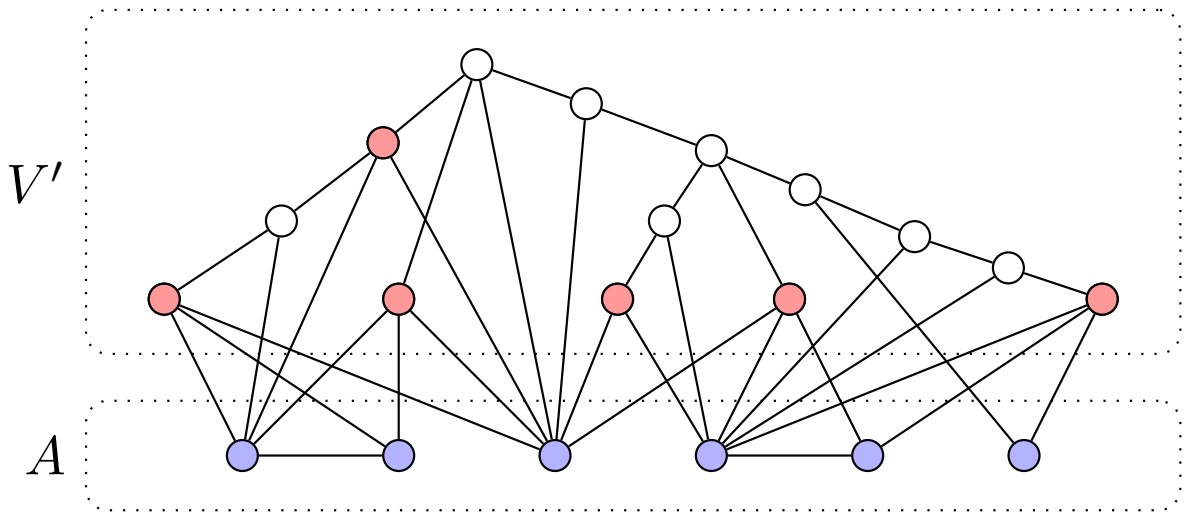


- $F := \{v \in V' || N(v) \cap A| \geq 2\}$

Beschränkung von $|V'|$

Idee:

- Teile V' in drei Mengen ein.
- Beschränke die Größe jeder Menge einzeln.

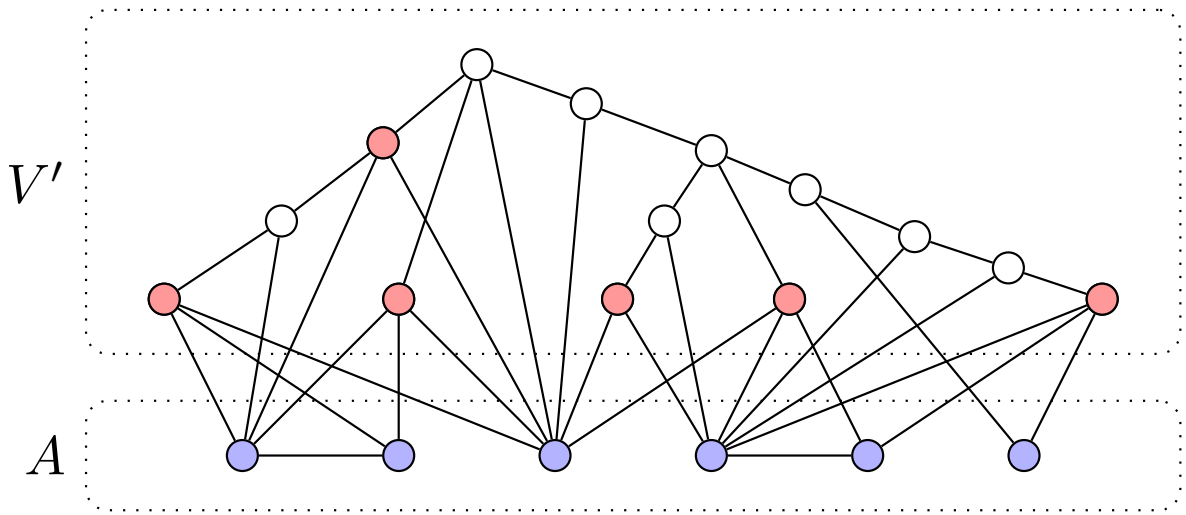


- $F := \{v \in V' || N(v) \cap A| \geq 2\}$

Beschränkung von $|V'|$

Idee:

- Teile V' in drei Mengen ein.
- Beschränke die Größe jeder Menge einzeln.

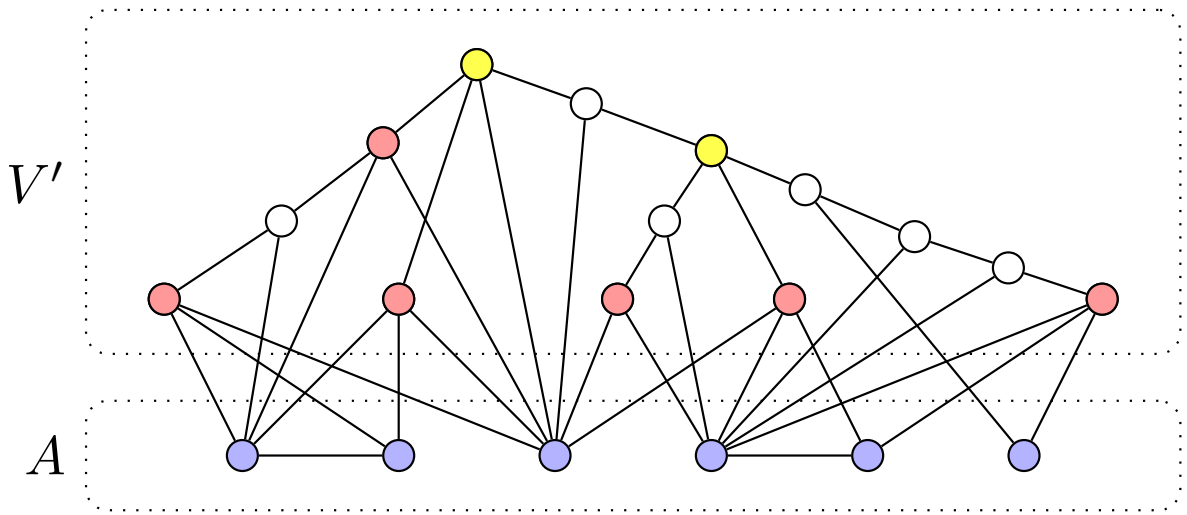


- $F := \{v \in V' || N(v) \cap A| \geq 2\}$
- $H := \{v \in V' \backslash F || N(v) \cap V'| \geq 3\}$

Beschränkung von $|V'|$

Idee:

- Teile V' in drei Mengen ein.
- Beschränke die Größe jeder Menge einzeln.

