CoNe Freiburg

Informatik III

4.3 Weitere NP-Vollständige Probleme und Approximation

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik
Wintersemester 2007/08

Komplexitätstheorie

Vertex Cover ist NP-vollständig

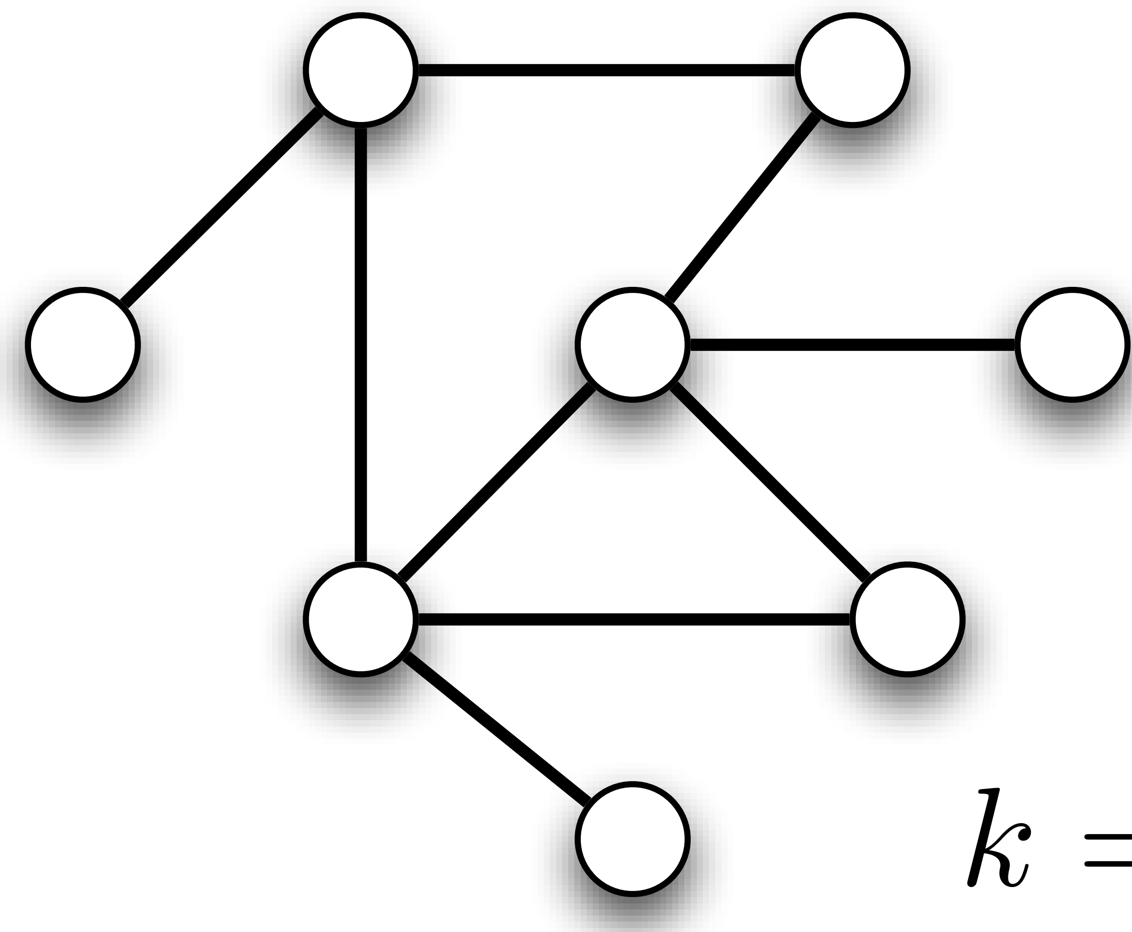
2

Wiederholung: 3-SAT

- **Definition:**
  - 3-SAT = { ψ | ψ ist eine erfüllbare Formel in 3-CNF}
  - z.B.: $\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$
- **3-SAT ist NP-vollständig**

VERTEX-COVER

- **Definition:**
  - Eine *Knotenüberdeckung* eines ungerichteten Graphs G
  - ist eine Teilmenge seiner Knoten, so dass jede Kante von
  - einem dieser Knoten berührt wird.
- **VERTEX-COVER = { (G, *k*) | G ist ein ungerichteter Graph mit einer *k*-Knotenüberdeckung }**

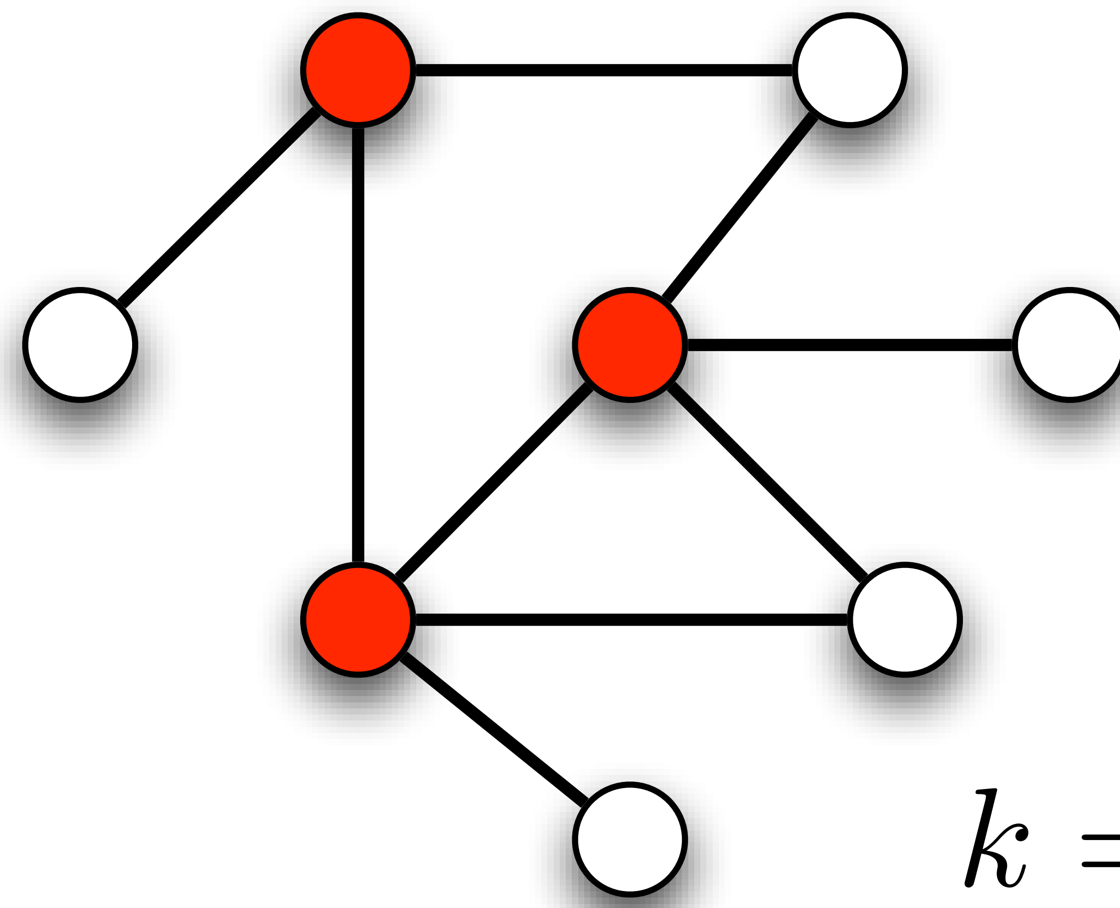


$$k = 3$$

VERTEX-COVER ist in NP

Beweis:

- **k-Knotenüberdeckung ist Zertifikat c**
  - Größe ist polynomiell in Eingabelänge
- **Verifizierer A(G=(V, E), k, c)**
  - Prüfe, ob c Kodierung von $U \subseteq V$, wobei $|U| \leq k$
  - Für alle Knoten $u \in U$ markiere alle Kanten $\{a, b\} \in E$ mit $u \in \{a, b\}$
  - Sind alle Kanten markiert, akzeptiere. Sonst verwerfe.
  - Laufzeit von A polynomiell in der Eingabelänge



$$k = 3$$

VERTEX-COVER ist NP-schwierig

- **Beweis durch 3-SAT $\leq_{m,p}$ VERTEX-COVER**
  - Idee:

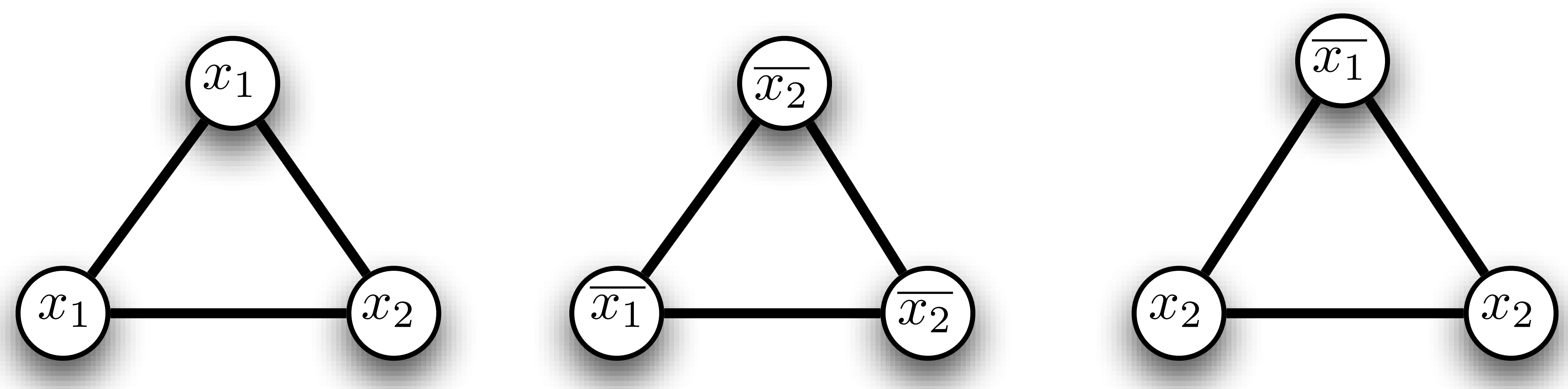
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

  - Jede Variable aus ψ abbilden auf Knotenpaar, das für positive bzw. negative Belegung steht:

3-SAT $\leq_{m,p}$ VERTEX-COVER

- Jede Klausel aus ψ abbilden auf Knotentripel, die den Literalen entsprechen

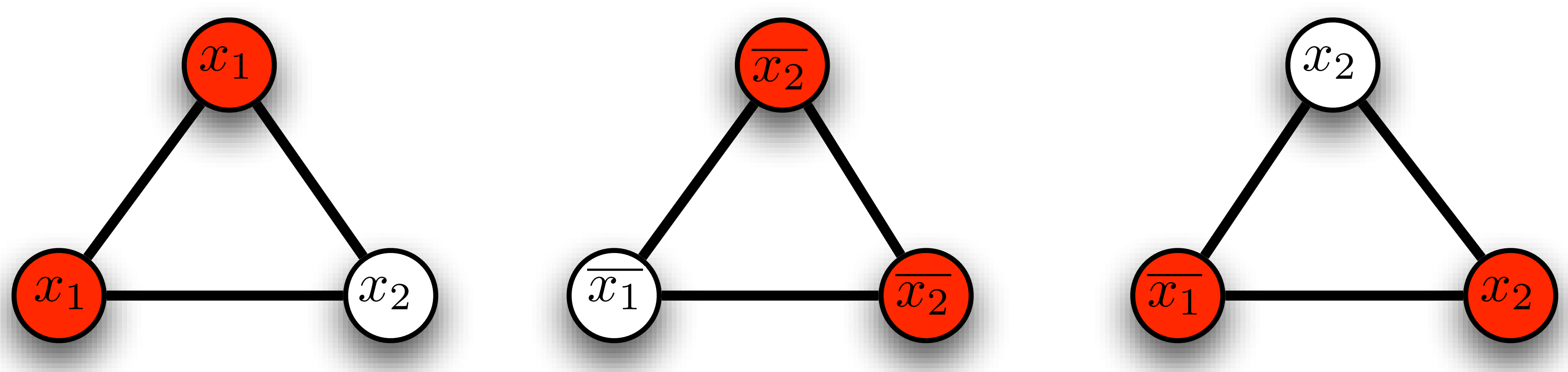
$$\psi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor x_2)$$



3-SAT $\leq_{m,p}$ VERTEX-COVER

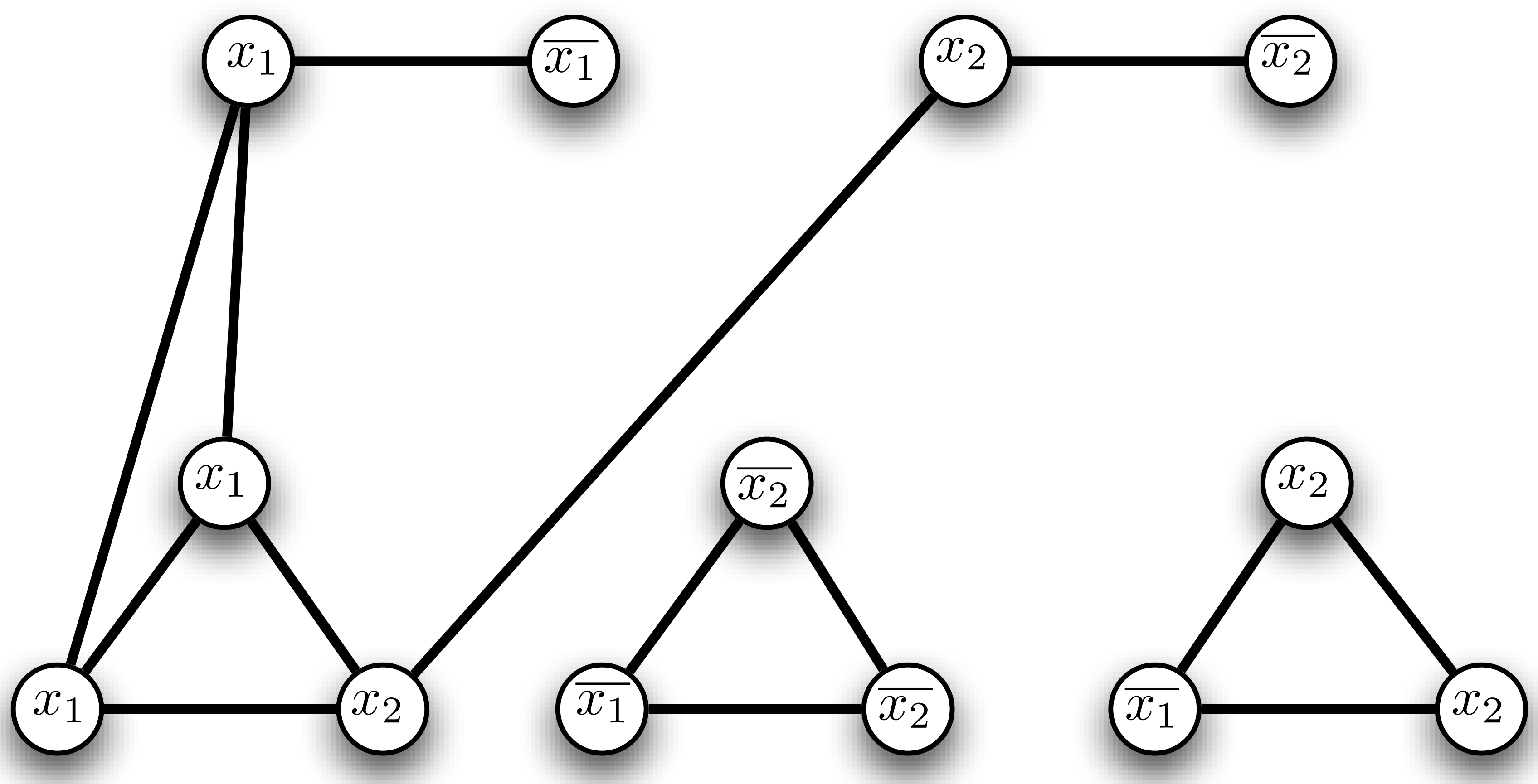
- ψ habe m Variablen und n Klauseln. Wähle k = m + 2n
- k-Knotenüberdeckung z.B.

$$k = 8$$



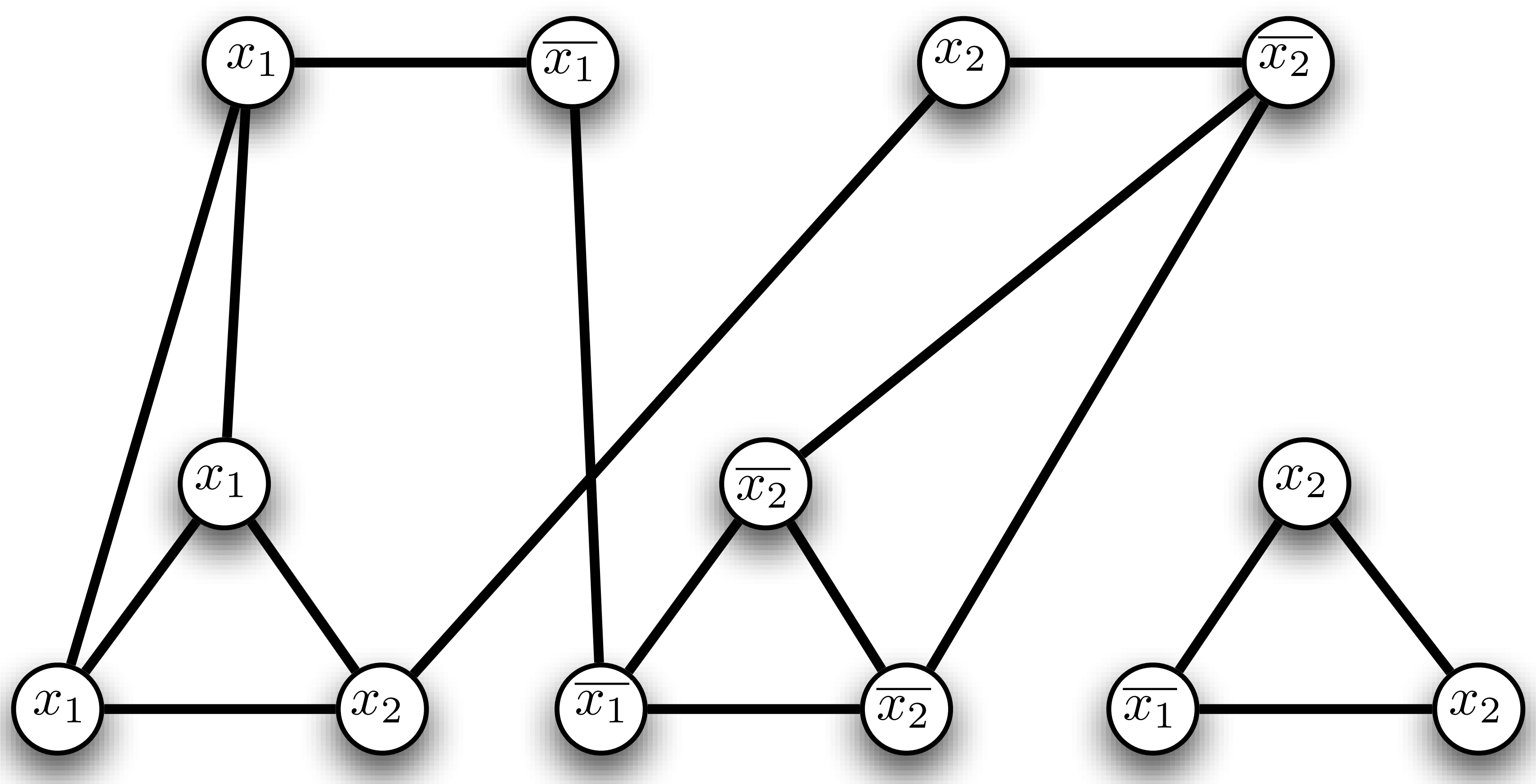
3-SAT $\leq_{m,p}$ VERTEX-COVER

- Verbinde ‚Literale' der Knotentripel mit den entsprechenden Knoten der Variablen-Knotenpaare



