Kap. 7: NP-Vollständigkeit

Formalisierung und Codierung von Problemen
Komplexitätsklassen P, NP und NPC
NP-vollständige Probleme, Reduktionsbeweis
Noch mehr NP-vollständige Probleme

NP-Vollständigkeit

- bisher:
  - Probleme analysiert und effiziente Algorithmen / Datenstrukturen entwickelt
- Vorgehen, wenn man keinen effizienten Algorithmus findet:
  1. Vermutung: Es gibt einen effizienten Algorithmus
     - Weitersuchen!
  2. Vermutung: Es gibt keinen effizienten Algorithmus
     - Nachweis durch eine untere Laufzeitschranke
       - extrem schwierig, gelingt nur sehr selten
     - Nachweis, dass das Problem zu einer Klasse von scheinbar schwer zu lösenden Problemen gehört
       - viel einfacher, gelingt häufig!
- Ziel: Formalismus, mit dem man die Schwierigkeit eines Problems verdeutlichen kann

Schwere und einfache Probleme

1. Finde den längsten, einfachen Weg in einem Graphen	schwer!
2. Finde den kürzesten, einfachen Weg in einem Graphen	O(|E|)
3. Hamilton-Kreis: einfacher Kreis, der alle Knoten enthält	schwer!
4. Euler-Tour: Kreis, der alle Kanten enthält	O(|E|)
5. Clique: vollständiger Teilgraph mit k Knoten	schwer!
6. Inpedependent Set: Teilgraph mit k Knoten ohne Kanten	schwer!
7. Färbbarkeit: adjazente Knoten haben unterschiedliche Farben.	
- Färbbar mit zwei Farben?	O(|E|)
- Färbbar mit drei Farben?	schwer!
8. PARTITION: geg. eine Menge ganzer Zahlen, Lässt sich die Menge in zwei Teilmengen gleicher Summe zerlegen?	schwer!

7.1 Formalisierung von ‚Problemen‘

- Problemtypen:
  - Optimierungsproblem: finde eine gültige Lösung mit einem minimalen Wert bzgl. einer Bewertungsfunktion
  - Entscheidungsproblem: prüfe, ob eine gültige Lösung existiert.
- Sei P ein Optimierungsproblem:
  - E_P = ‚Gibt es eine Lösung für P mit Wert $\leq$ k?‘ ist das zugehörige Entscheidungsproblem (***k*-Threshold-Problem**)
  - Eingabe ist die Eingabe von P und der Wert k
  - Sei A ein Algorithmus für ein k-Threshold-Problem E_P,

    P kann durch die wiederholte Anwendung von A mit binärer Suche gelöst werden
- Im folgenden betrachten wir nur Entscheidungsprobleme.
  - Komplexität von Optimierungsproblemen lassen sich über das zugehörige k-Threshold-Problem bewerten.

Codierung von Problemen

- Codierung:
  - Abbildung von Objekten in die Menge der Binärstrings $\{0,1\}^*$
    - Abstraktes Problem: Problem definiert über komplexe Objekte
    - Konkretes Problem: Funktion f: {0,1}* → {0,1}
  - Codierung bildet ein abstraktes Problem auf ein konkretes Problem ab
  - für nicht belegte Eingabecodes x gilt f(x) = 0
- Formale Sprachen:
  - konkretes Entscheidungsproblem f kann als Menge betrachtet werden:

    $L_P = \{ x \in \{0,1\}^* \mid f(x) = 1 \}$
- Länge der Codierung
  - Ein abstraktes Problem mit Eingabe n lässt sich als ein konkretes Problem mit Eingabelänge $l(n) = O(n^k)$, k konstant, codieren.

Codierung von Problemen

- Bsp (Länge der Codierung):

  Graph mit *n* Knoten und *m* Kanten

  - Verwende

    00: binär 0, 01: binär 1,

    10: neues Listenelement 11: neue Liste

  - Adj.-Liste zu Knoten i: 11bin(i)10bin(Nachbar 1)10bin(Nachbar 2)...

  - Länge l(m,n) der binärcodierten Eingabe
    - $2\log_2 n$ pro Knotennummer, bei n Knoten und m Kanten:

    $l(n,m) = n * 2 \log_2 n + 2*m*(2+ 2\log_2 n) = O(m \log n)$

7.2 Komplexitätsklassen P, NP und NPC

- f: {0,1}* → {0,1}* heißt *polynomzeit-berechenbar*

  g.d.w. ein Algorithmus A zur Berechnung von f existiert mit Laufzeit $T_A(n) = O(n^k)$ für eine Konstante k

- **Komplexitätsklasse P:**

  Menge aller konkreten Entscheidungsprobleme f:{0,1}*→ {0,1}, die polynomzeit-berechenbar sind.