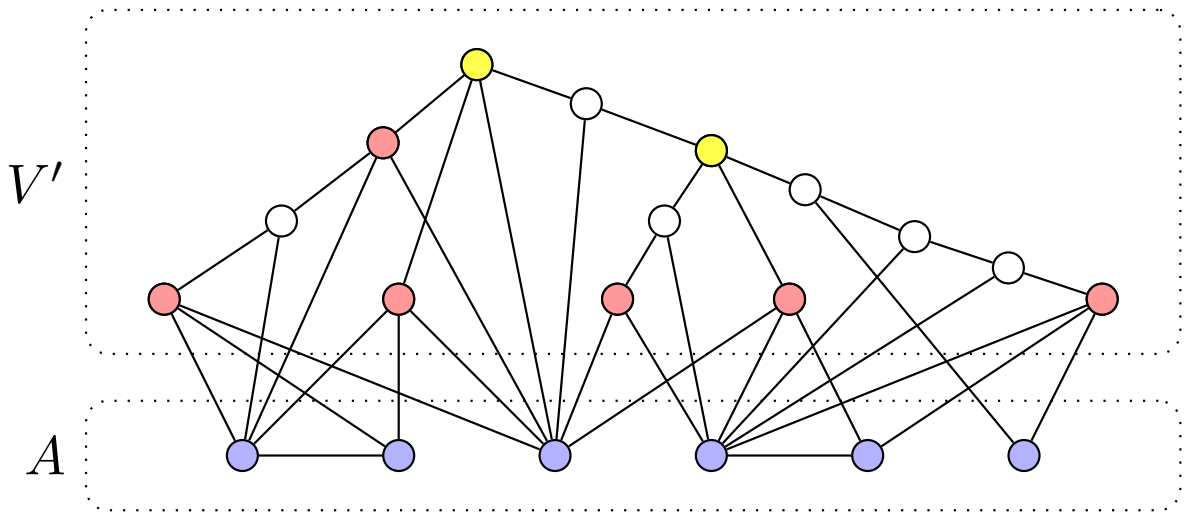


- $F := \{v \in V' || N(v) \cap A| \geq 2\}$
- $H := \{v \in V' \backslash F || N(v) \cap V'| \geq 3\}$

Beschränkung von $|V'|$

Idee:

- Teile V' in drei Mengen ein.
- Beschränke die Größe jeder Menge einzeln.

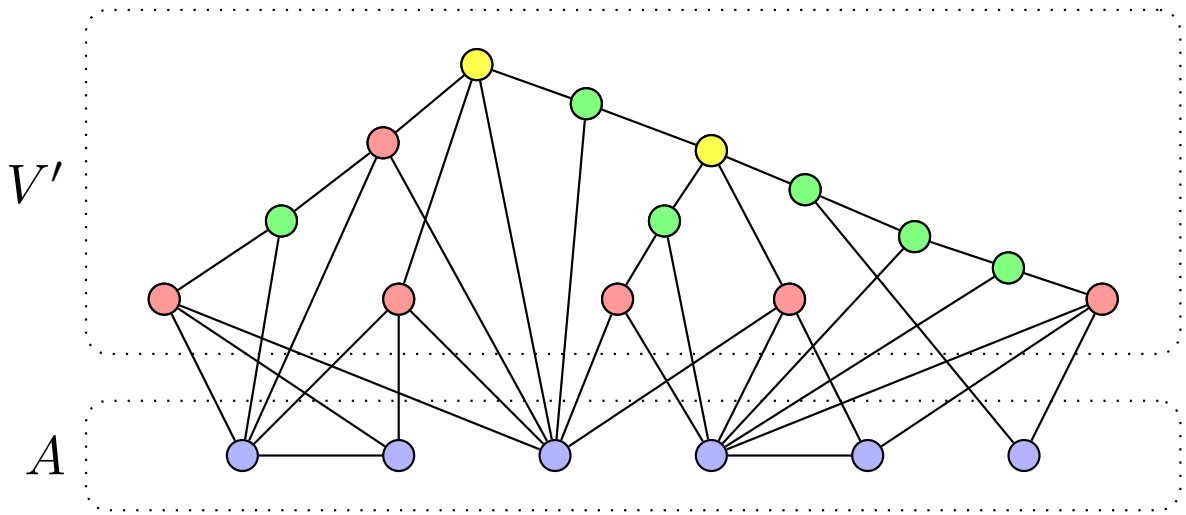


- $F := \{v \in V' || N(v) \cap A| \geq 2\}$
- $H := \{v \in V' \backslash F || N(v) \cap V'| \geq 3\}$
- $R := V' \backslash (F \cup H)$

Beschränkung von $|V'|$

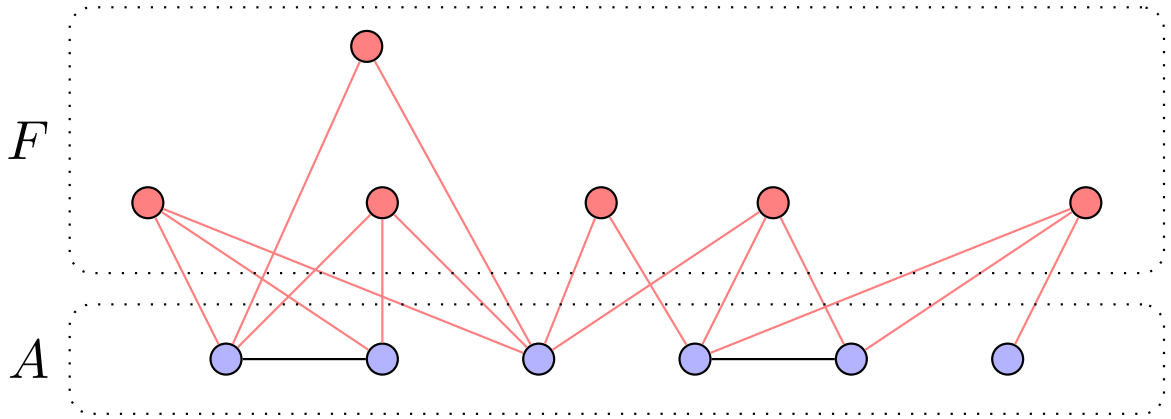
Idee:

- Teile V' in drei Mengen ein.
- Beschränke die Größe jeder Menge einzeln.