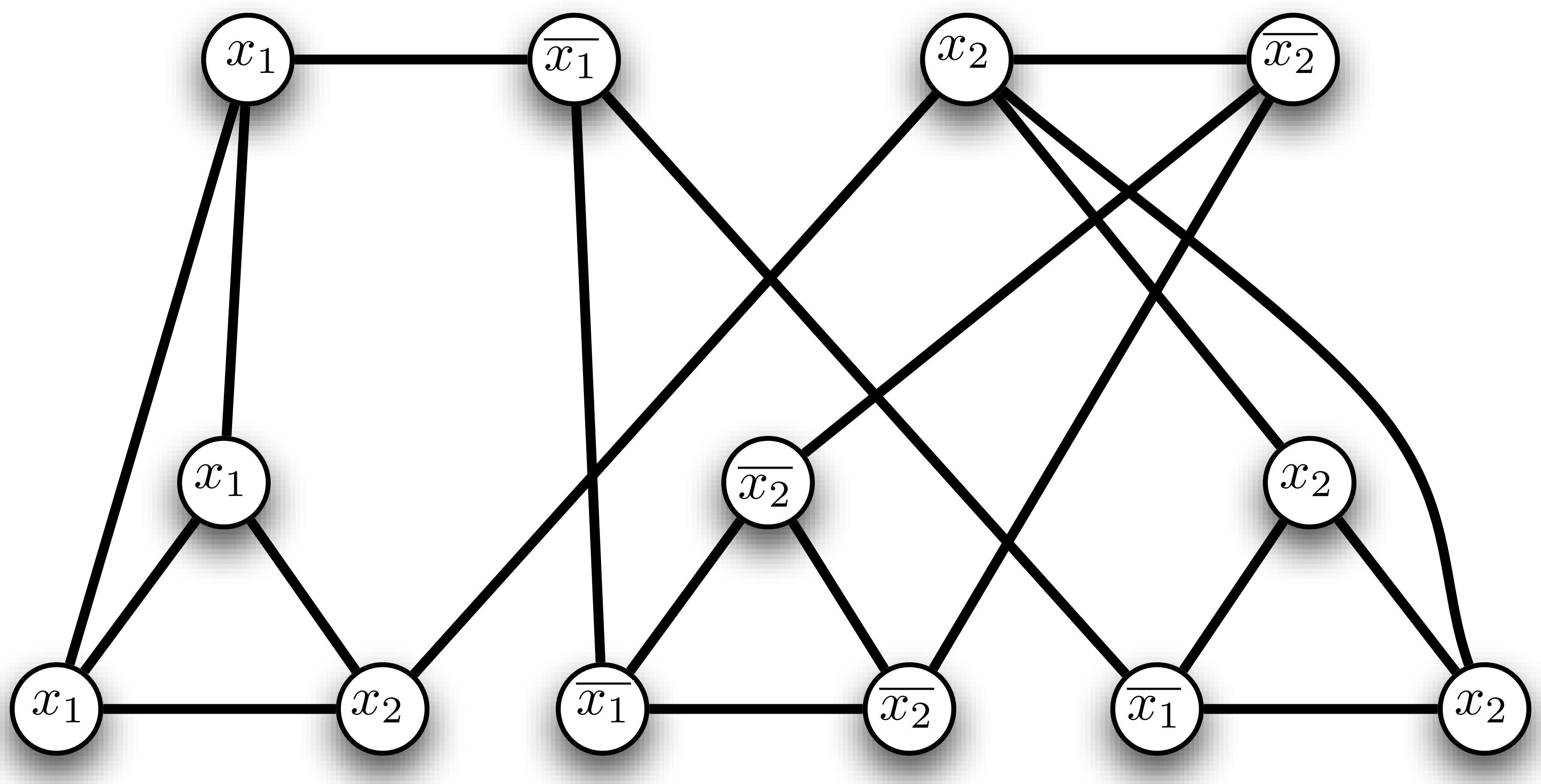
3-SAT $\leq_{m,p}$ VERTEX-COVER

- Verbinde ‚Literale' der Knotentripel mit den entsprechenden Knoten der Variablen-Knotenpaare



VERTEX-COVER

- Verbinde ‚Literale' der Knotentripel mit den entsprechenden Knoten der Variablen-Knotenpaare

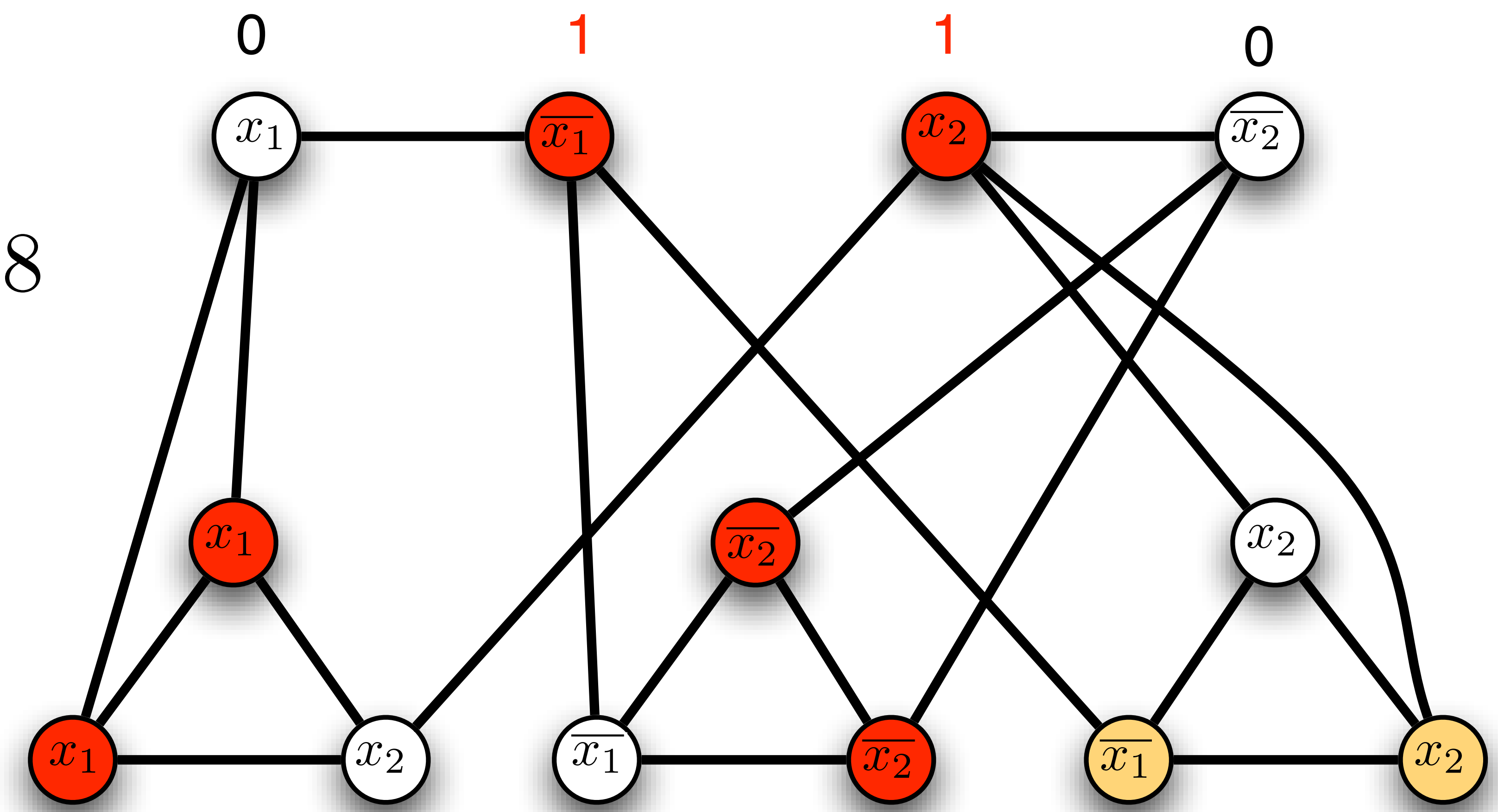


3-SAT $\leq_{m,p}$ VERTEX-COVER

- **ψ ist erfüllbar genau dann, wenn G eine k-Knoten-überdeckung hat**

$x_1 = 0$ $\qquad$ $x_2 = 1$

$k = 8$



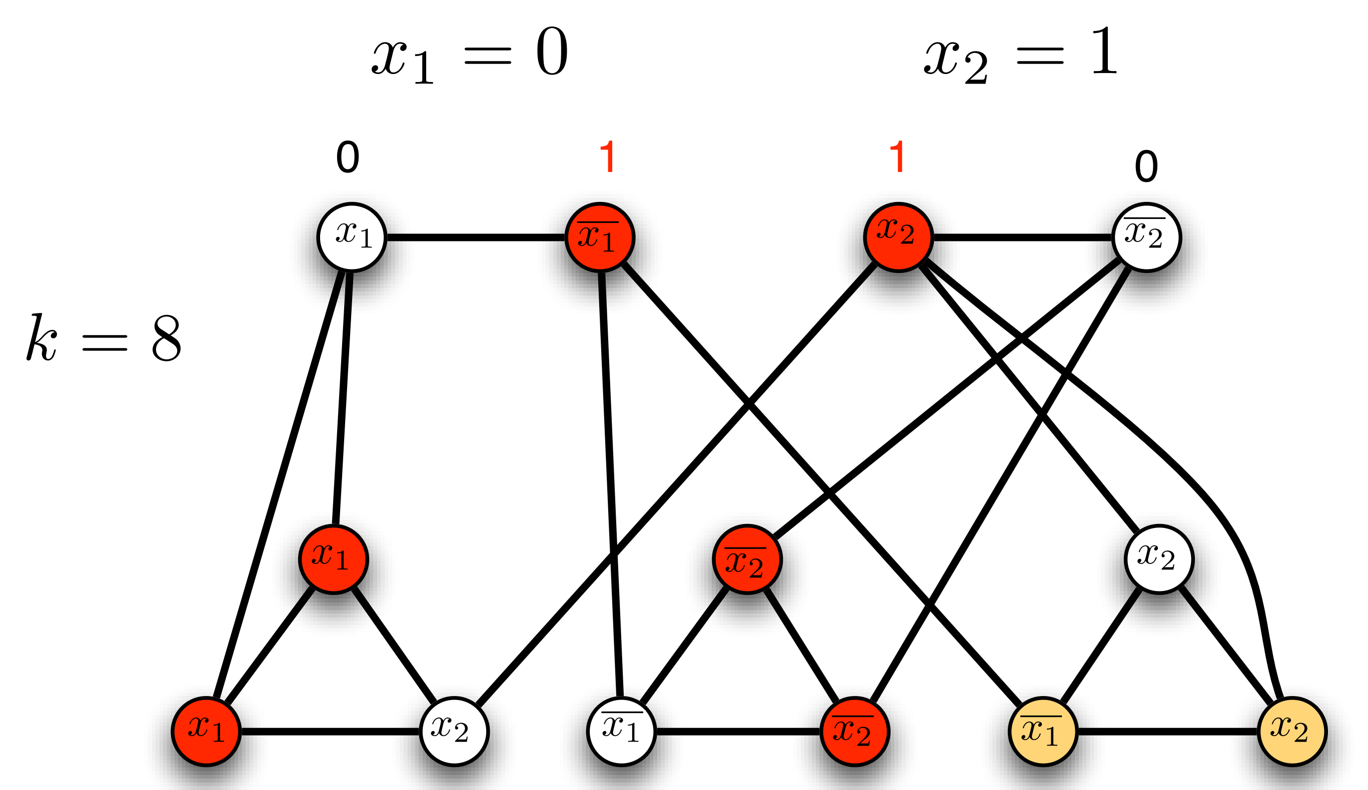
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

0 0 1 1 0 0 1 1 1

3-SAT $\leq_{m,p}$ VERTEX-COVER

- **ψ ist erfüllbar → G hat eine *k*-Knotenüberdeckung**
  - Wähle die Knotenüberdeckung wie folgt:
  - Für jede in der erfüllenden Belegung von ψ mit „wahr“ belegte Variable x_i wähle im entsprechenden Knotenpaar in G den Knoten x_i; falls x_i mit „falsch“ belegt ist den entsprechenden negierten Knoten.
  - ψ ist erfüllbar
    - → jede Klausel von ψ ist erfüllbar
    - → jede Klausel enthält maximal zwei falsche Literale. Wähle diese Knoten. (Falls weniger als zwei Literale falsch sind, fülle mit beliebigen wahren Literalen auf.)

- Gewählte Knotenüberdeckung hat offensichtlich Größe *k*

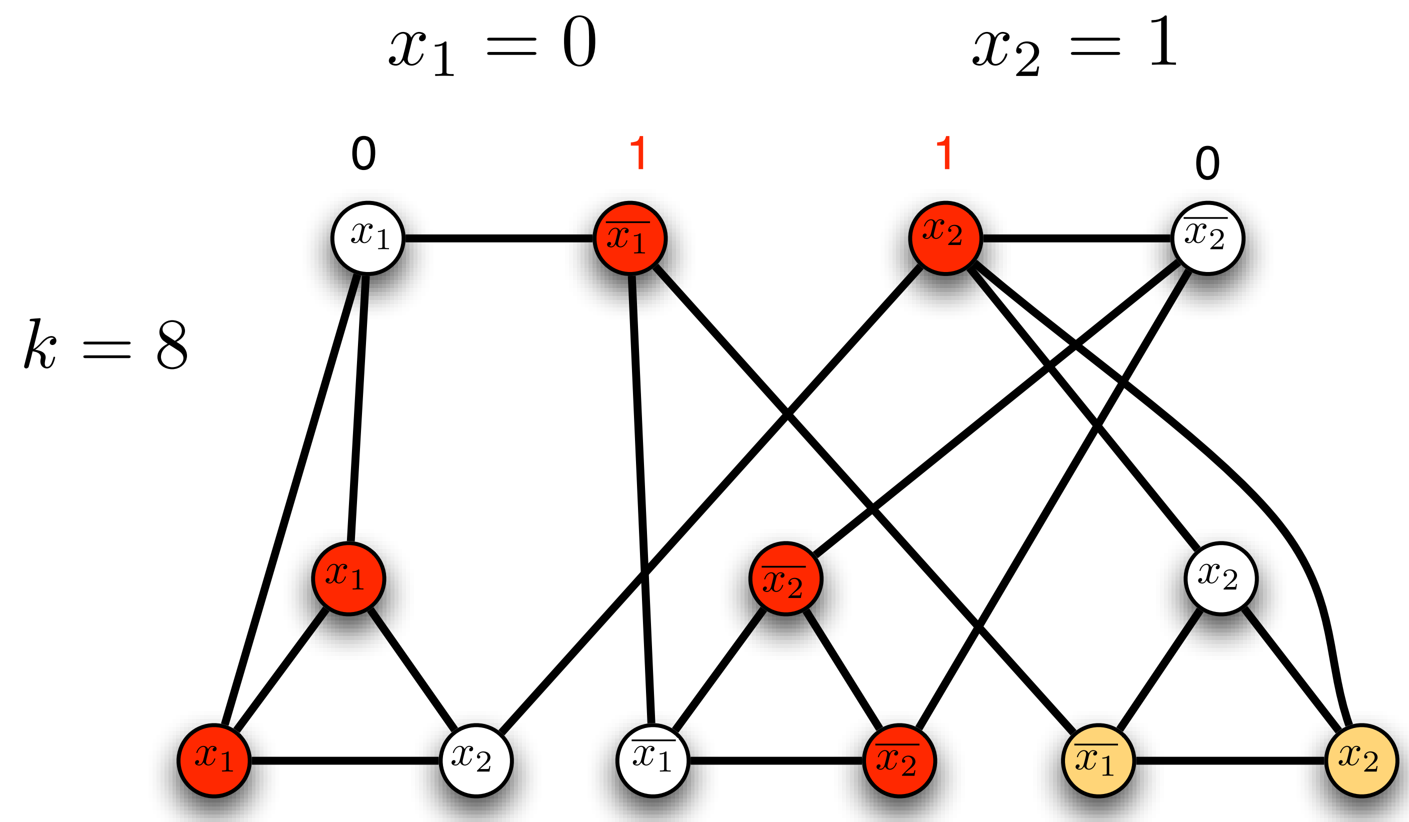


$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

0 0 1 1 0 0 1 1 1

VERTEX-COVER

- Gewählte k-Knotenüberdeckung berührt alle Kanten:
  - Kanten der Knotenpaare, da je einer der beiden gewählt
  - Kanten der Knotentripel, da je zwei der drei gewählt
  - Verbindungskanten zu erfüllten Literalen in Variablen-Knotenpaaren abgedeckt
  - Verbindungskanten zu nicht erfüllten Literalen über Knotentripel abgedeckt