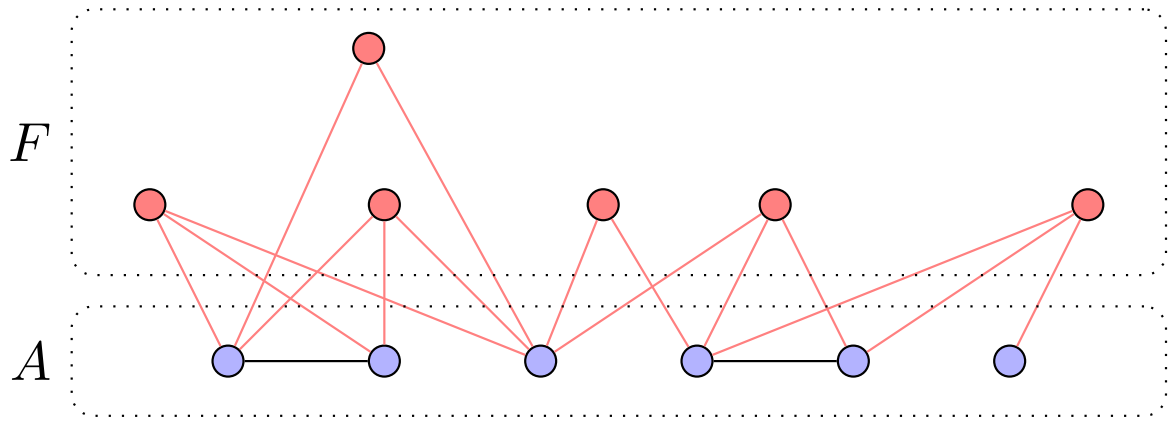


- $F := \{v \in V' || N(v) \cap A| \geq 2\}$
- $H := \{v \in V' \backslash F || N(v) \cap V'| \geq 3\}$
- $R := V' \backslash (F \cup H)$

Beschränkung von $|F|$

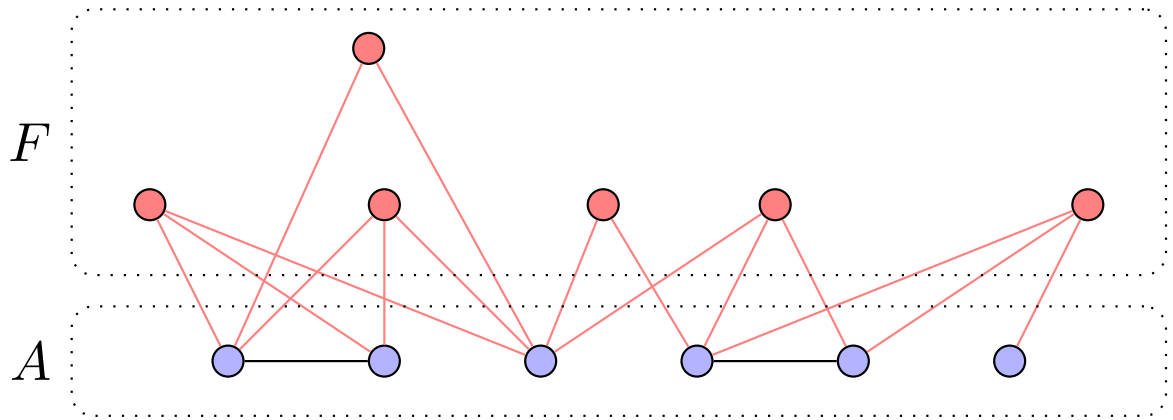


Beschränkung von $|F|$



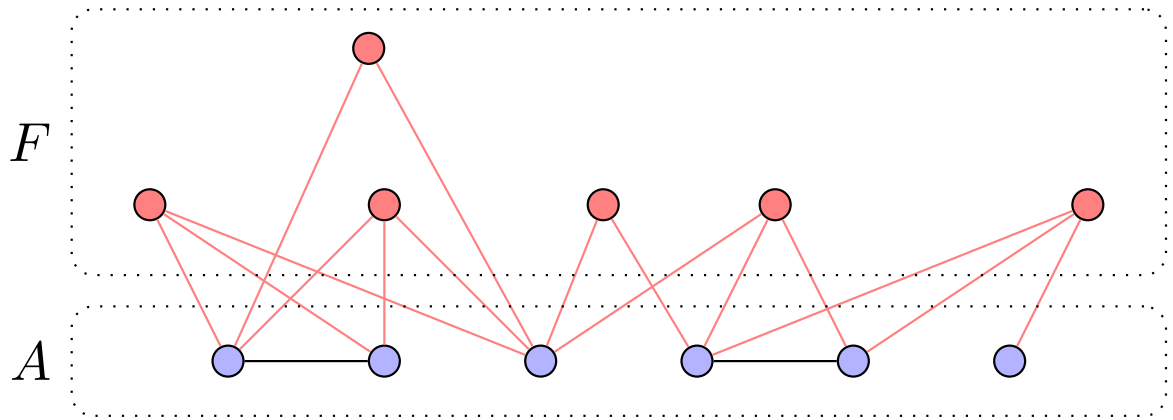
- F hat mind. 2 Nachbarn in A

Beschränkung von $|F|$



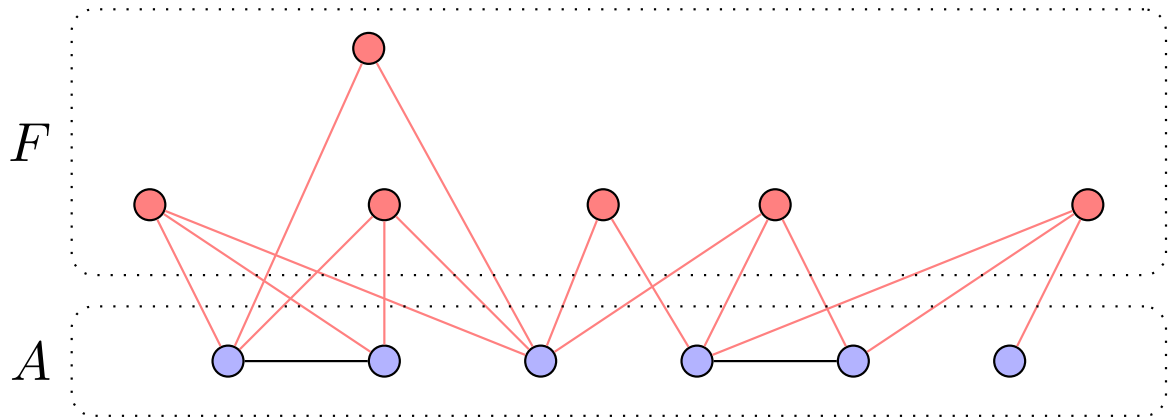
- F hat mind. 2 Nachbarn in A
- Ist $|F| = |A|$, so gibt es mind. 1 Kreis!

Beschränkung von $|F|$



- F hat mind. 2 Nachbarn in A
- Ist $|F| = |A|$, so gibt es mind. 1 Kreis!
  - Mit jedem Knoten in F mehr, kommt mind. 1 Kreis hinzu

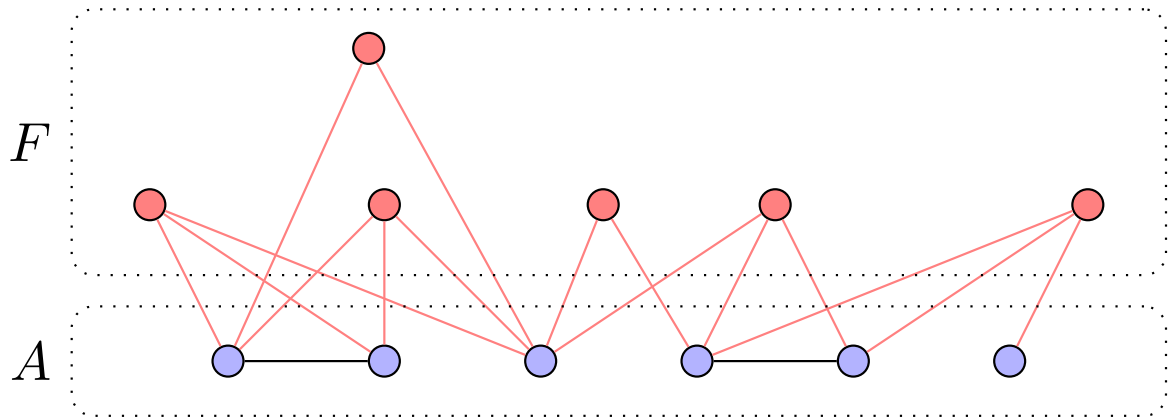
Beschränkung von $|F|$



- F hat mind. 2 Nachbarn in A
- Ist $|F| = |A|$, so gibt es mind. 1 Kreis!
  - Mit jedem Knoten in F mehr, kommt mind. 1 Kreis hinzu

  $\Rightarrow$ Ist $|F| = 2|A|$, so gibt es mind. $|A|$ Kreise

Beschränkung von $|F|$

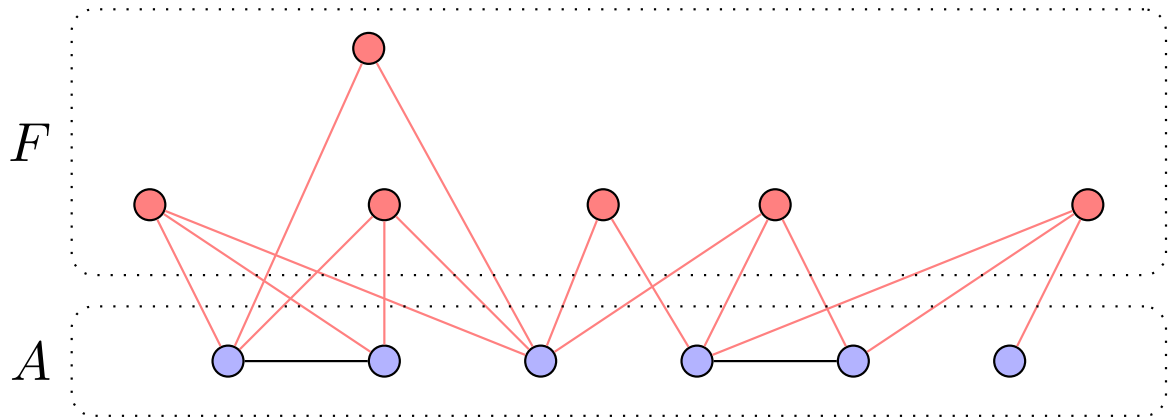


- F hat mind. 2 Nachbarn in A
- Ist $|F| = |A|$, so gibt es mind. 1 Kreis!
  - Mit jedem Knoten in F mehr, kommt mind. 1 Kreis hinzu

  $\Rightarrow$ Ist $|F| = 2|A|$, so gibt es mind. $|A|$ Kreise

  $\Rightarrow$ Wir müssen jedoch alle Kreise mit $|A| - 1$ Löschungen auflösen

Beschränkung von $|F|$

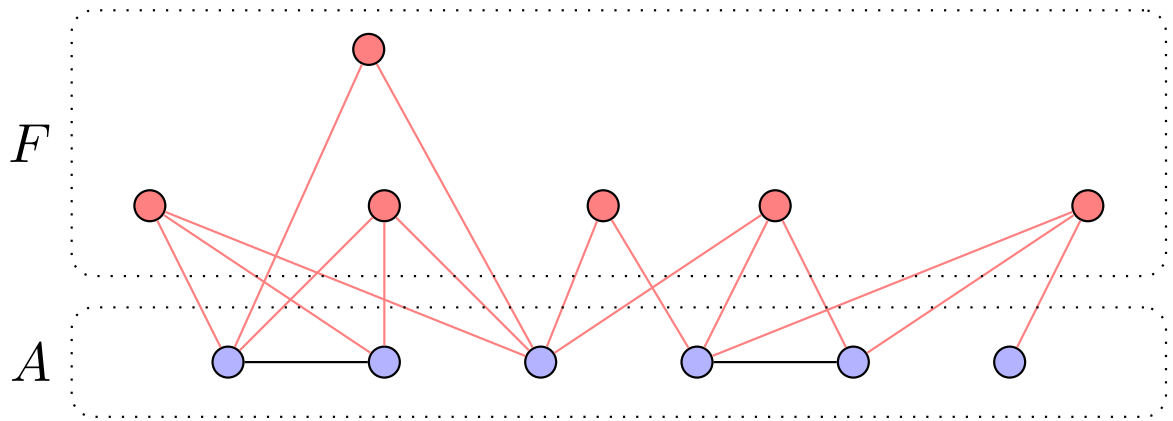


- F hat mind. 2 Nachbarn in A
- Ist $|F| = |A|$, so gibt es mind. 1 Kreis!
  - Mit jedem Knoten in F mehr, kommt mind. 1 Kreis hinzu

  $\Rightarrow$ Ist $|F| = 2|A|$, so gibt es mind. $|A|$ Kreise

  $\Rightarrow$ Wir müssen jedoch alle Kreise mit $|A| - 1$ Löschungen auflösen
    - Mit $|F| \geq 2|A|$ unmöglich

Beschränkung von $|F|$



- F hat mind. 2 Nachbarn in A
- Ist $|F| = |A|$, so gibt es mind. 1 Kreis!
  - Mit jedem Knoten in F mehr, kommt mind. 1 Kreis hinzu

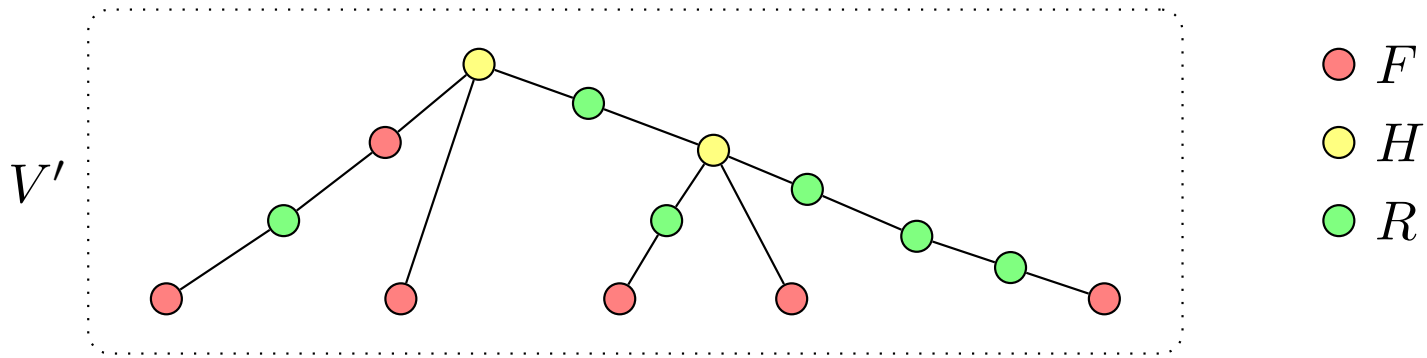
  $\Rightarrow$ Ist $|F| = 2|A|$, so gibt es mind. $|A|$ Kreise