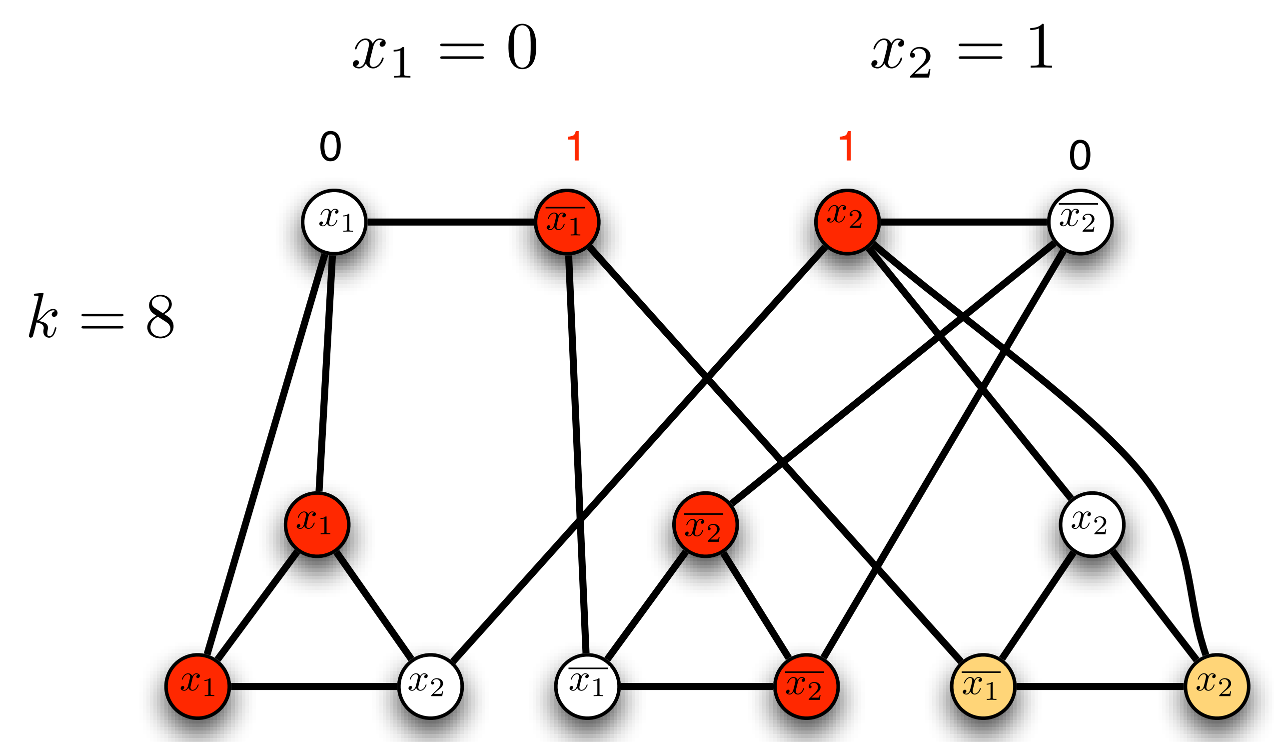
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

0 0 1 1 0 0 1 1 1

VERTEX-COVER

- **G hat eine *k*-Knotenüberdeckung → ψ ist erfüllbar**
  - In jedem Variablen-Knotenpaar muss genau ein Knoten der Überdeckung angehören. Belege die Variable in ψ dementsprechend.
  - In jedem Klausel-Knotentripel müssen genau zwei Knoten Teil der Überdeckung sein. Die zu dem dritten Knoten gehörende Verbindungskante muss daher durch einen Variablen-Knoten überdeckt sein
    - → das entsprechende Literal ist wahr
    - → die Klausel ist erfüllt

- **Theorem:**
  - VERTEX-COVER ist NP-vollständig



$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

0 0 1 1 0 0 1 1 1

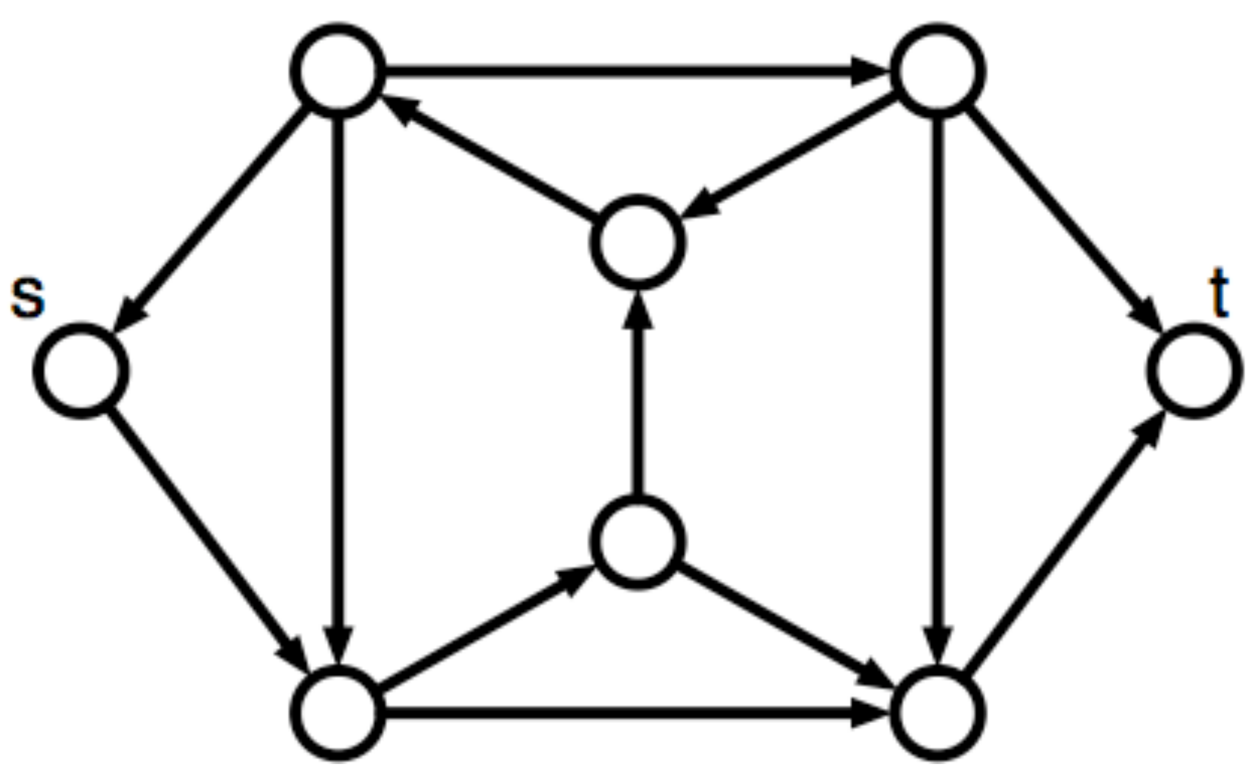
Komplexitätstheorie

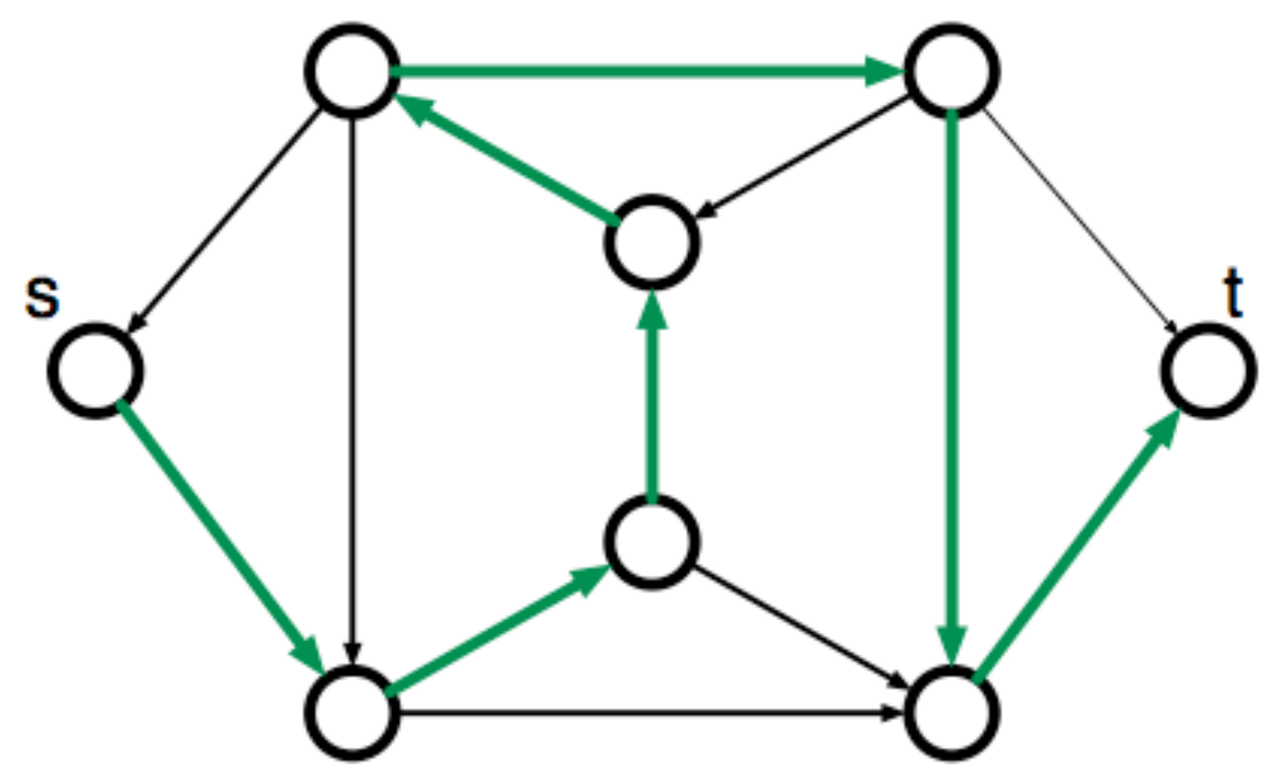
Hamilton ist NP-vollständig

16

Hamiltonsche Pfade

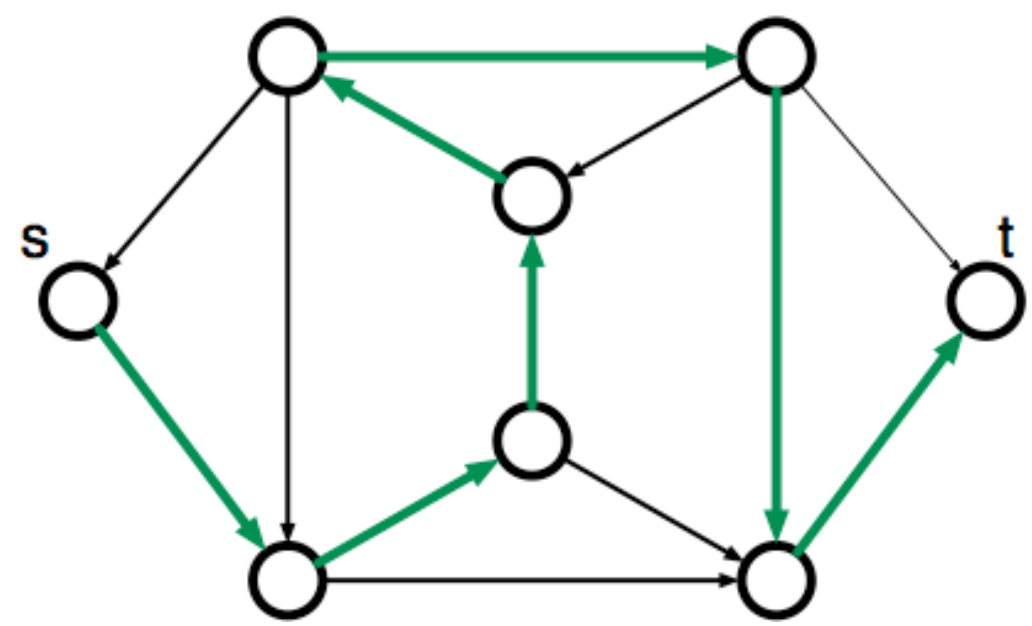
- **Definition:**
  - HAMPATH = { (G, *s*, *t*) | Der gerichtete Graph G enthält einen Weg von *s* nach *t*, der jeden Knoten genau einmal besucht. }





Hamiltonsche Pfade

- **Theorem:**
  - HAMPATH ist NP-vollständig
- **Beweis:**
  - HAMPATH ∈ NP:
    - Hamiltonscher Pfad $(s, v_1, v_2, ..., v_{n-2}, t)$ dient als Zertifikat c (Größe offensichtlich polynomiell in Eingabelänge)
    - Verifizierer A(G=(V, E), s, t, c)
      - Prüfe, ob c Kodierung einer Permutation der Knoten $(s, v_1, v_2, ..., v_{n-2}, t)$ ist
      - Für je zwei aufeinander folgende Knoten $(x_1, x_2) \in c$, prüfe, ob es eine gerichtete Kante $(x_1, x_2) \in E$ gibt. Falls nicht, verwerfe.
      - Akzeptiere.
    - Laufzeit von A polynomiell in der Eingabelänge
  - z.z.: HAMPATH ist NP-schwierig

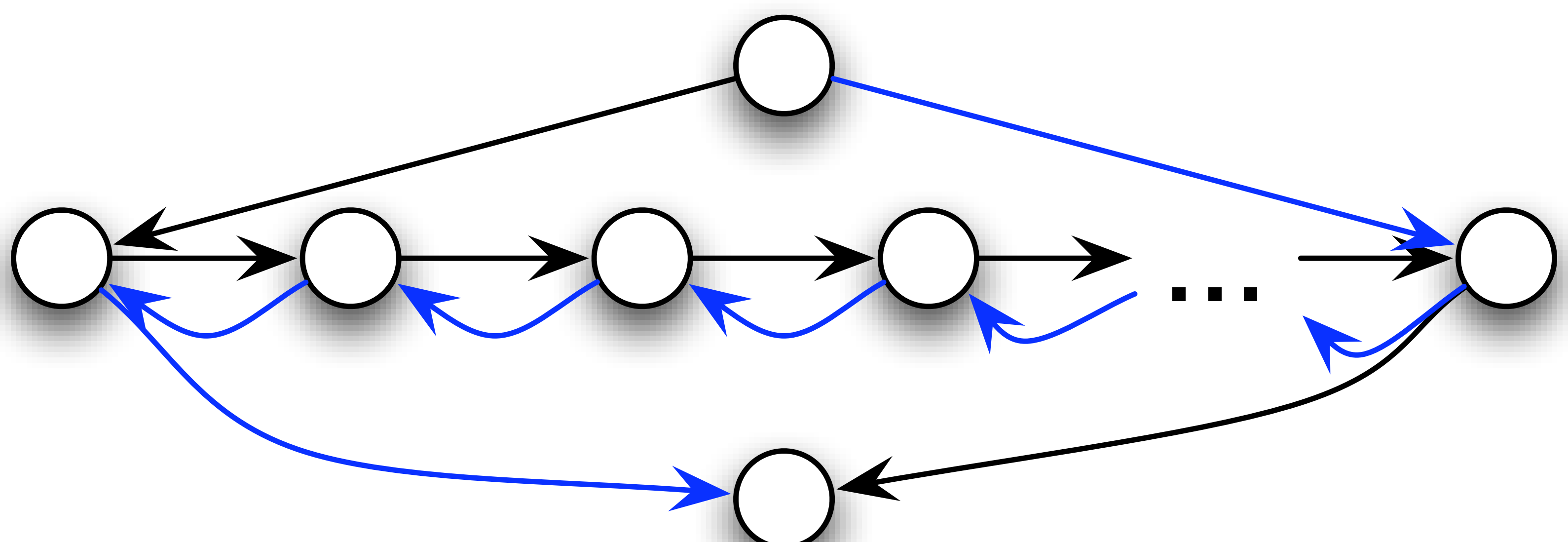


Hamiltonsche Pfade

- **HAMPATH ist NP-schwierig**
- **Beweis durch 3-SAT ≤m,p HAMPATH**
  - Idee:

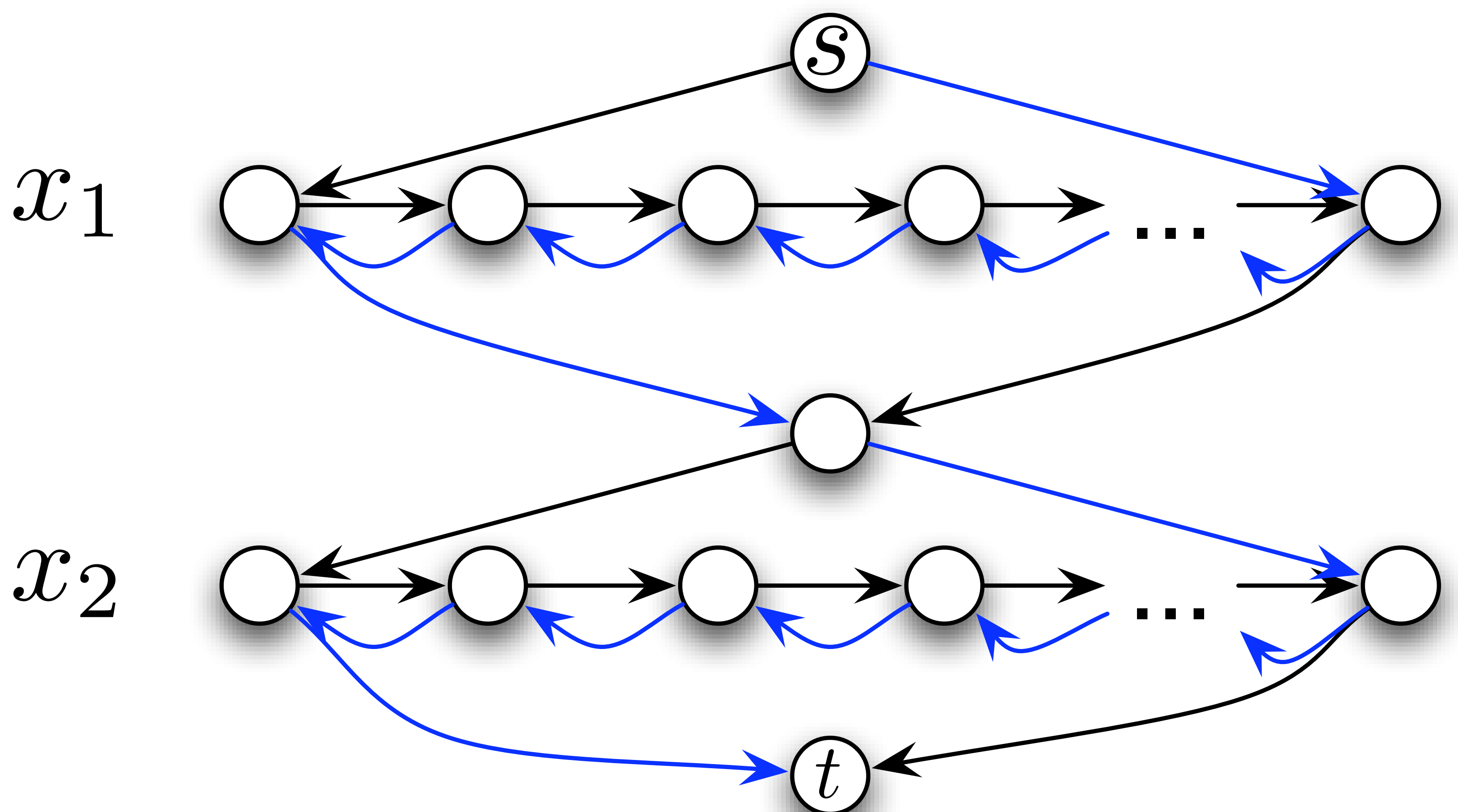
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

  - Jede Variable x_i aus ψ abbilden auf rautenartige Knotenstruktur mit einer horizontalen Knotenzeile