  $\Rightarrow$ Wir müssen jedoch alle Kreise mit $|A| - 1$ Löschungen auflösen
    - Mit $|F| \geq 2|A|$ unmöglich

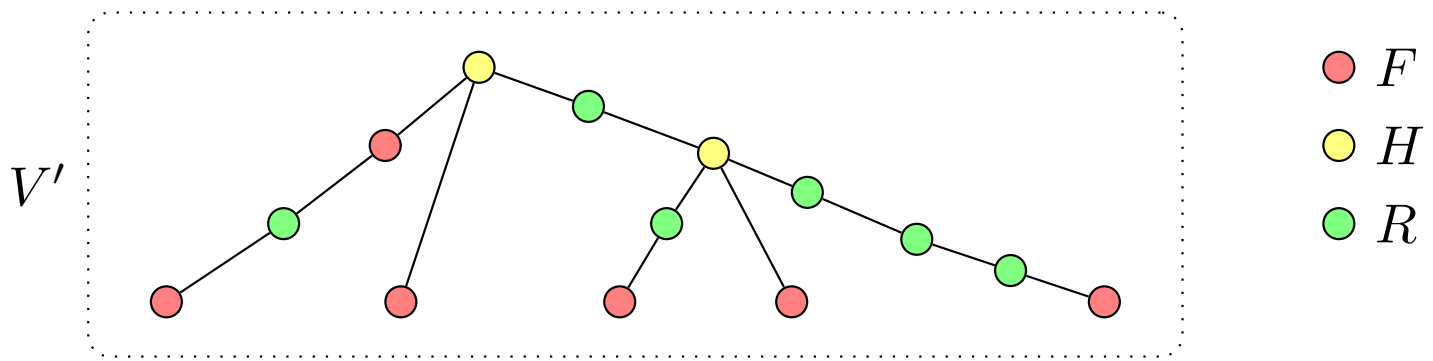
Daraus folgt:

$$|F| < 2|A|$$

Beschränkung von $|H|$

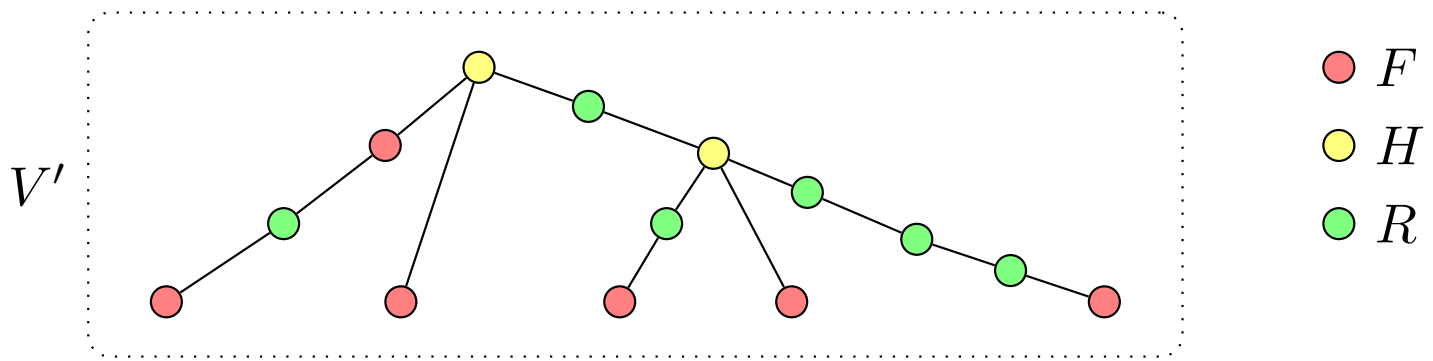


Beschränkung von $|H|$



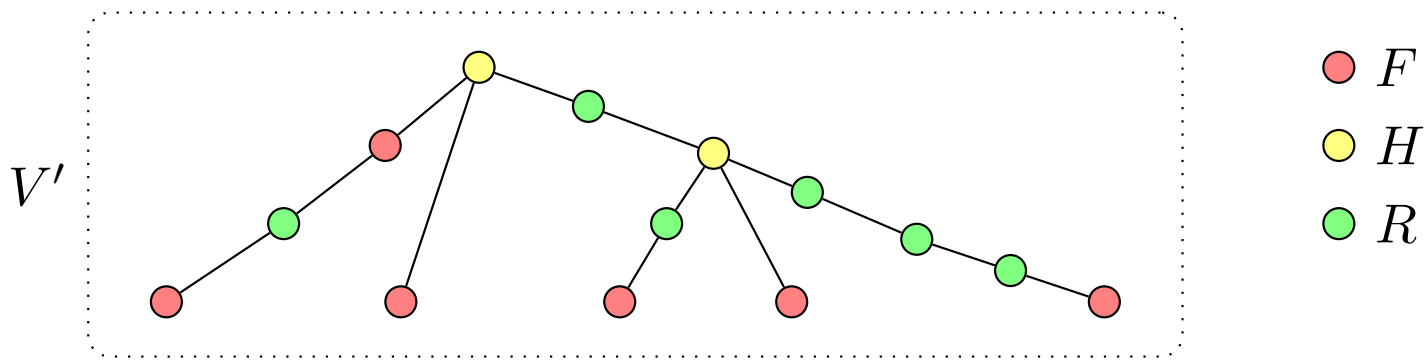
- V' ist ein Baum

Beschränkung von $|H|$



- V' ist ein Baum
- Alle Blätter sind aus F

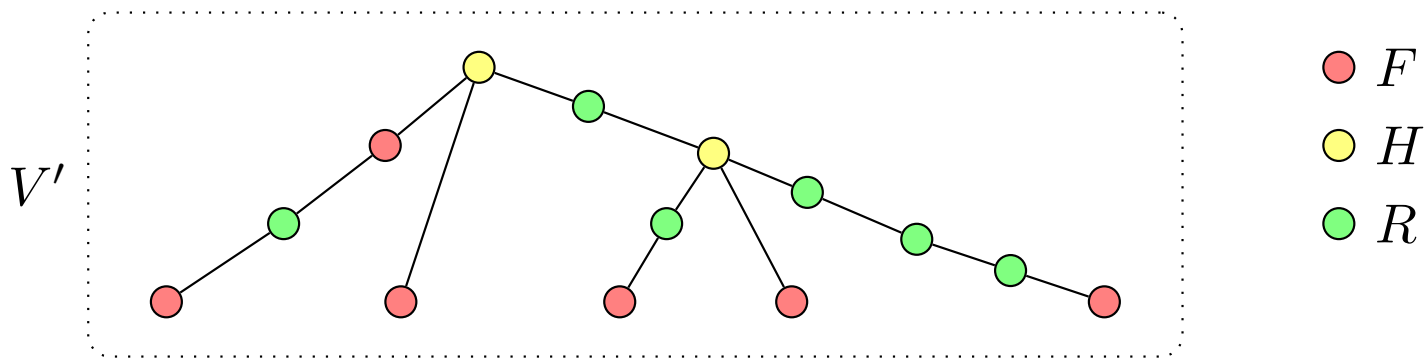
Beschränkung von $|H|$



- V' ist ein Baum
- Alle Blätter sind aus F

$\Rightarrow$ Daher kann es nicht mehr Knoten aus H geben, als aus F!