Hamiltonsche Pfade

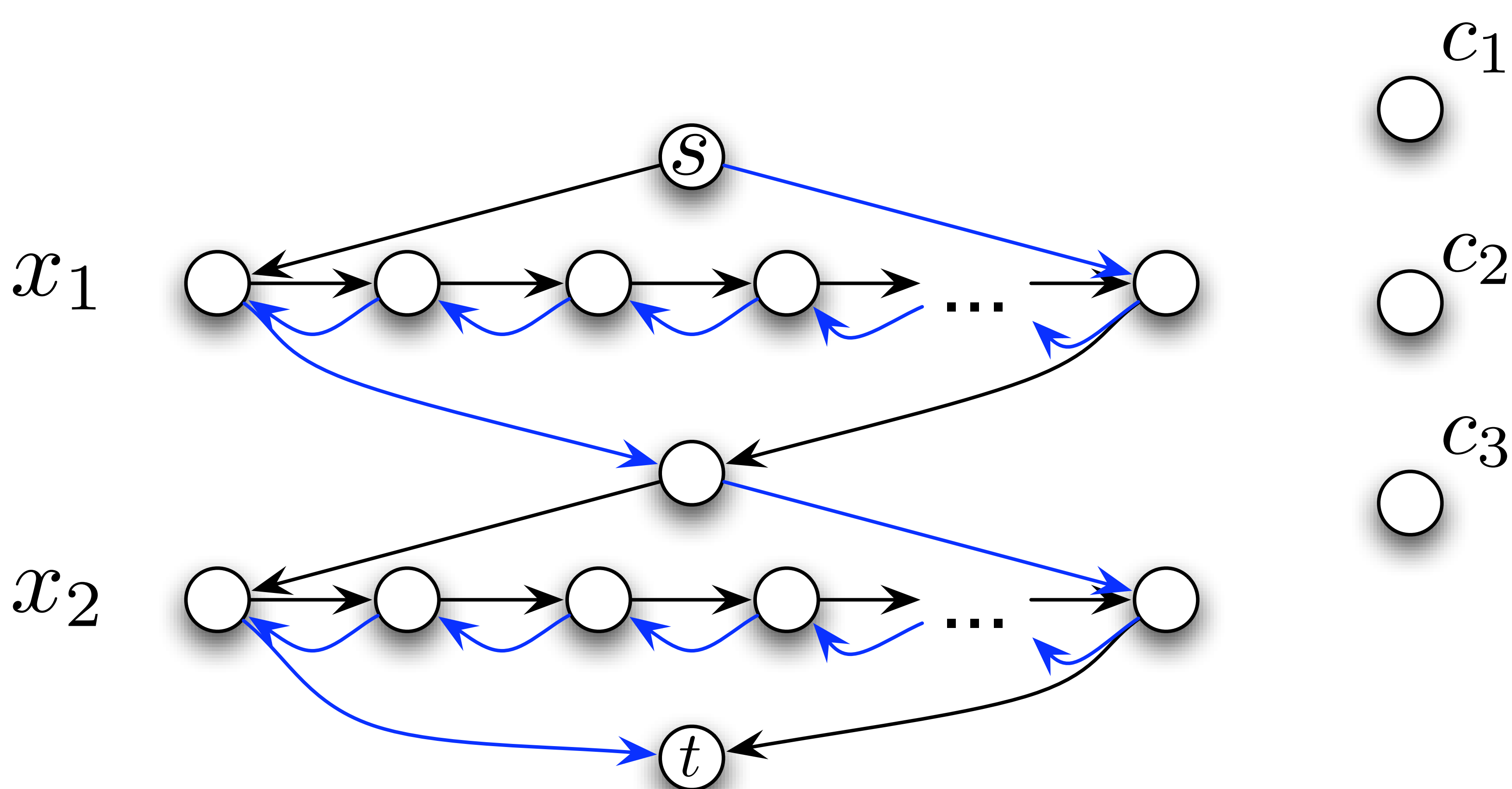
- Rautenförmige Knotenkonstrukte für die einzelnen Variablen xi verketten
- Startknoten s, Endknoten t



Hamiltonsche Pfade

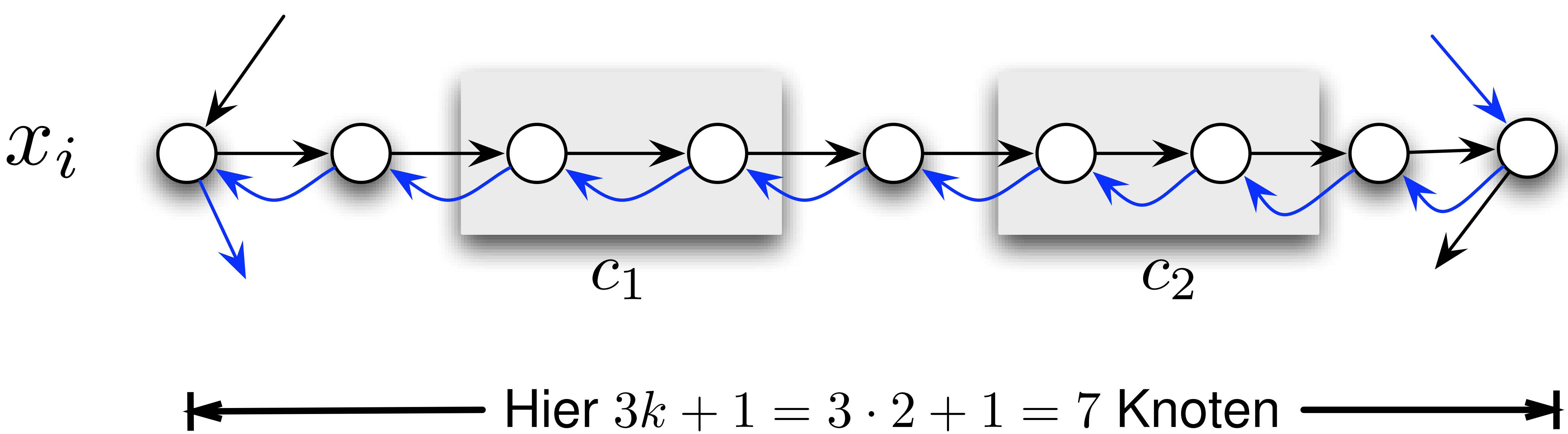
- Klauseln c_i aus ψ abbilden auf separate Knoten

$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



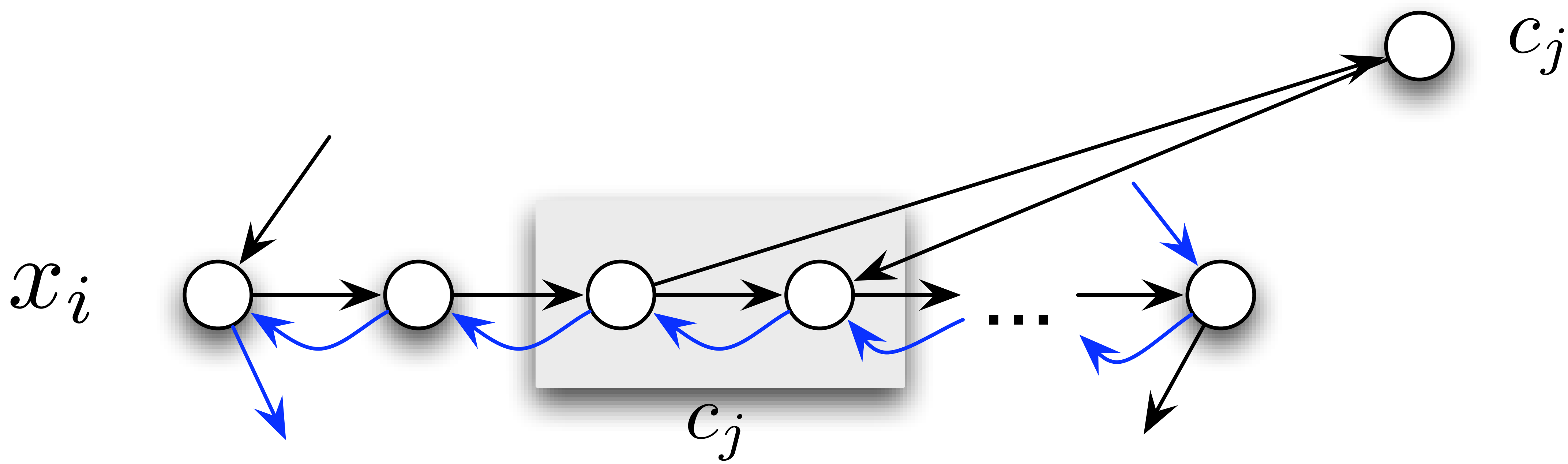
Hamiltonsche Pfade

- Horizontale Knotenzeile einer Knotenstruktur im Detail:
  - Für jede Klausel c_1 ... c_k ein Knotenpaar
  - „Trenner-Knoten“ zwischen den einzelnen Knotenpaaren, sowie am Anfang und Ende der Knotenzeile
  - d.h. 3k + 1 Knoten im Inneren der Rauten-Knotenstruktur



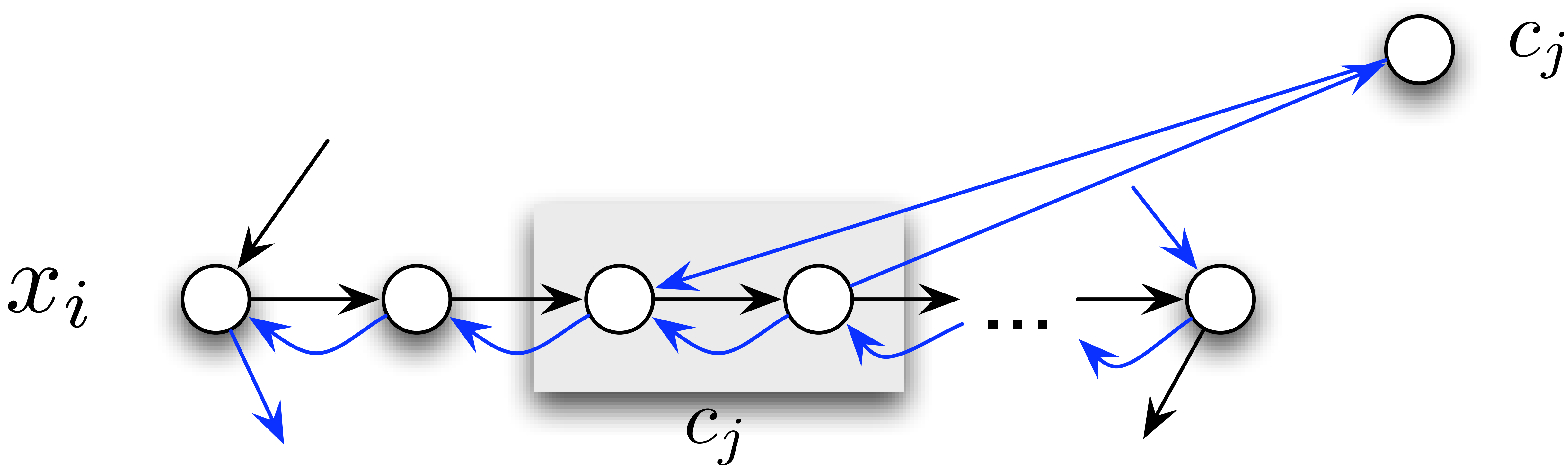
Hamiltonsche Pfade

- Zusätzliche Kanten zu den Klausel-Knoten
  - Falls Klausel c_j die Variable x_i enthält:



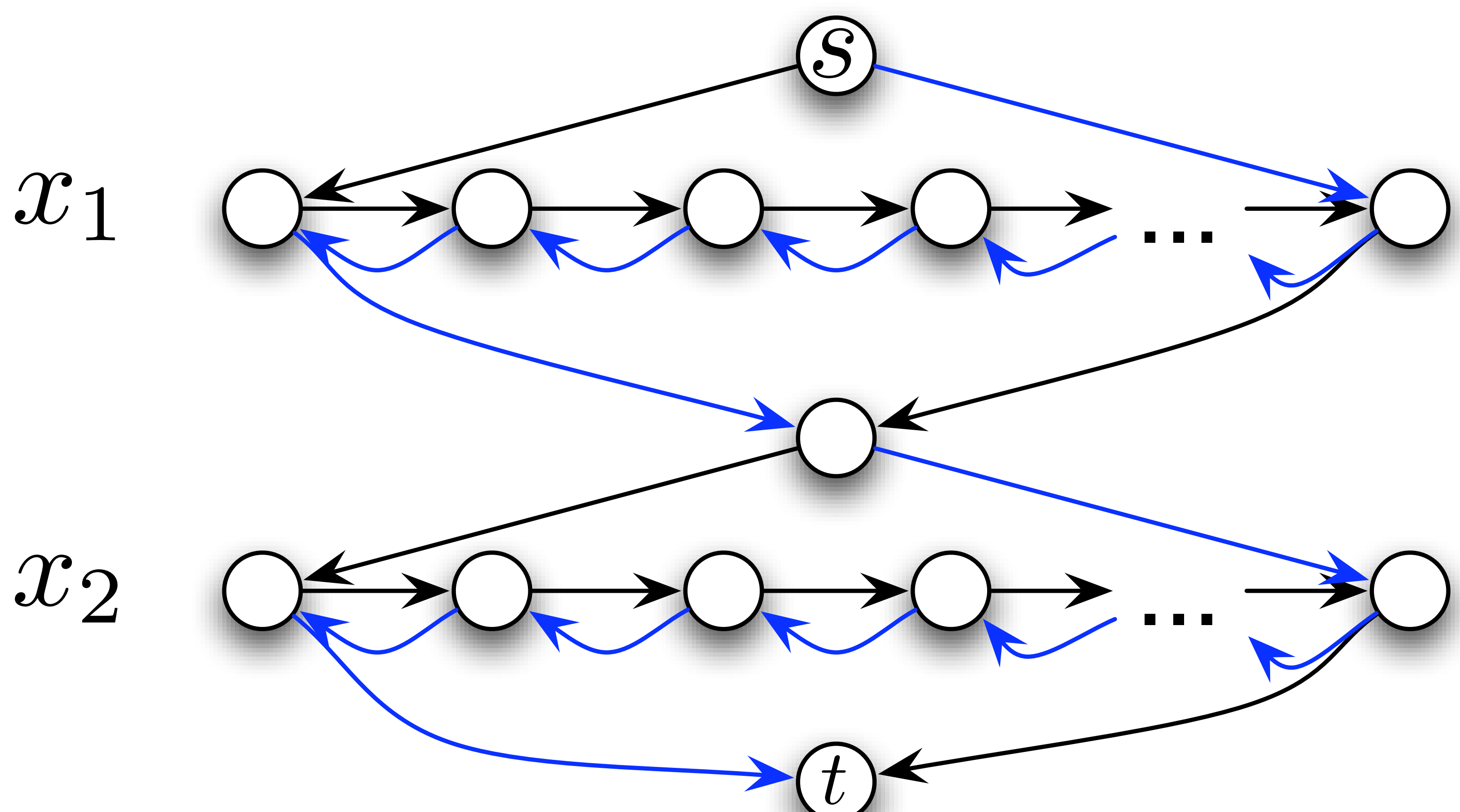
Hamiltonsche Pfade

- Zusätzliche Kanten zu den Klausel-Knoten
  - Falls Klausel c_j die Variable $\neg x_i$ enthält:



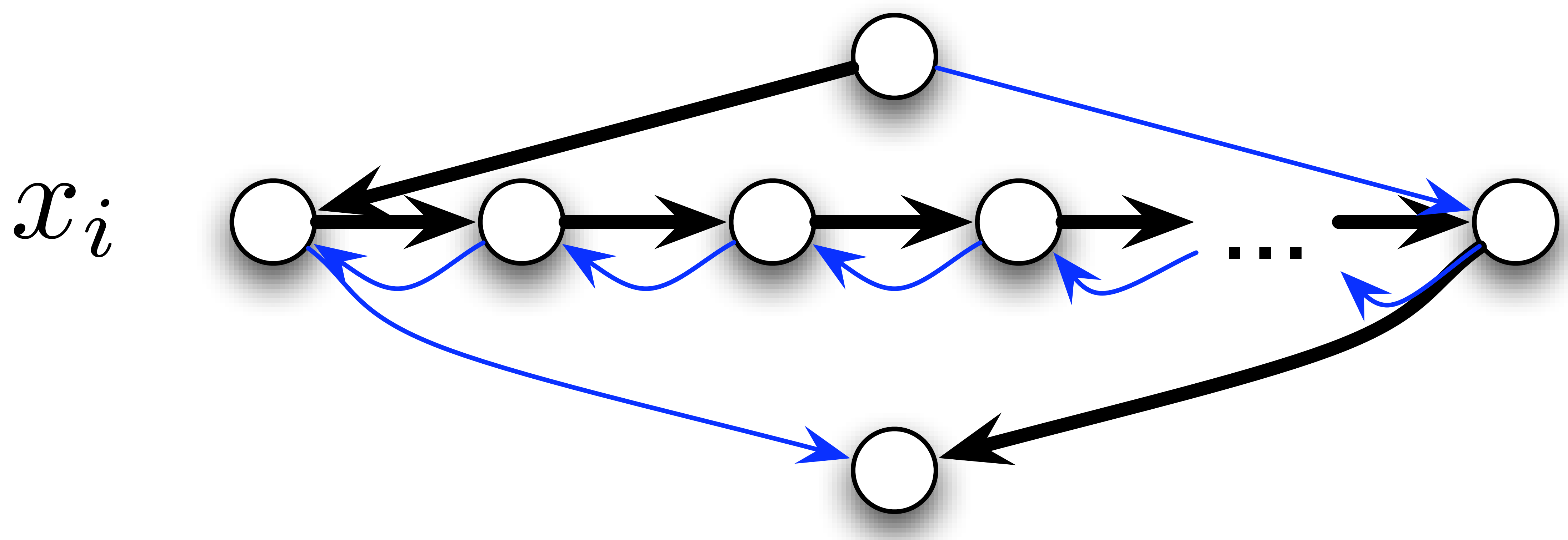
Hamiltonsche Pfade

- **ψ ist erfüllbar → G besitzt einen Hamiltonschen Pfad**
  - Vernachlässige zunächst die separaten Klausel-Knoten
  - Dann gibt es einen Pfad von s nach t, der der Reihe nach alle rautenförmigen Knotenstrukturen durchläuft



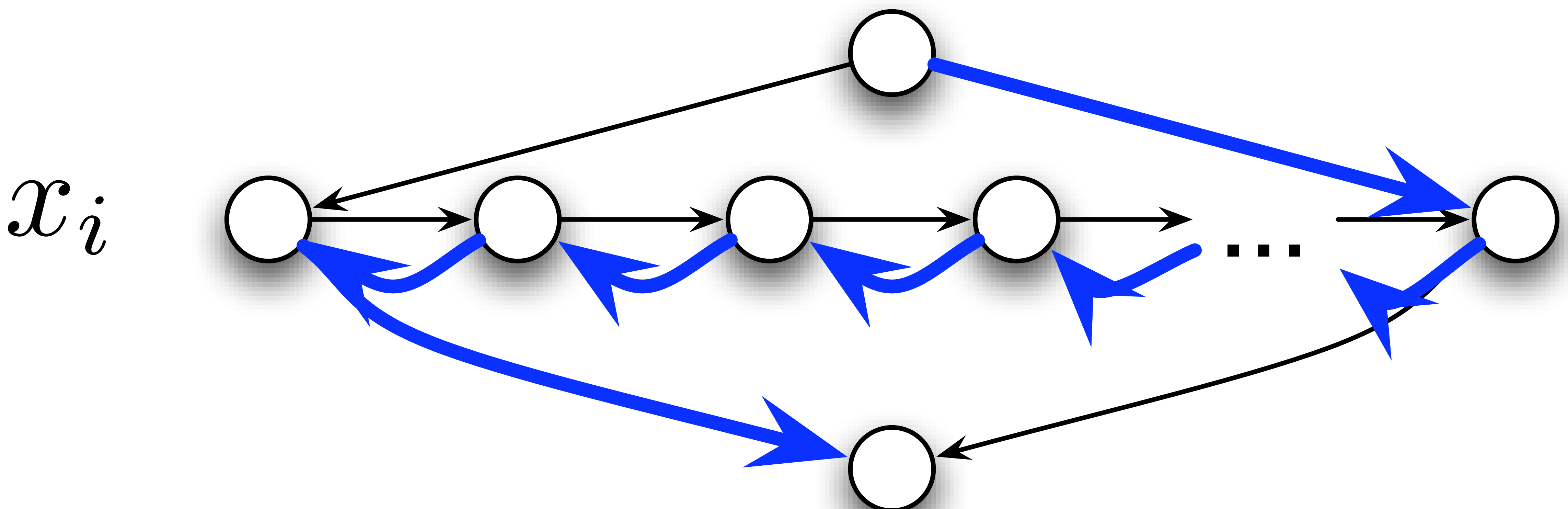
Hamiltonsche Pfade

- Durchlaufe dabei eine Rautenstruktur „zick-zack“, wenn die Variable x_i in der erfüllenden Belegung von ψ mit „wahr“ belegt ist, d.h.