Beschränkung von $|H|$



- V' ist ein Baum
- Alle Blätter sind aus F

$\Rightarrow$ Daher kann es nicht mehr Knoten aus H geben, als aus F!

Also:

$$|H| < 2|A|$$

Beschränkung von $|R|$

Wie lässt sich $|R|$ beschränken?

Beschränkung von $|R|$

Wie lässt sich $|R|$ beschränken?

- Das ist ein wenig komplizierter...

Beschränkung von $|R|$

Wie lässt sich $|R|$ beschränken?

- Das ist ein wenig komplizierter...
- Daher hier nur das Ergebnis:

Beschränkung von $|R|$

Wie lässt sich $|R|$ beschränken?

- Das ist ein wenig komplizierter...
- Daher hier nur das Ergebnis:

$$|R| < 10|A|$$

Beschränkung von $|V'|$

Was haben wir erreicht?

Beschränkung von $|V'|$

Was haben wir erreicht?

- $|F| < 2|A|$

Beschränkung von $|V'|$

Was haben wir erreicht?

- $|F| < 2|A|$
- $|H| < 2|A|$

Beschränkung von $|V'|$

Was haben wir erreicht?

- $|F| < 2|A|$
- $|H| < 2|A|$
- $|R| < 10|A|$

Beschränkung von $|V'|$

Was haben wir erreicht?

- $|F| < 2|A|$
- $|H| < 2|A|$
- $|R| < 10|A|$

Also:

$$|V'| = |F| + |H| + |R| < 14|A|$$

Beschränkung von $|V'|$

Was haben wir erreicht?

- $|F| < 2|A|$
- $|H| < 2|A|$
- $|R| < 10|A|$

Also:

$$|V'| = |F| + |H| + |R| < 14|A|$$

Was bedeutet das?

Beschränkung von $|V'|$

Was haben wir erreicht?

- $|F| < 2|A|$
- $|H| < 2|A|$
- $|R| < 10|A|$

Also:

$$|V'| = |F| + |H| + |R| < 14|A|$$

Was bedeutet das?

- Wenn nach den Reduktionen $V' \geq 14|A|$ ist, so gibt es keine Lösung (für diese Partition)!

Laufzeit

1. Laufzeit für den Kompressionsschritt:

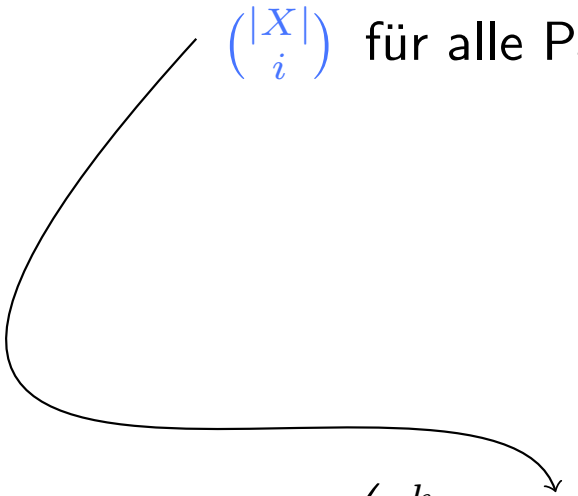
Laufzeit

1. Laufzeit für den Kompressionsschritt:

$$T = O\left(\sum_{i=0}^{k} \binom{|X|}{i} \cdot \left(O(m) + \binom{14 \cdot (|X| - i)}{|X| - i - 1} \cdot O(m)\right)\right)$$

Laufzeit

1. Laufzeit für den Kompressionsschritt:



$\binom{|X|}{i}$ für alle Partitionen mit $|B| = i$ und $|A| = |X| - i$

$$T = O\left(\sum_{i=0}^{k} \binom{|X|}{i} \cdot \left(O(m) + \binom{14 \cdot (|X| - i)}{|X| - i - 1} \cdot O(m)\right)\right)$$

Laufzeit

1. Laufzeit für den Kompressionsschritt:

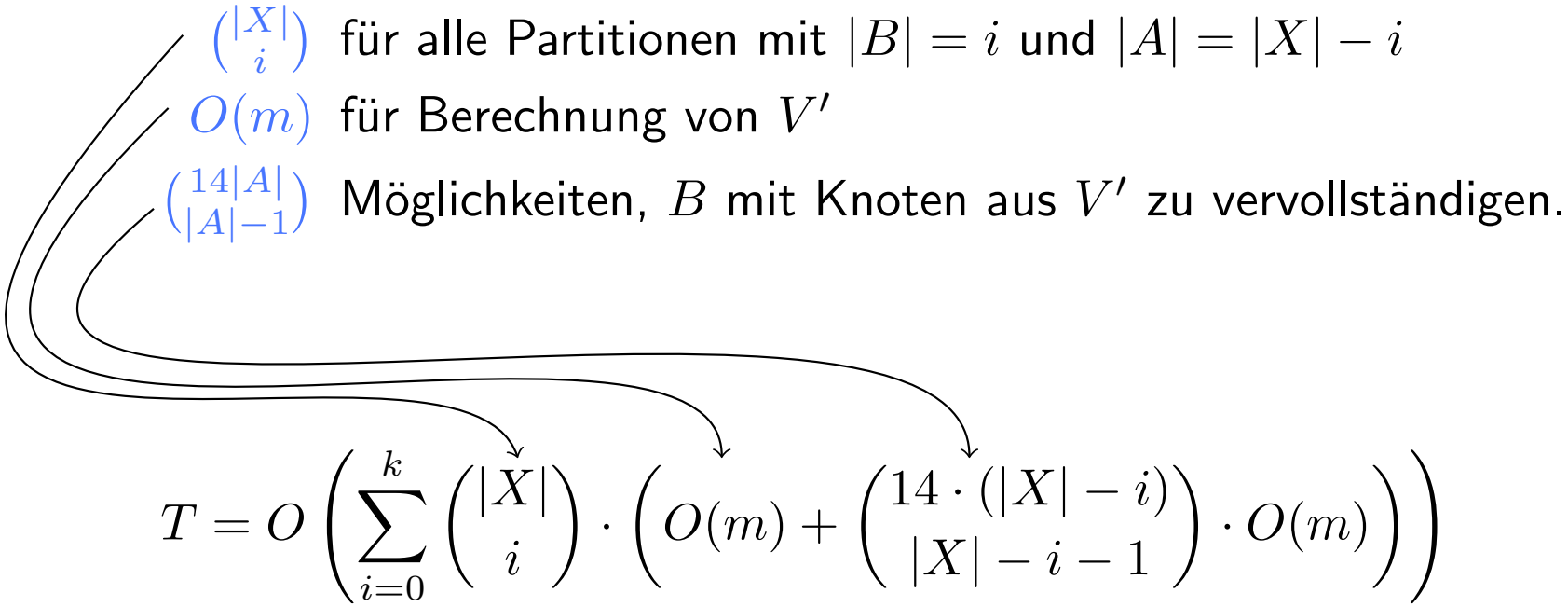
$\binom{|X|}{i}$ für alle Partitionen mit $|B| = i$ und $|A| = |X| - i$

$O(m)$ für Berechnung von V'

$$T = O\left(\sum_{i=0}^{k} \binom{|X|}{i} \cdot \left(O(m) + \binom{14 \cdot (|X| - i)}{|X| - i - 1} \cdot O(m)\right)\right)$$

Laufzeit

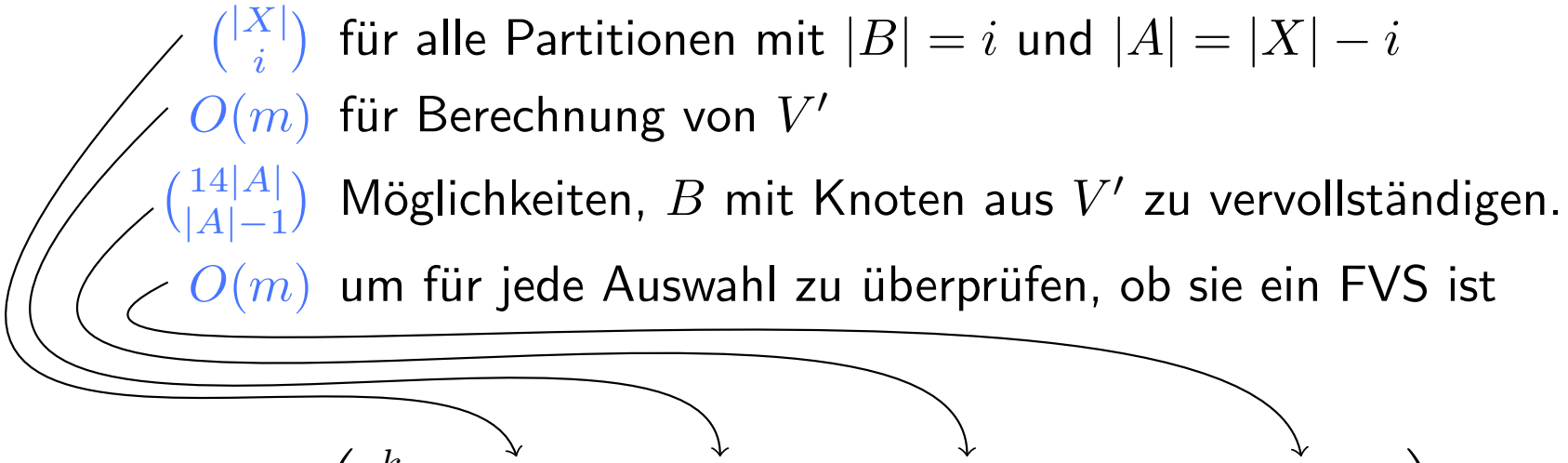
1. Laufzeit für den Kompressionsschritt:



$$T = O\left(\sum_{i=0}^{k} \binom{|X|}{i} \cdot \left(O(m) + \binom{14 \cdot (|X| - i)}{|X| - i - 1} \cdot O(m)\right)\right)$$

Laufzeit

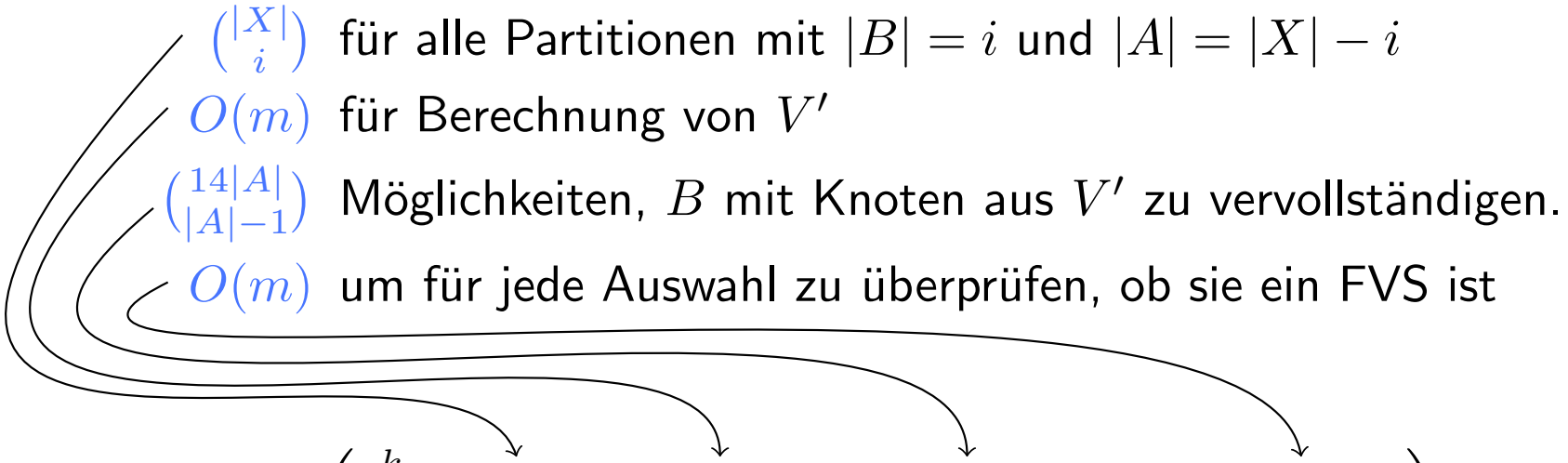
1. Laufzeit für den Kompressionsschritt:



$$T = O\left(\sum_{i=0}^{k} \binom{|X|}{i} \cdot \left(O(m) + \binom{14 \cdot (|X| - i)}{|X| - i - 1} \cdot O(m)\right)\right)$$