Hamiltonsche Pfade

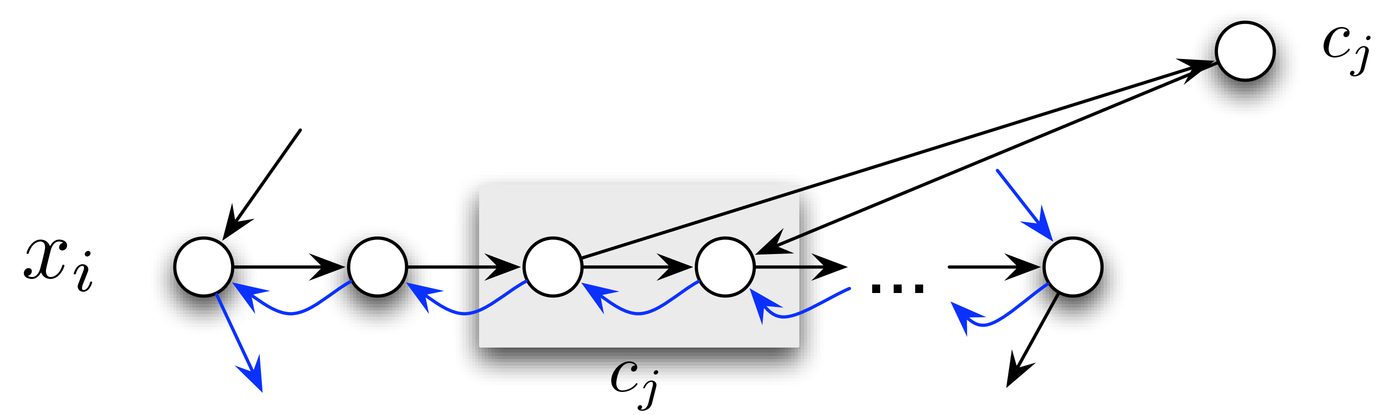
- **Durchlaufe dabei eine Rautenstruktur „zack-zick“, wenn die Variable x_i in der erfüllenden Belegung von ψ mit „falsch“ belegt ist, d.h.**



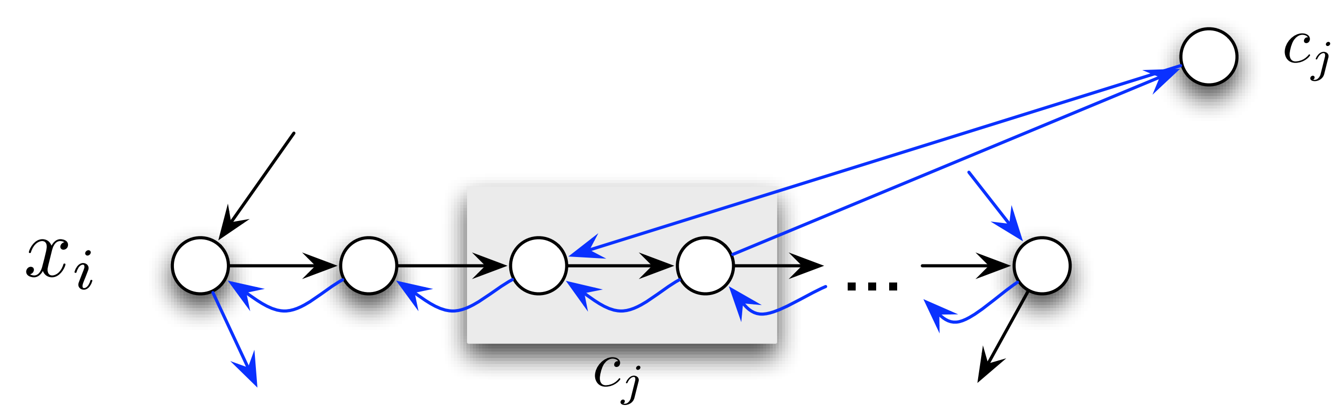
Hamiltonsche Pfade

- **Binde separate Klausel-Knoten in Pfad ein**
  - ψ ist erfüllbar
    → jede Klausel von ψ ist erfüllbar
    → jede Klausel enthält mindestens ein wahres Literal
  - Wähle in jeder Klausel genau ein wahres Literal und baue Umweg über Klausel-Knoten in Pfad ein
  - Durch Wahl des Wegs „zick-zack“ bzw. „zack-zick“ abhängig von der Variablenbelegung, zeigen die Kanten zu den Klauselknoten für wahre Literale stets in die richtige Richtung
    → Der beschriebene Pfad von *s* nach *t* existiert und ist ein Hamiltonscher Pfad

1. Fall c_j enthält x_i



2. Fall c_j enthält $\neg x_i$



Hamiltonsche Pfade

- **G besitzt einen Hamiltonschen Pfad → ψ ist erfüllbar**
  - Falls Rautenstrukturen der Reihe nach von oben nach unten durchlaufen werden:
    - Bestimme Variablenbelegung anhand „zick-zack“ bzw. „zack-zick“ Wegen
    - Hamiltonscher Pfad besucht alle Knoten, insbesondere auch alle separaten Klauselknoten

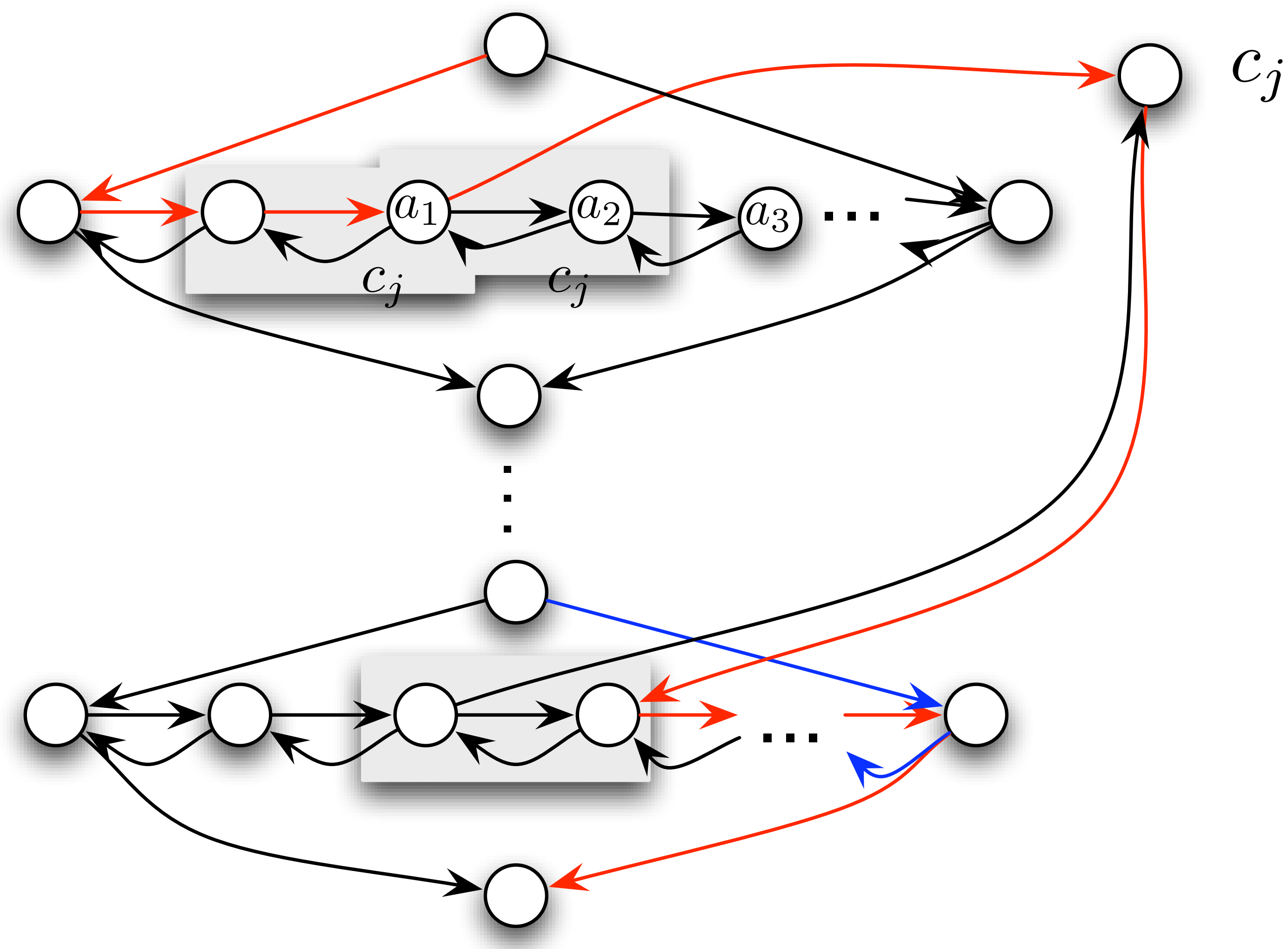
      → nach Konstruktion ist pro Klausel wenigstens ein Literal wahr

      → jede Klausel ist erfüllbar

      → ψ ist erfüllbar
  - Z.z.: Rautenstrukturen werden der Reihe nach von oben nach unten durchlaufen
    - Umrahmende Kanten der Rautenstrukturen gerichtet → Sprünge nur über Klauselknoten möglich

Hamiltonsche Pfade

- Falls der Pfad einen Sprung macht, muss entweder Knoten a_2 oder a_3 ein Trenner-Knoten sein
- a_2 Trenner-Knoten → a_2 hat eingehende Kanten von a_1 und a_3
- a_3 Trenner-Knoten → a_2 hat eingehende Kanten von a_1, a_3 und c_j
- a_1 und c_j schon im Pfad enthalten & Kante nach a_3 einzig möglicher weiterführender Pfad → kein Weg zurück nach a_2
- Widerspruch! → Pfad kann keinen Sprung gemacht haben

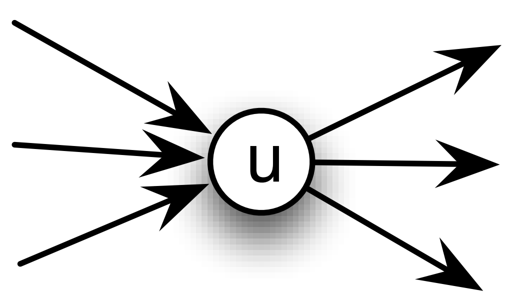


Ungerichtete Hamiltonsche Pfade

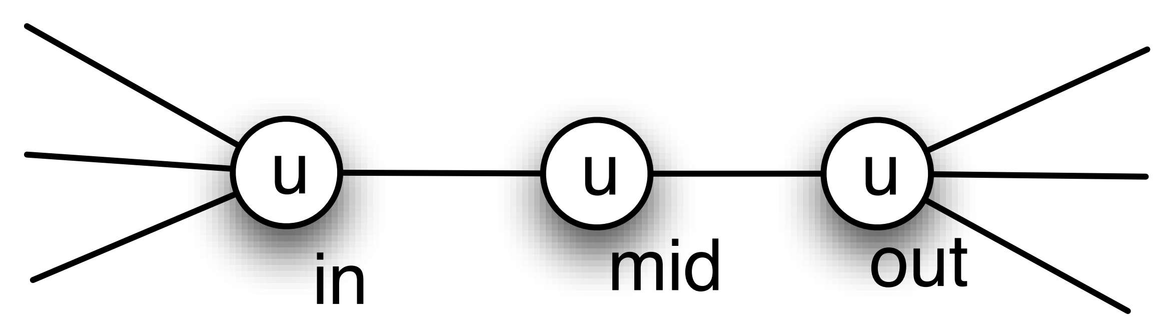
- **Definition:**
  - UHAMPATH = { (G, s, t) | Der ungerichtete Graph G enthält einen Weg von s nach t, der jeden Knoten genau einmal besucht. }
- **Theorem:**
  - UHAMPATH ist NP-vollständig
- **Beweis:**
  - UHAMPATH $\in$ NP: Verifizierer analog zu HAMPATH
  - Z.z.: UHAMPATH ist NP-schwierig

Ungerichtete Hamiltonsche Pfade

- **UHAMPATH ist NP-schwierig**
- **Beweis durch HAMPATH ≤m,p UHAMPATH**
  - Idee:



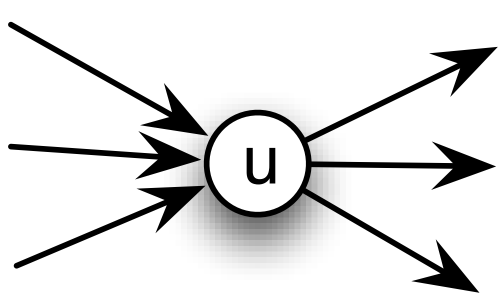
- **Reduktionsfunktion: f(G, *s*, *t*) = (G´, s^{out}, t^{in})**
  - G = (V, E), G´ = (V´, E´)
  - Für alle $u \in V \setminus \{s, t\}$
    - füge u^{in}, u^{mid}, u^{out} zu V´ und $\{u^{in}, u^{mid}\}$, $\{u^{mid}, u^{out}\}$ zu E´ hinzu
  - Für alle $(u, v) \in E \setminus \{(*, s), (t, *)\}$
    - füge $\{u^{out}, v^{in}\}$ zu E´ hinzu
  - Füge $s^{out} = s$, $t^{in} = t$ zu V´ hinzu

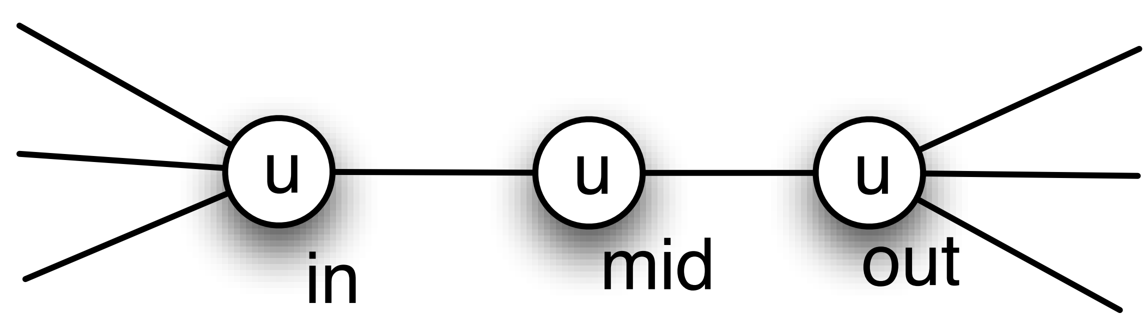


Ungerichtete Hamiltonsche Pfade

- **(G, s, t) $\in$ HAMPATH $\rightarrow$ (G´, s^{out}, t^{in}) $\in$ UHAMPATH**
  - Dem Pfad s, u_1 , u_2, …, u_k, t in G entspricht
  - nach Konstruktion offensichtlich der Pfad
  - s^{out}, u_1^{in}, u_1^{mid}, u_1^{out}, u_2^{in}, u_2^{mid}, u_2^{out}, …, t^{in} in G´

- **(G´, s^{out}, t^{in}) $\in$ UHAMPATH $\rightarrow$ (G, s, t) $\in$ HAMPATH**
  - auf s^{out} muss ein u_i^{in} folgen
  - auf alle u_i^{in} müssen u_i^{mid} und u_i^{out} folgen
  - auf alle u_i^{out} muss ein u_j^{in} folgen (Spezialfall: $u_i^{out} \rightarrow t^{in}$)
  - Da es keine Kanten $\{t^{in}, u_i^{in}\} \in E´$ gibt, muss der Pfad in t^{in} enden. Weiterhin enthält der Pfad alle Knoten.
    $\rightarrow$ Es gibt einen entsprechenden Hamiltonschen Pfad in G





Komplexitätstheorie

Das Teilsummen-problem ist NP-vollständig

34

Das Teilsummenproblem

- **Definition SUBSET-SUM:**
  - Gegeben:
    - Menge von natürlichen Zahlen S = $\{x_1, \ldots, x_k\}$
    - Eine natürliche Zahl t
  - Gesucht:
    - Gibt es eine Teilmenge $\{y_1, \ldots, y_m\} \subseteq \{x_1, \ldots, x_k\}$ so dass $$\sum_{i=1}^{m} y_i = t$$
- **Theorem:**
  - SUBSET-SUM ist NP-vollständig

Teilsummenproblem ist in NP

- **Beweis:**
  - Teilmenge $\{y_1, ..., y_m\}$ dient als Zertifikat c (Größe offensichtlich polynomiell in Eingabelänge)
  - Verifizierer A(S, t, c)
    - Prüfe, ob c Kodierung einer Teilmenge $\{y_1, ..., y_m\}$ von S ist
    - Addiere die Elemente von $\{y_1, ..., y_m\}$ auf. Falls die Summe t ist, akzeptiere. Andernfalls verwerfe.
  - Laufzeit von A polynomiell in der Eingabelänge

3-SAT $\leq_{m,p}$ SUBSET-SUM

- **Sei ψ eine Boolesche Formel**
  - mit Variablen $x_1, \ldots, x_k$ und Klauseln $c_1, \ldots, c_m$.
- **Konstruiere die Menge S wie folgt:**
  - Für jede Variable x_i füge zwei Zahlen y_i, z_i hinzu
  - Für jede Klausel c_j füge zwei Zahlen g_j, h_j hinzu
  - Initialisiere y_i, z_i mit 10^{i+m-1} (Ziffern im Dezimalsystem)
  - Für jedes Literal x_i in Klausel c_j addiere 10^{j-1} zu y_i
  - Für jedes Literal $\neg x_i$ in Klausel c_j addiere 10^{j-1} zu z_i
  - Initialisiere g_j, h_j mit 10^{j-1}
  - Wähle t als $(k+m)$-stellige Dezimalzahl, bestehend aus k 1en gefolgt von m 3en
  - Reduktion in polynomieller Zeit durchführbar