Laufzeit

1. Laufzeit für den Kompressionsschritt:



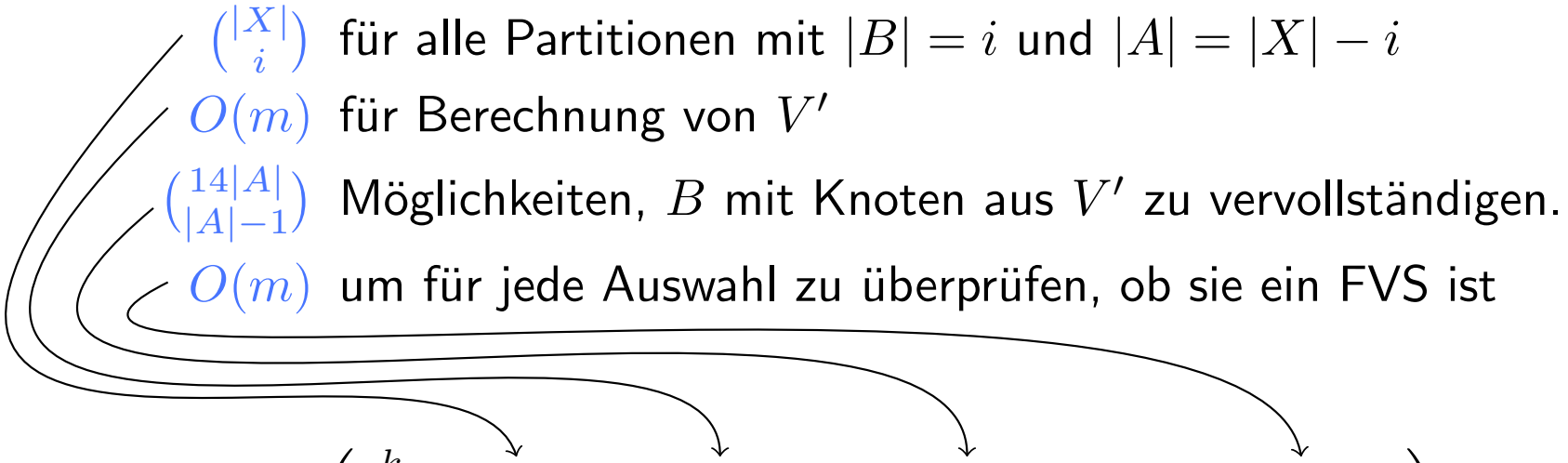
- $\binom{|X|}{i}$ für alle Partitionen mit $|B| = i$ und $|A| = |X| - i$
- $O(m)$ für Berechnung von V'
- $\binom{14|A|}{|A|-1}$ Möglichkeiten, B mit Knoten aus V' zu vervollständigen.
- $O(m)$ um für jede Auswahl zu überprüfen, ob sie ein FVS ist

$$T = O\left(\sum_{i=0}^{k} \binom{|X|}{i} \cdot \left(O(m) + \binom{14 \cdot (|X| - i)}{|X| - i - 1} \cdot O(m)\right)\right)$$

$$= \ldots$$

Laufzeit

1. Laufzeit für den Kompressionsschritt:



$$T = O\left(\sum_{i=0}^{k} \binom{|X|}{i} \cdot \left(O(m) + \binom{14 \cdot (|X| - i)}{|X| - i - 1} \cdot O(m)\right)\right)$$

$$= \ldots$$

$$= O(37.7^k \cdot m)$$

Laufzeit gesamt

Zum Lösen von Feedback Vertex Set kommen wir so auf folgende Laufzeit:

Laufzeit gesamt

Zum Lösen von Feedback Vertex Set kommen wir so auf folgende Laufzeit:

$$O(c^k \cdot m \cdot n) \qquad c = 37.7$$

Laufzeit gesamt

Zum Lösen von Feedback Vertex Set kommen wir so auf folgende Laufzeit:

$$O(c^k \cdot m \cdot n) \qquad c = 37.7$$

Mit einer verfeinerten Analyse (nach ähnlichem Ansatz) kann man folgende Laufzeit zeigen:

Laufzeit gesamt

Zum Lösen von Feedback Vertex Set kommen wir so auf folgende Laufzeit:

$$O(c^k \cdot m \cdot n) \qquad c = 37.7$$

Mit einer verfeinerten Analyse (nach ähnlichem Ansatz) kann man folgende Laufzeit zeigen:

$$O(c^k \cdot n^3) \qquad c = 10.6$$

Ergebnisse Iterative Compression

Vertex Cover:

Ergebnisse Iterative Compression

Vertex Cover:

- Ansatz nicht praxisrelevant!

Ergebnisse Iterative Compression

Vertex Cover:

- Ansatz nicht praxisrelevant!
- Laufzeit: $O(2^k \cdot m \cdot n)$

Ergebnisse Iterative Compression

Vertex Cover:

- Ansatz nicht praxisrelevant!
- Laufzeit: $O(2^k \cdot m \cdot n)$

Feedback Vertex Set:

Ergebnisse Iterative Compression

Vertex Cover:

- Ansatz nicht praxisrelevant!
- Laufzeit: $O(2^k \cdot m \cdot n)$

Feedback Vertex Set:

- Erster Algorithmus, der die kombinatorische Explosion in der Form von c^k hat.

Ergebnisse Iterative Compression

Vertex Cover:

- Ansatz nicht praxisrelevant!
- Laufzeit: $O(2^k \cdot m \cdot n)$

Feedback Vertex Set:

- Erster Algorithmus, der die kombinatorische Explosion in der Form von c^k hat.
- Normale Analyse: $O(37.7^k \cdot m \cdot n)$

Ergebnisse Iterative Compression

Vertex Cover:

- Ansatz nicht praxisrelevant!
- Laufzeit: $O(2^k \cdot m \cdot n)$

Feedback Vertex Set:

- Erster Algorithmus, der die kombinatorische Explosion in der Form von c^k hat.
- Normale Analyse: $O(37.7^k \cdot m \cdot n)$
- Verfeinerte Analyse: $O(10.6^k \cdot n^3)$

Further Advanced Techniques - Fortgeschrittene Techniken

Further Advanced Techniques - Fortgeschrittene Techniken

- Konzepte und Techniken aus anderen Bereichen werden genutzt:

Further Advanced Techniques - Fortgeschrittene Techniken

- Konzepte und Techniken aus anderen Bereichen werden genutzt:
  - Color-Coding: Dynamische Programmierung

Further Advanced Techniques - Fortgeschrittene Techniken

- Konzepte und Techniken aus anderen Bereichen werden genutzt:
  - Color-Coding: Dynamische Programmierung
  - Iterative Compression: Datenreduktion

Further Advanced Techniques - Fortgeschrittene Techniken

- Konzepte und Techniken aus anderen Bereichen werden genutzt:
  - Color-Coding: Dynamische Programmierung
  - Iterative Compression: Datenreduktion
- Es werden neue Möglichkeiten und Ansätze zum Entwickeln von parametrisierten Algorithmen eröffnet.

Literatur I


Noga Alon, Raphael Yuster, and Uri Zwick.
Color-coding.
J. ACM, 42(4):844–856, 1995.


Hannes Moser.
Iterative compression for solving hard network problems.
Colloquium of the DFG-Schwerpunkt 1126 "Algorithms on large and complex networks", 2006.


Rolf Niedermeier.
Invitation to Fixed-Parameter Algorithms.
Oxford University Press, 2006.


Rolf Niedermeier and Jiong Guo.
Fixed-parameter algorithms - slides.
Winter Term 2005/2006.

Literatur II


B. Reed, K. Smith, and A. Vetta.
Finding odd cycle transversals.
Operations Research Letters, 32:299–301, 2004.


Jeanette P. Schmidt and Alan Siegel.
The spatial complexity of oblivious k-probe hash functions.
SIAM Journal on Computing, 19(5):775–786, 1990.

Vielen Dank für die Aufmerksamkeit!