Das Teilsummenproblem

- **Beispiel für** $\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

S	2	1	c_3	c_2	c_1
y_1					
z_1					
y_2					
z_2					

1. Schritt:

Für jede Variable x_i füge zwei Zahlen y_i, z_i hinzu

Das Teilsummenproblem

- **Beispiel für** $\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

S	2	1	c_3	c_2	c_1
y_1					
z_1					
y_2					
z_2					
g_1					
h_1					
g_2					
h_2					
g_3					
h_3					

2. Schritt:

Für jede Klausel c_j füge zwei Zahlen g_j, h_j hinzu

Das Teilsummenproblem

- **Beispiel für** $\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

S	2	1	c_3	c_2	c_1
y_1	0	1	0	0	0
z_1	0	1	0	0	0
y_2	1	0	0	0	0
z_2	1	0	0	0	0
g_1					
h_1					
g_2					
h_2					
g_3					
h_3					

3. Schritt:

Initialisiere y_i, z_i mit 10^{i+m-1}

Das Teilsummenproblem

- **Beispiel für** $\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

S	2	1	c_3	c_2	c_1
y_1	0	1	0	0	1
z_1	0	1	0	0	0
y_2	1	0	0	0	0
z_2	1	0	0	0	0
g_1					
h_1					
g_2					
h_2					
g_3					
h_3					

4. Schritt:

Für jedes Literal x_i in Klausel c_j addiere 10^{j-1} zu y_i

Das Teilsummenproblem

- **Beispiel für** $\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

S	2	1	c_3	c_2	c_1
y_1	0	1	0	0	1
z_1	0	1	0	0	0
y_2	1	0	0	0	1
z_2	1	0	0	0	0
g_1					
h_1					
g_2					
h_2					
g_3					
h_3					

4. Schritt:

Für jedes Literal x_i in Klausel c_j addiere 10^{j-1} zu y_i

Das Teilsummenproblem

- **Beispiel für** $\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

S	2	1	c_3	c_2	c_1
y_1	0	1	0	0	1
z_1	0	1	0	0	0
y_2	1	0	1	0	1
z_2	1	0	0	0	0
g_1					
h_1					
g_2					
h_2					
g_3					
h_3					

4. Schritt:

Für jedes Literal x_i in Klausel c_j addiere 10^{j-1} zu y_i

Das Teilsummenproblem

- **Beispiel für** $\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

S	2	1	c_3	c_2	c_1
y_1	0	1	0	0	1
z_1	0	1	0	1	0
y_2	1	0	1	0	1
z_2	1	0	0	0	0
g_1					
h_1					
g_2					
h_2					
g_3					
h_3					

5. Schritt:

Für jedes Literal $\neg x_i$ in Klausel c_j addiere 10^{j-1} zu z_i

Das Teilsummenproblem

- **Beispiel für** $\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

S	2	1	c_3	c_2	c_1
y_1	0	1	0	0	1
z_1	0	1	1	1	0
y_2	1	0	1	0	1
z_2	1	0	0	0	0
g_1					
h_1					
g_2					
h_2					
g_3					
h_3					

5. Schritt:

Für jedes Literal $\neg x_i$ in Klausel c_j addiere 10^{j-1} zu z_i

Das Teilsummenproblem

- **Beispiel für** $\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

S	2	1	c_3	c_2	c_1
y_1	0	1	0	0	1
z_1	0	1	1	1	0
y_2	1	0	1	0	1
z_2	1	0	0	1	0
g_1					
h_1					
g_2					
h_2					
g_3					
h_3					

5. Schritt:

Für jedes Literal $\neg x_i$ in Klausel c_j addiere 10^{j-1} zu z_i

Das Teilsummenproblem

- **Beispiel für** $\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

S	2	1	c_3	c_2	c_1
y_1	0	1	0	0	1
z_1	0	1	1	1	0
y_2	1	0	1	0	1
z_2	1	0	0	1	0
g_1			0	0	1
h_1			0	0	1
g_2			0	1	0
h_2			0	1	0
g_3			1	0	0
h_3			1	0	0

6. Schritt:

Initialisiere g_j, h_j mit 10^{j-1}

Das Teilsummenproblem

- **Beispiel für** $\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

S	2	1	c_3	c_2	c_1
y_1	0	1	0	0	1
z_1	0	1	1	1	0
y_2	1	0	1	0	1
z_2	1	0	0	1	0
g_1			0	0	1
h_1			0	0	1
g_2			0	1	0
h_2			0	1	0
g_3			1	0	0
h_3			1	0	0
t	1	1	3	3	3

7. Schritt:

Wähle *t* als (*k*+*m*)-stellige Dezimalzahl, bestehend aus *k* 1en gefolgt von *m* 3en

Das Teilsummenproblem

- **$\psi \in$ 3-SAT → (S, t) $\in$ SUBSET-SUM**
  - Falls x_i wahr in erfüllender Belegung, füge y_i zur Teilsumme hinzu, andernfalls z_i
  - Die linken k Stellen von t sind 1en
  - ψ erfüllbar
    - → jede Klausel erfüllbar
    - → in der Summe wenigstens eine 1 pro Klausel-Spalte
    - → fülle Teilsumme mit g_i, h_i auf, so dass jede Klausel-Spalte in der Summe 3 hat
  - Damit (S, t) $\in$ SUBSET-SUM

S	2	1	c_3	c_2	c_1
y_1	0	1	0	0	1
z_1	0	1	1	1	0
y_2	1	0	1	0	1
z_2	1	0	0	1	0
g_1			0	0	1
h_1			0	0	1
g_2			0	1	0
h_2			0	1	0
g_3			1	0	0
h_3			1	0	0
t	1	1	3	3	3

Das Teilsummenproblem

- **Beispiel für** $\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

S	2	1	c_3	c_2	c_1
y_1	0	1	0	0	1
z_1	0	1	1	1	0
y_2	1	0	1	0	1
z_2	1	0	0	1	0
g_1			0	0	1
h_1			0	0	1
g_2			0	1	0
h_2			0	1	0
g_3			1	0	0
h_3			1	0	0
t	1	1	3	3	3

ψ ist erfüllbar (x_1 = falsch, x_2 = wahr)

es gibt eine Teilmenge {z_1, y_2, g_1, h_1, g_2, h_2, g_3} von S, deren Summe t ist

Das Teilsummenproblem

- **(S, *t*) ∈ SUBSET-SUM → ψ ∈ 3-SAT**
  - Beobachtungen:
    - alle Ziffern der Zahlen aus S sind entweder 0 oder 1
    - in jeder Klausel-Spalte können nach Konstruktion niemals mehr als fünf 1en stehen

      → es kann keinen Übertrag geben
    - Damit die linken k Stellen von t 1en sind, muss in der Teilsumme für jedes i entweder y_i oder z_i enthalten sein
  - Falls die Teilsumme y_i enthält, setze x_i = „wahr“,
    - andernfalls (z_i in Teilsumme) setze x_i = „falsch“
  - Teilsumme hat eine 3 in allen Klausel-Spalten
  - Potentielle Summanden g_i, h_i können max. 2 beitragen

    → y_i, z_i in Teilsumme haben min. eine 1 pro Klausel-Spalte

    → jede Klausel erfüllbar

    → ψ ∈ erfüllbar

Komplexitätstheorie

Approximation

52

Approximation

- **Ziele dieser Vorlesung:**
  - Verständnis der Begriffe
    - Approximations-Güte
    - Approximations-Algorithmus
    - Approximations-Schema
  - Verständnis der Beispiel-Algorithmen für die Probleme
    - Vertex Cover (Knotenüberdeckung)
    - Traveling Salesman Problem (TSP)

Motivation

- **Viele wichtige Probleme sind NP-vollständig**
  - (also nicht effizient lösbar unter der Annahme $P \neq NP$)
- **Diese sind zu wichtig um sie zu ignorieren**
- **Mögliche Lösungen:**
  - Für kleine n ist exponentielle Laufzeit akzeptabel
  - Spezialfälle vielleicht in polynomieller Zeit lösbar
  - Vielleicht tritt worst-case Laufzeit extrem selten auf
  - Möglicherweise kann eine beweisbar gute Näherungs-Lösung in polynomieller Zeit berechnet werden

Konzepte und Terminologie (1)

- **Wir betrachten Optimierungsprobleme**
- **Problem X hat viele Lösungen**
- **Wir suchen Lösung S für X, die eine Kostenfunktion c(S) minimiert oder maximiert.**
  - Beispiele für Graphen:
    - finde einen minimalen Spannbaum eines Graphen
    - finde einen minimalen Hamiltonkreis
- **Seien C, C* Kosten der approximierten bzw. optimalen Lösung.**
- **Ein Approximations Algorithmus hat Approximations-Güte ρ(n),**
  - falls für jede Eingabegröße n

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$$

Konzepte und Terminologie (2)

- **Approximations Algorithmus mit Güte ρ(n) wird als ρ(n)-Approximations Algorithmus bezeichnet**
- **Approximations-Schema**
  - Approximations Algorithmus mit zusätzlichem Parameter ε>0 der (1+ε)-Approximation liefert
  - Falls Laufzeit polynomiell in n für jedes feste ε, spricht man von einem **polynomiellen Approximations-Schema** (polynomial time approximation scheme, PTAS)
  - Falls Laufzeit polynomiell in n und ε (z.B. $(1/\varepsilon)^2 n^2$), spricht man von einem **streng polynomiellen Approximations-Schema**

Komplexitätstheorie

Approximation von Vertex-Cover

57

Vertex Cover Problem

- **Szenario:**
  - Alte Netzwerk-Router (Knoten) sollen gegen neue ausgetauscht werden, die Netzwerkverbindungen (Kanten) überwachen können.
  - Zum Überwachen einer Verbindung genügt es, wenn ein Router adjazent ist. Wieviele neue Router werden mindestens benötigt?
- **Formal:**
  - Gegeben $G = (V, E)$.
  - Finde minimale Teilmenge V' so dass
    - für alle $(u,v) \in E$ gilt:

      $$u \in V' \text{ oder } v \in V'.$$

Approximation für Vertex Cover

ApproxVertexCover(G(V,E))

1. C $\leftarrow \emptyset$

2. E´ $\leftarrow$ E

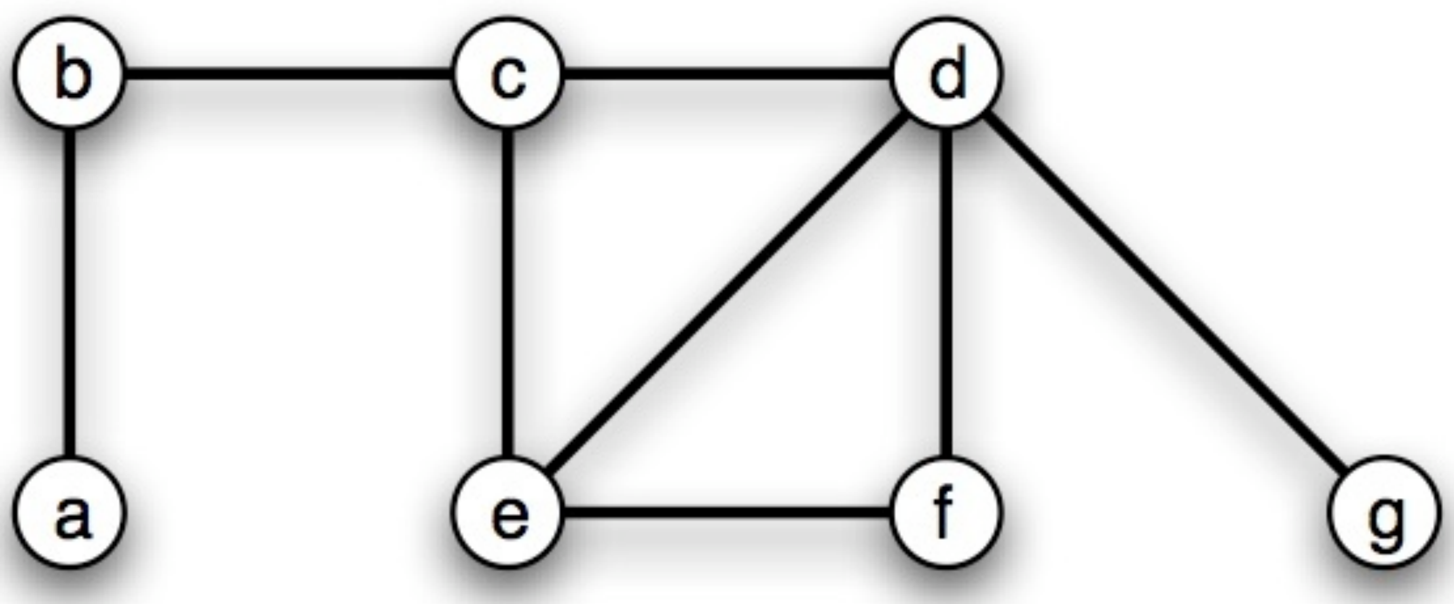
3. solange E´$\neq \emptyset$

4. wähle {u,v} aus E‘ beliebig

5. C $\leftarrow$ C $\cup$ {u,v}

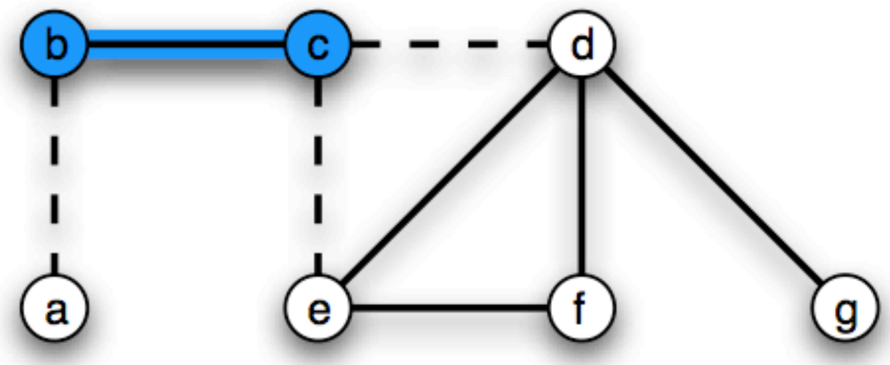
– entferne zu u oder v inzidente Kanten aus E‘

6. gebe C aus

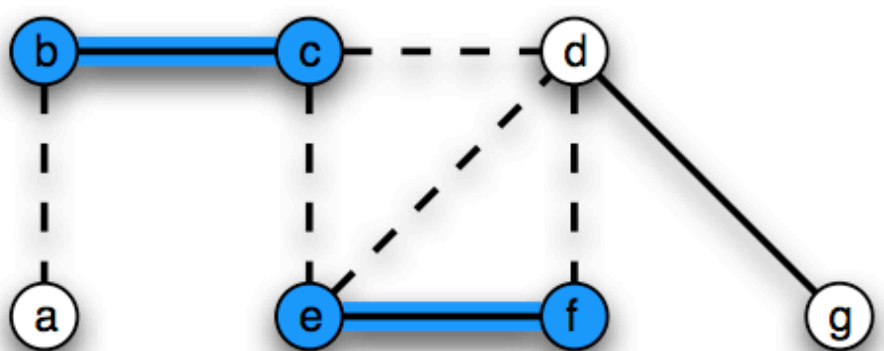


Beispiel

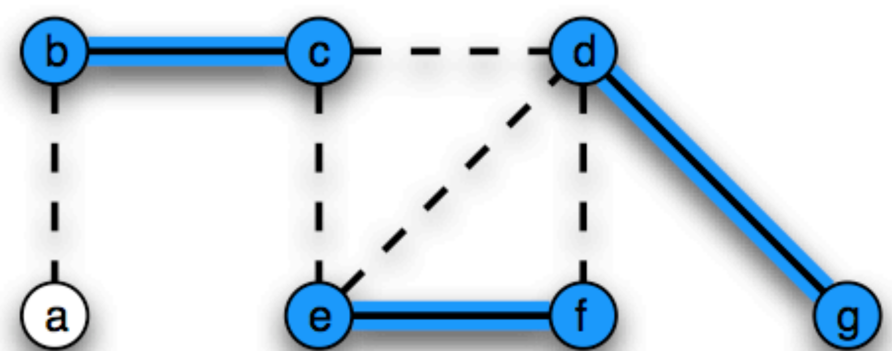
APPROXVERTEXCOVER($G(V,E)$)
$C \leftarrow \emptyset$
$E' \leftarrow E$
so lange $E' \neq \emptyset$
    wähle $\{u,v\} \in E'$ zufällig
    $C \leftarrow C \cup \{u,v\}$
    entferne zu u,v inzidente Kanten aus E'
gebe C aus



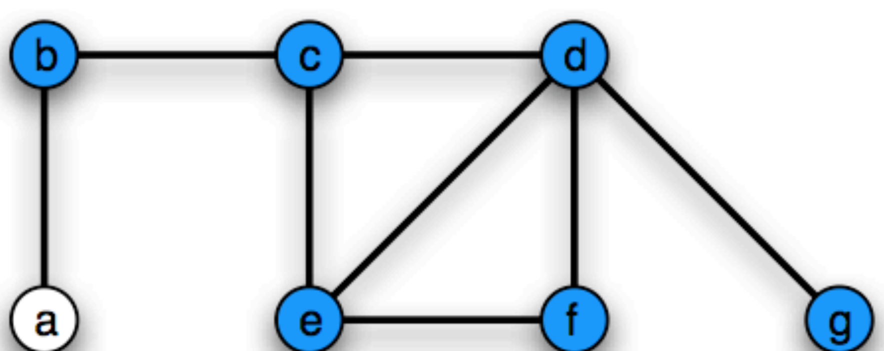
(1)



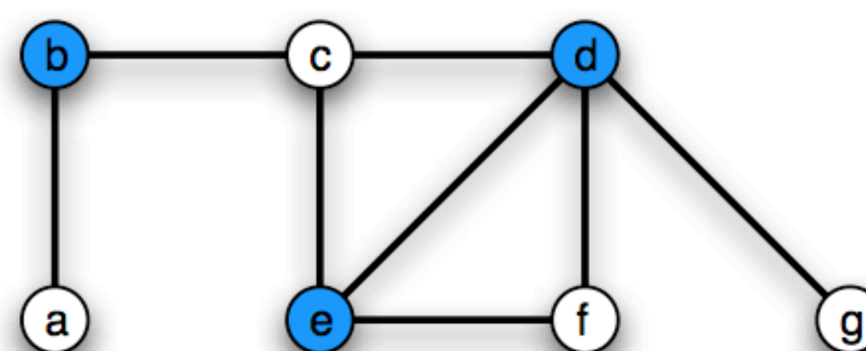
(2)



(3)



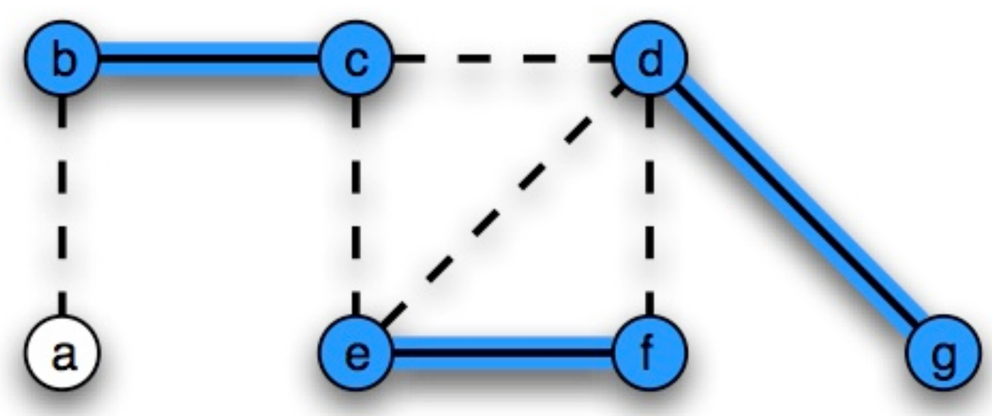
(4)



(minimale Lösung)

Analyse: ApproxVertexCover (1)

- **Theorem:**
  - ApproxVertexCover hat Güte 2 und Laufzeit O(E)
- **Beweis:**
  - Korrektheit,
    d.h. Lösung C ist Vertex Cover
    - Algorithmus läuft bis jede Kante in E zu Knoten in C inzident ist
      - (sobald Kante gewählt, werden alle inzidenten Kanten entfernt)
  - Güte 2
    - Sei A Menge der in Zeile 4 gewählten Kanten (blau)
    - Keine 2 Kanten in A teilen einen Endpunkt
    - Jede Iteration fügt 2 neue Knoten zu C hinzu, $|C| = 2\,|A|$
    - Minimales Vertex Cover C^* muss wenigstens einen Knoten jeder Kante in A enthalten
    - Da keine Kanten in A Endpunkte teilen: $|A| \leq |C^*|$
    - somit gilt $|C| \leq 2|C^*|$



Analyse: ApproxVertexCover (2)

ApproxVertexCover(G(V,E))

1. C $\leftarrow \emptyset$

2. E´ $\leftarrow$ E

3. solange E´$\neq \emptyset$

4. wähle {u,v} aus E‘ beliebig

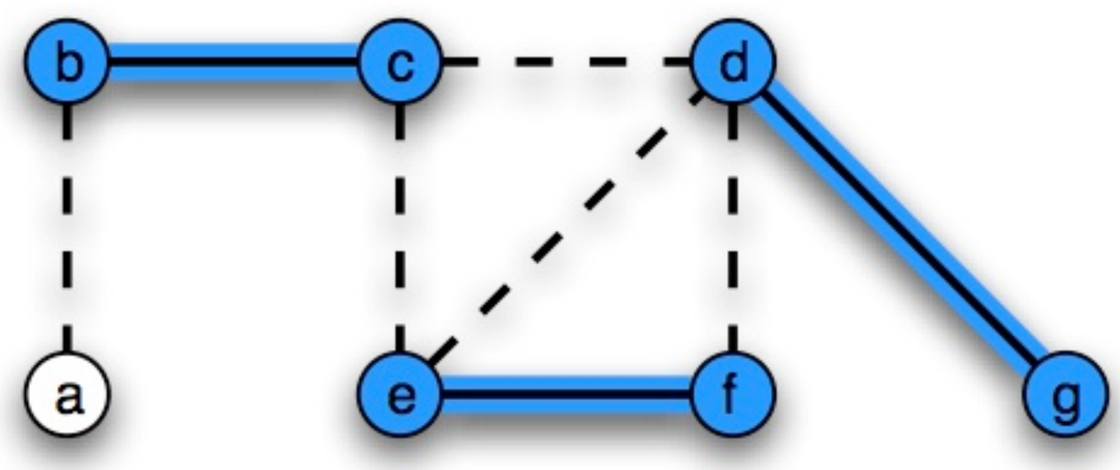
5. C $\leftarrow$ C $\cup$ {u,v}

6. entferne zu u oder v inzidente Kanten aus E‘

7. gebe C aus

- **Theorem:**
  - ApproxVertexCover hat Güte 2 und Laufzeit O(E)

- **Beweis:**
  - Laufzeit O(E)
    - In jeder Iteration wird eine Kante aus E’ entfernt
    - bei geeigneter Datenstruktur für E’: Laufzeit O(E)



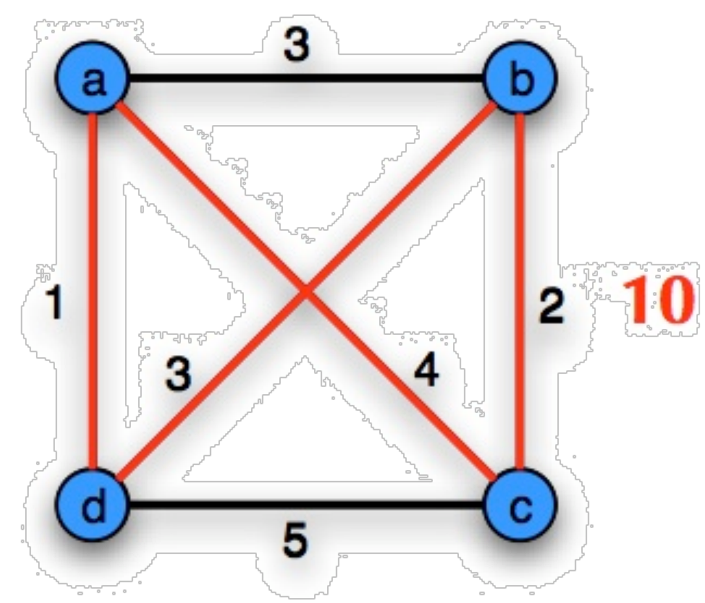
Komplexitätstheorie

Approximation von Traveling Salesman

63

Traveling Salesman Problem (TSP)

- **Gegeben:**
  - vollständiger Graph G=(V,E)
  - Kostenfunktion c(u,v) für alle (u,v) ∈ E
- **Gesucht:**
  - Hamiltonkreis (Tour) mit minimalen Kosten
- **Theorem:**
  - Falls P≠NP existiert kein polynomieller Approximations-Algorithmus für TSP mit konstanter Güte
- **Beweis:**
  - Annahme es existiert ein Algorithmus, der TSP in pol. Zeit löst
  - Man kann zeigen: Hamiltonkreis $\leq_{m,p}$ TSP
  - Wir wissen: Hamiltonkreis ist NP-vollständig
  - Also kann A nicht existieren, falls P≠NP

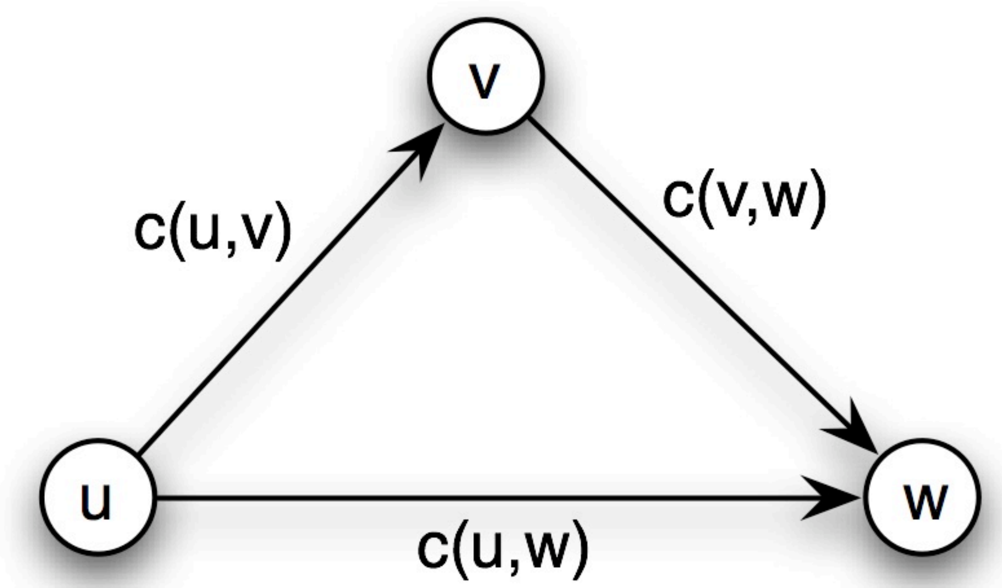


Traveling Salesman Problem

- **TSP mit Einschränkung: Δ-TSP**
- **Gegeben:**
  - vollständiger Graph G=(V,E)
  - Kostenfunktion c(u,v) für alle (u,v) ∈ E

$$\forall u, v, w \in V \ : \ c(u,w) \leq c(u,v) + c(v,w)$$

- **Gesucht:**
  - Hamiltonkreis (Tour) mit minimalen Kosten
- **Beschränkung der Gewichte durch Dreiecksungleichung ist „natürlich“:**
  - ist in vielen Anwendungsfällen automatisch erfüllt
  - z.B. wenn Gewichte Entfernungen im Euklidischen Raum repräsentieren
- **Aber: Auch Δ-TSP ist NP-schwierig!**
  - wird hier nicht bewiesen



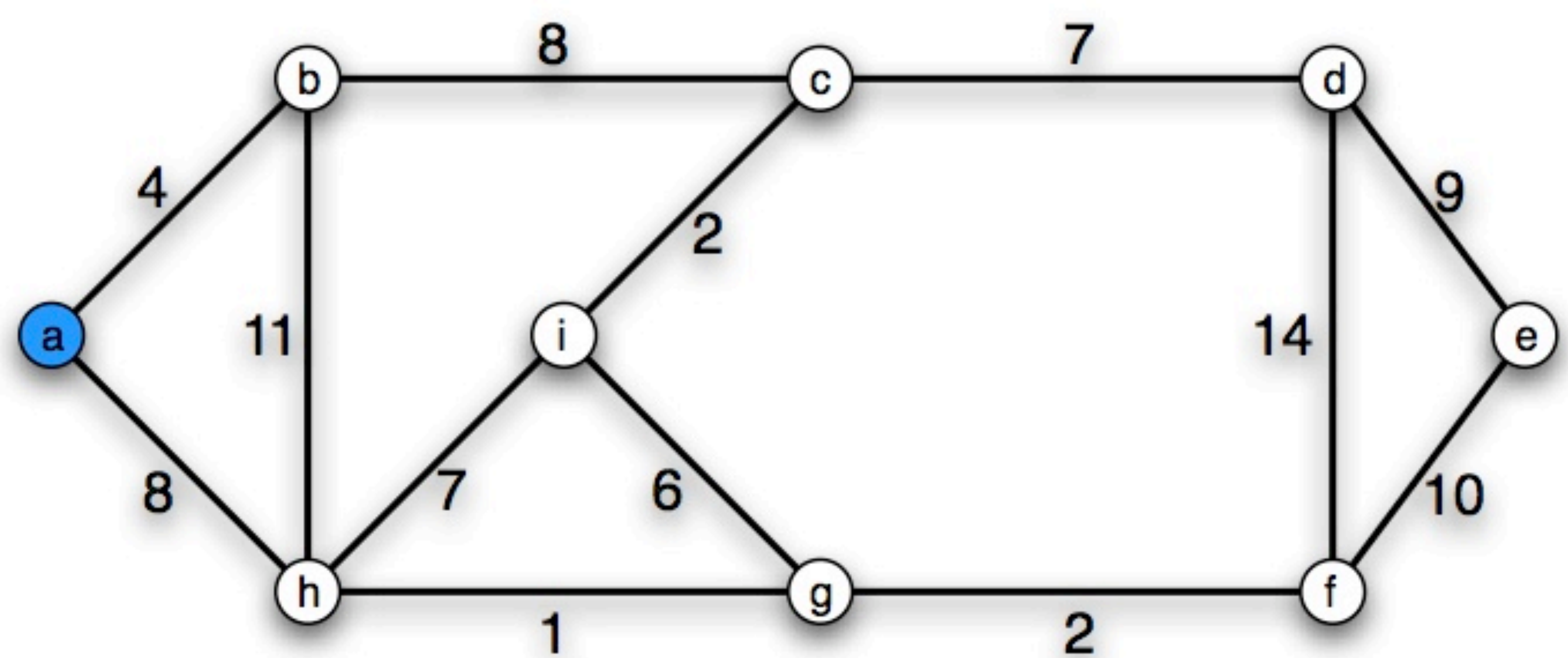
Approximation für Δ-TSP

- **Lösungsansatz:**
  - Gibt es ein ähnliches/verwandtes Problem?
  - Ist es einfacher zu berechnen?
- **Minimale Spannbäume**
  - MST: Minimal Spanning Tree
  - Aber: wie kann ein MST in eine kürzeste Tour umgeformt werden?
- **Idee:**
  - bei Tiefensuche im MST wird jede Kante zweimal traversiert
  - Besuche alle Knoten in der Reihenfolge eines Pre-Order Tree-Walks

Prims Algorithmus zur Berechnung des MST

MST-PRIM(G)

1. **Initialisiere Baum B mit beliebigen Knoten**
2. **Wiederhole bis B alle Knoten enthält oder nicht erweitert werden kann**
   - Erweitere B mit Kante mit geringstem Gewicht, die B mit dem Rest von G verbindet

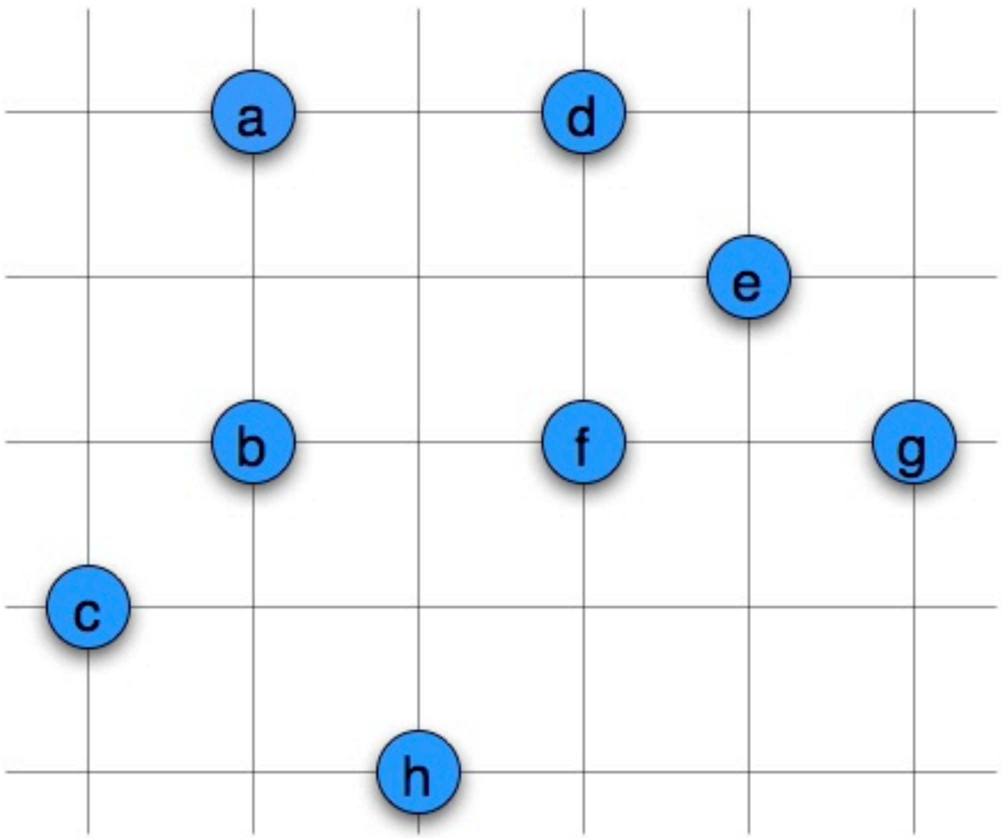


67

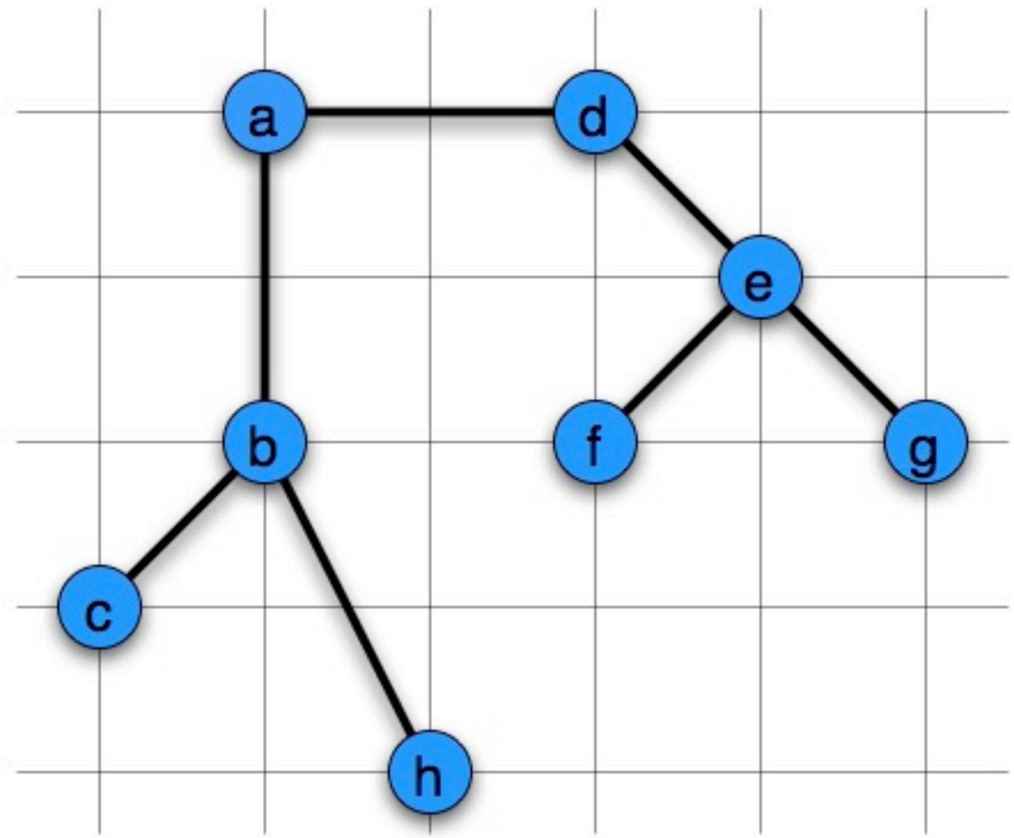
Approximation für Δ-TSP

$\textsc{Approx-}\Delta\textsc{-TSP}(G, c)$
1 wähle Knoten $v \in V$
2 berechne minimalen Spannbaum T für G mit Wurzel v
3 konstruiere Liste L der Knoten die durch pre-order tree-walk in T entsteht
4 Gebe den durch L definierten Hamiltonkreis aus

Graph G
(gew. gemäß euklidischer Distanz)



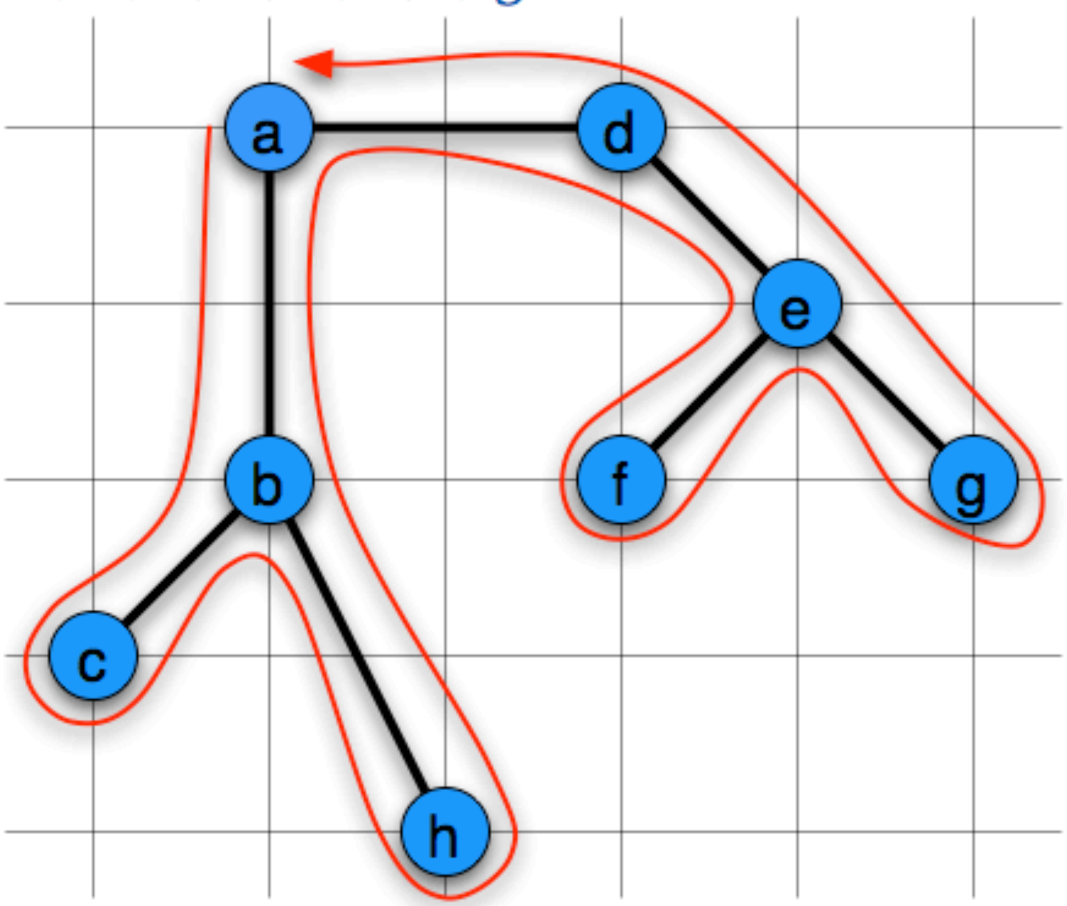
Spannbaum T mit Wurzel A



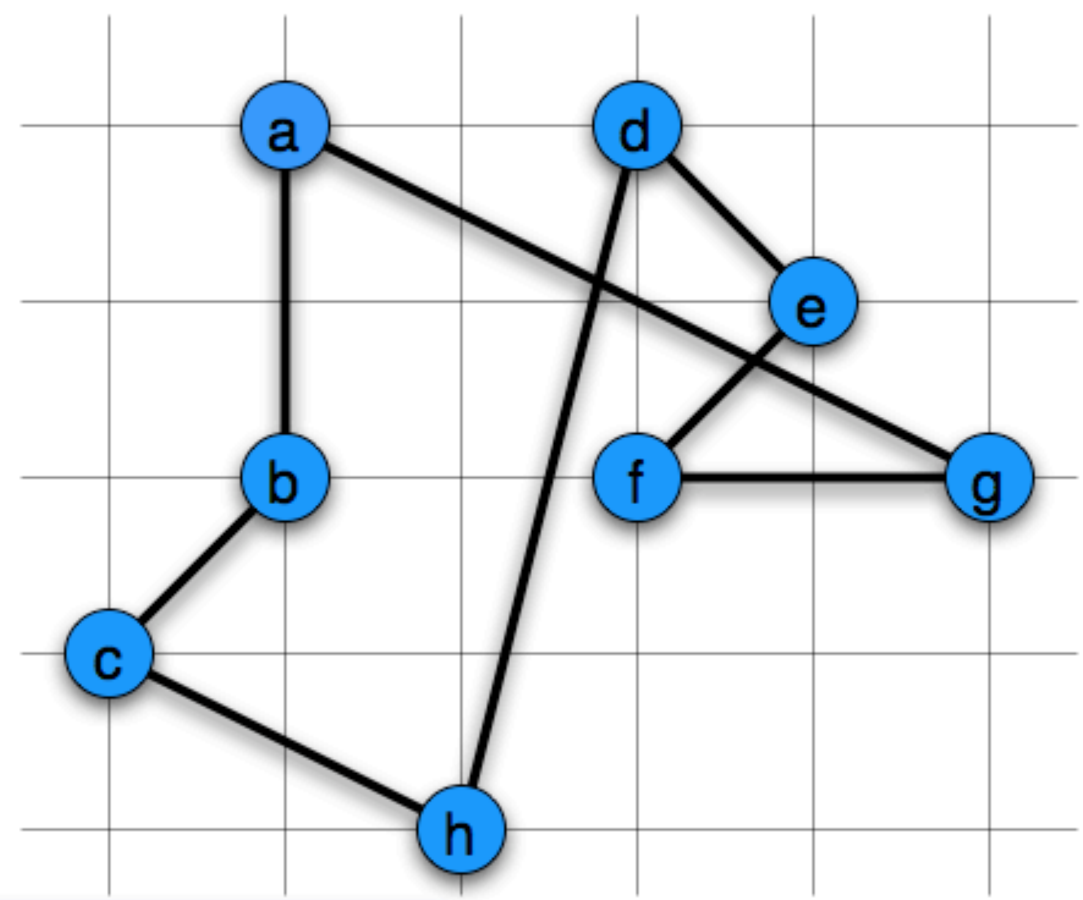
Approximation für Δ-TSP

APPROX-Δ-TSP(G, c)
1 wähle Knoten $v \in V$
2 berechne minimalen Spannbaum T für G mit Wurzel v
3 konstruiere Liste L der Knoten die durch pre-order tree-walk in T entsteht
4 Gebe den durch L definierten Hamiltonkreis aus

pre-order tree-walk:
(a, b, c, h, d, e, f, g)



durch *L* definierter Hamiltonkreis:



Approximation für Δ-TSP

$\text{Approx-}\Delta\text{-TSP}(G, c)$

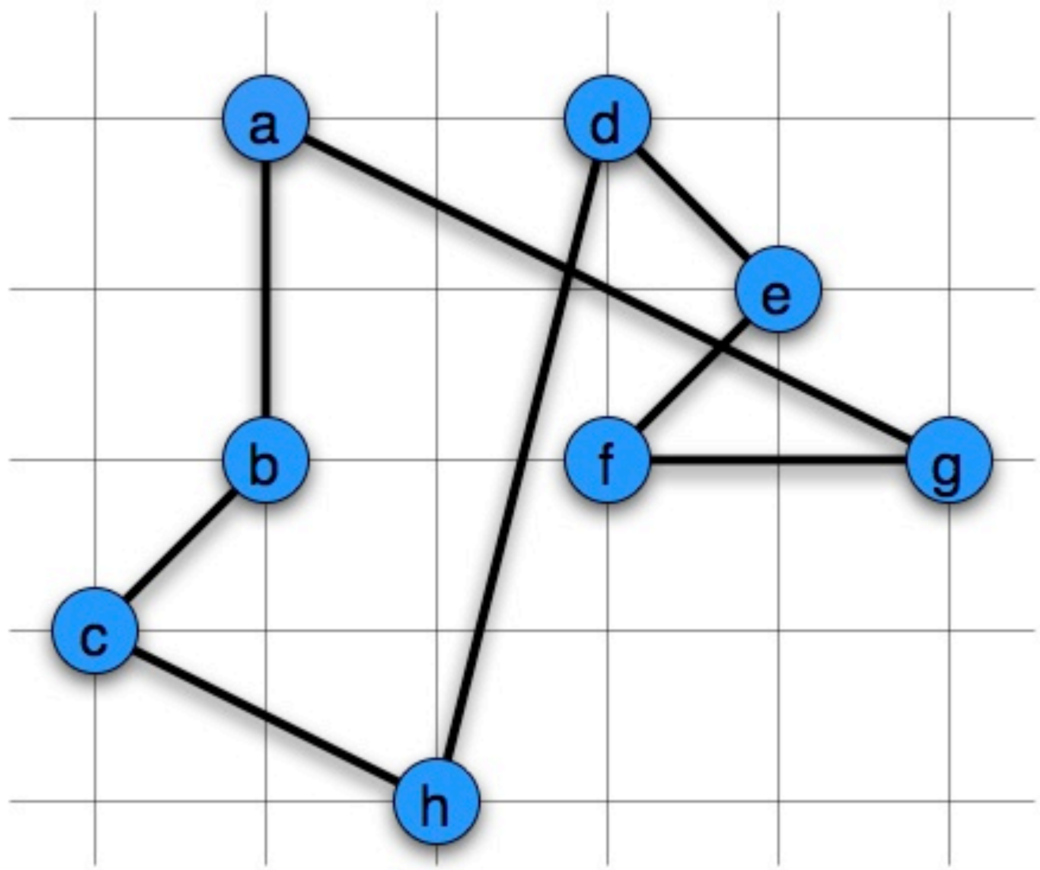
1 wähle Knoten $v \in V$

2 berechne minimalen Spannbaum T für G mit Wurzel v

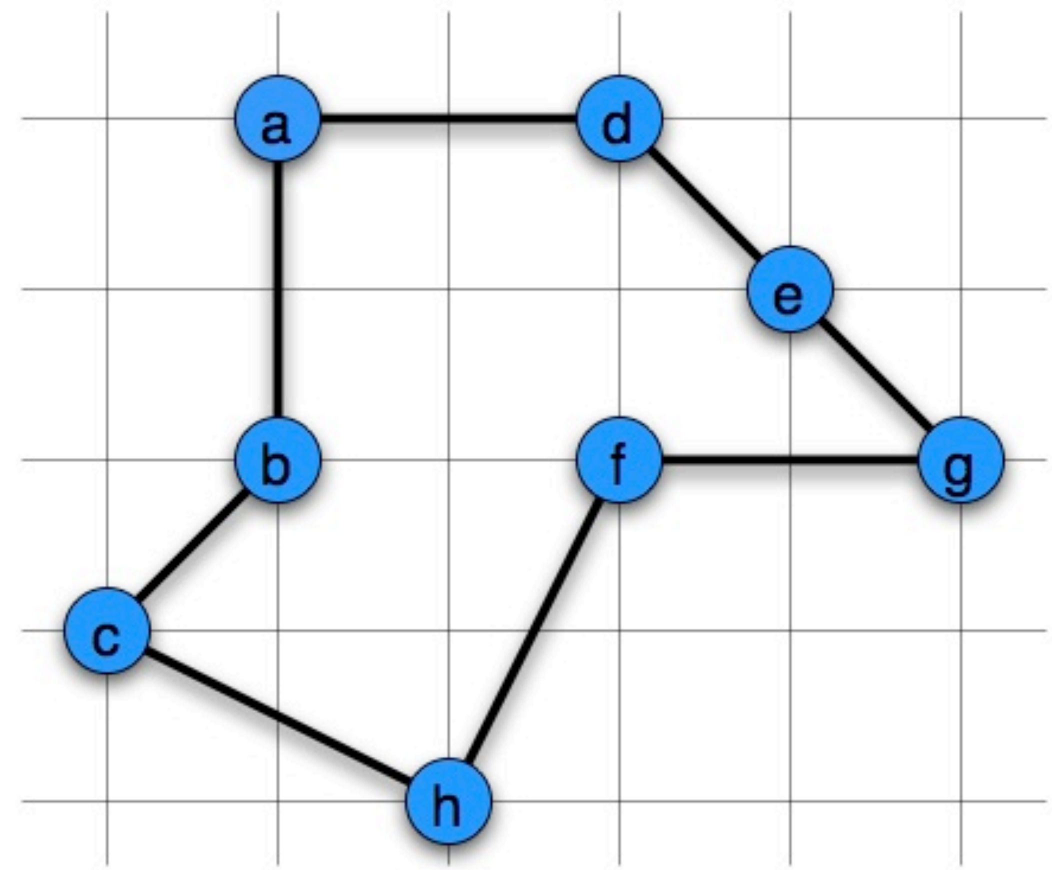
3 konstruiere Liste L der Knoten die durch pre-order tree-walk in T entsteht

4 Gebe den durch L definierten Hamiltonkreis aus

durch L definierter Hamiltonkreis:



minimaler Hamiltonkreis:



Approximation für Δ-TSP

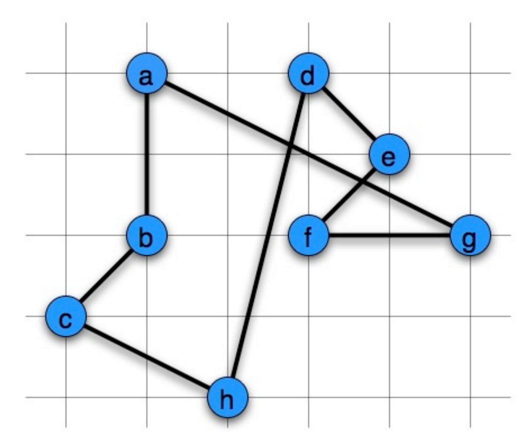
```
APPROX-Δ-TSP(G, c)
wähle Knoten v ∈ V
berechne minimalen Spannbaum T für G mit Wurzel v
konstruiere Liste L der Knoten die durch pre-order tree-walk in T entsteht
Gebe den durch L definierten Hamiltonkreis aus
```

- **Theorem:**
  - Approx-Δ-TSP hat Laufzeit $\Theta(|E|) = \Theta(|V|^2)$
  - Approx-Δ-TSP ist Approximations-Algorithmus für Δ-TSP mit Güte 2
- **Beweis:**
  - Sei H Ergebnis von Approx-Δ-TSP und H* optimale Lösung
  - Seien Kosten einer Tour A definiert durch:

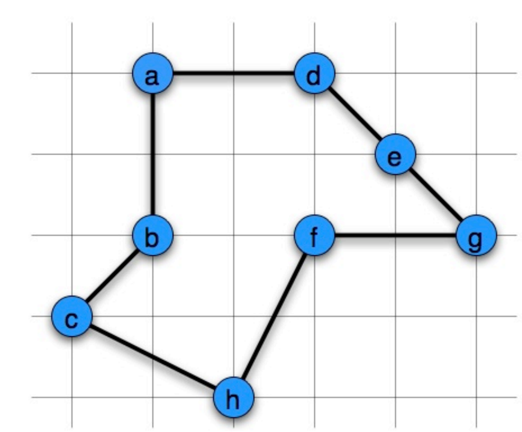
$$c(A) = \sum_{(u,v) \in A} c(u,v)$$

  - Theorem besagt: $c(H) \leq 2\, c(H^*)$

durch L definierter Hamiltonkreis:

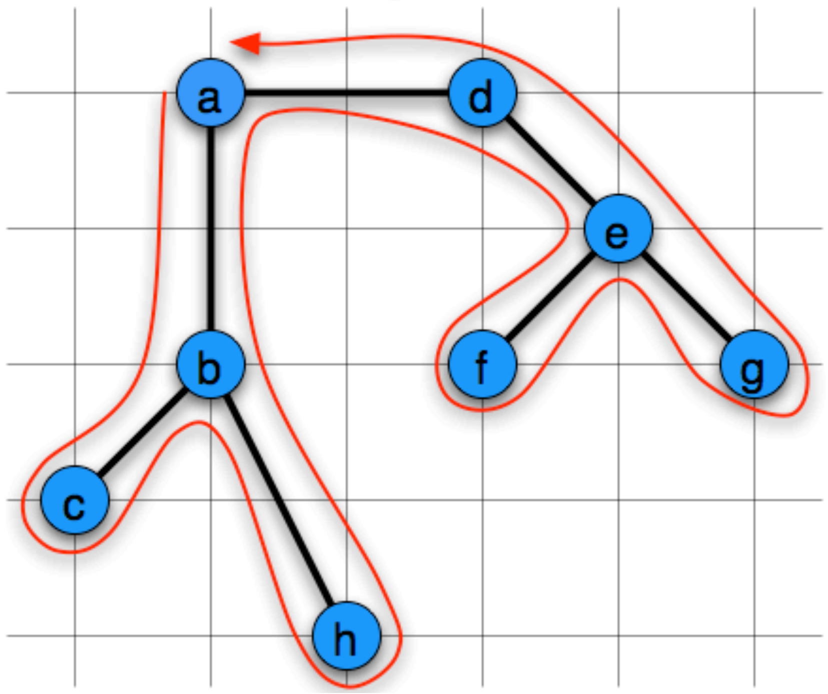


minimaler Hamiltonkreis:



Approximation für Δ-TSP

- **Theorem:**
  - Approx-Δ-TSP ist Approximations-Algorithmus für Δ-TSP mit Güte 2
- **Beweis (Fortsetzung):**
  - Sei T der erzeugte Spannbaum. Es gilt: $c(T) \leq c(H^*)$
    - Streichen einer Kante liefert in H* liefert Spannbaum mit Kosten $\leq c(H^*)$
  - Betrachte Folge F der Kanten die beim pre-order walk besucht werden:
    - z.B. F= (a,b,c,b,h,b,a,d,e,f,e,g,e,a)
    - F besucht jede Kante zweimal, somit gilt:
      - $c(F) = 2c(T)$
  - Daraus folgt $c(F) \leq 2c(H^*)$
  - H entsteht durch Streichen von Knoten in F. Wegen Dreiecksungleichung folgt
    - $c(H) \leq c(F)$
  - Nun folgt:
    - $c(H) \leq 2\, c(H^*)$



Anmerkungen zum TSP

- **Der angegebene Algorithmus kann leicht verbessert werden**
- **Algorithmus von Christofides liefert Güte 3/2**
- **Weiterhin hat Sanjeev Arora 1996 ein streng polynomielles Approximations-Schema vorgestellt:**
  - Eingabe:
    - eine Instanz des Δ-TSP im d-dimensionalen Euklidischen Raum
    - und $\varepsilon > 0$
  - Erzeugt in Zeit $n^{O(d/\varepsilon)}$ eine Rundreise mit Güte $1 + \varepsilon$

4.2 NP

Ende

74