  - umgangsprachlich:
    - abstrakte Entscheidungsprobleme f sind in P, g.d.w. es eine Codierung polynomieller Länge gibt und das zugehörige konkrete Entscheidungsproblem ist in P
    - Ein Optimierungsproblem f ist in P, g.d.w. zugehörige k-Threshold-Problem E_f in P ist.

- Polynome sind gegen Verkettung abgeschlossen, d.h. sind f und g Polynome, so ist h: x → f(g(x)) ebenfalls ein Polynom
- Solange die Codierung polynomiell ist, ändert sie nichts an der Tatsache, ob ein abstraktes Problem in P ist oder nicht.

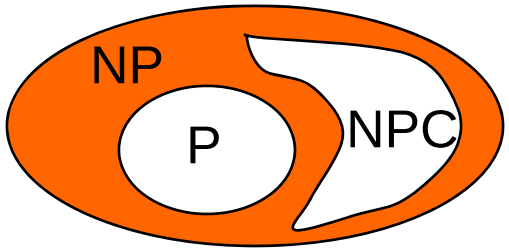
Die Klasse NP

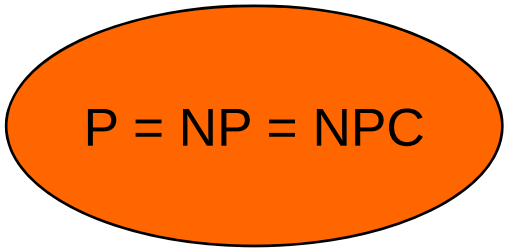
- f:{0,1}*→{0,1} heißt *polynomzeit-verifizierbar, g.d.w.*
  - zu jeder Eingabe x ∈ {0,1}* es existiert ein Zertifikat y ∈ {0,1}* mit |y| = $O(|x|^k)$
  - es existiert ein Algorithmus A mit A(x,y) = 1 g.d.w. f(x) = 1 und $T_A(n)=O(n^k)$ für ein konstantes k
- umgangssprachlich:
  - Zertifikat stellt die Lösung dar, der Algorithmus A überprüft in polynomieller Zeit, ob die Lösung korrekt ist.
  - Bsp. für Zertifikate:
    - k-Clique: Menge der Knoten, die eine k-Clique bilden
    - 3-Färbbarkeit: Funktion, die jedem Graphknoten eine Farbe zuweist
- **Komplexitätsklasse NP:**
  - Menge aller konkreten Entscheidungsprobleme f:{0,1}*→ {0,1}, die polynomzeit-verifizierbar sind
- Offensichtlich gilt: P ⊆ NP. Gilt P = NP? (vermutlich nicht!)

NP-Vollständigkeit

- Da P $\subseteq$ NP gilt, können wir schwierige Probleme über ihre Zugehörigkeit zu NP nicht identifizieren
- Was sind die schwierigsten Probleme in NP?

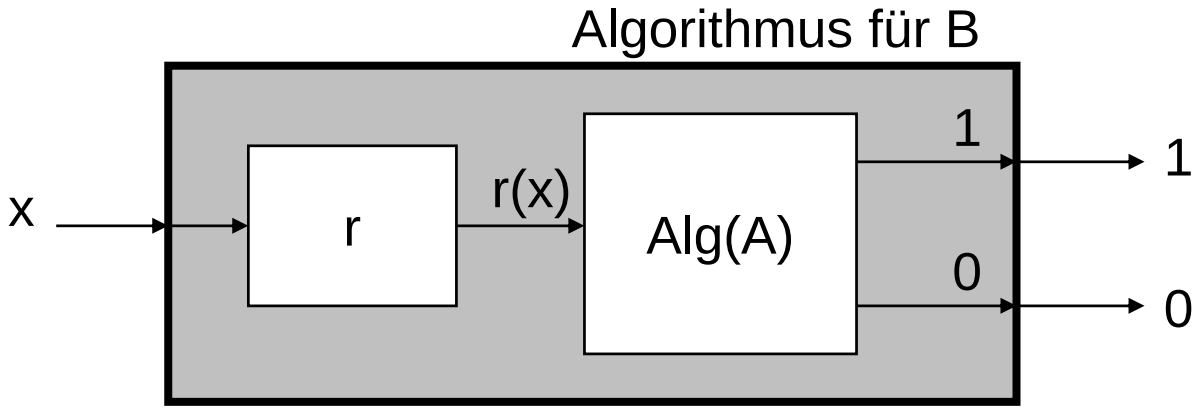
- NP-Vollständigkeit:
  - ein konkretes Entscheidungsproblem A $\in$ NP heißt NP-vollständig, g.d.w gilt: A $\in$ P => P = NP, bzw. P $\neq$ NP => A $\notin$ P
- **Komplexitätsklasse NPC**
  - NPC ist die Menge der NP-vollständigen Entscheidungsprobleme
  - Ein Optimierungsproblem f heißt NP-schwer, wenn das zugehörige k-Threshold-Problem E_f NP-vollständig ist.
- Zwei Szenarien sind denkbar:





Polynomzeit-Reduktion

- Seien A,B Entscheidungsprobleme:
- Wie können wir zeigen, dass A mindestens so komplex ist wie B?
  - Suche eine Reduktionsfunktion r:{0,1}* → {0,1}* mit den Eigenschaften:
    - $x \in B \Leftrightarrow r(x) \in A$
    - r kann in polynomieller Zeit berechnet werden
  - Gibt es ein entsprechendes r, so heißt B *polynomzeit-reduzierbar* auf A, oder $B \leq_P A$
  - Mit der Funktion r und einen Algorithmus Alg(A) für A kann B wie folgt berechnet werden:



  - Angenommen $A \in P \Rightarrow B \in P$

Nachweis von NP-Vollständigkeit

- **Satz:** Ein Problem A ist NP-vollständig ($A \in NPC$), g.d.w.
  1. $A \in NP$
  2. für alle $B \in NP$ gilt: $B \leq_P A$
- Angenommen $A \in P$, dann können wir Problem B mit folgenden Algorithmus Alg(B) lösen:
  - Berechne Reduktionsfunktion r(x)
  - Verwende Algorithmus für A

  => Alg(B) hat polynomielle Laufzeit, folglich gilt $B \in P$

  => Damit gilt für alle $B \in NP$: $B \in P$, also P = NP
- Wie weist man nun NP-Vollständigkeit eines Problems A nach?
  - zeige, dass $A \in NP$ gilt
  - wähle ein beliebiges Problem $Q \in NPC$
  - zeige, dass $Q \leq_P A$ gilt (Polynomzeit-Reduktion)

    $Q \in NPC$, also gilt für alle $B \in NP$: $B \leq_P Q$. mit $Q \leq_P A$ folgt für alle $B \in NP$: $B \leq_P A$

7.3 NP-vollständige Probleme

- Satisfiability-Problem (SAT)
  - Eingabe: logischer Ausdruck A in CNF (konjunktiver Normalform)
    - boole'schen Variablen x_i (mögliche Belegung: 0 oder 1)
    - NOT, AND und OR Operatoren
    - CNF: AND-Verknüpfung von *Klauseln*; eine Klausel ist eine OR-Verknüpfung von *Literalen* ((potentiell negierten) Variablen)
  - Ausgabe: 1 g.d.w. es eine Belegung gibt, so dass A wahr(1) ist.
- Beispiel:

  $(x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee \neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$

  Ausgabe: 1 Belegung: $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$

- **Cook's Theorem:** SAT $\in$ NPC

Das erste Problem in NPC

- Cook‘s Beweis (grobe Skizze)
  1. Zeige SAT $\in$ NP
     - Zertifikat: gültige Belegung für die Variablen
     - Verifikation durch den folgenden Algorithmus A:
       - Sei x eine Eingabe des SAT-Problems (logischer Ausdruck in CNF); y, das zugehörige Zertifikat (Belegung der Variablen)
       - Setze die Variablenbelegung y in den Ausdruck x ein
       - für jede Klausel von x: prüfe, ob die Klausel wahr ist
     - Es gilt |y| = O(|x|) (ein Bit für jede Variable, die in x vorkommt)
     - Algorithmus A hat polynomielle Laufzeit $T_A(n) = O(n)$

     Es folgt, SAT ist polynomzeit-verifizierbar und somit SAT $\in$ NP

Das erste Problem in NPC

- Cook‘s Beweis (grobe Skizze)
  2. Zeige für ein beliebiges B $\in$ NP: B $\leq_P$ SAT

     Sei $A_B(x,y)$ der Algorithmus zur Polynomzeit-Verifikation von B

     $A_B(x,y)$ benötige T(n) Schritte auf einer RAM

     Die Funktionsweise einer RAM kann auf der Basis logischer Schaltkreise beschrieben werden

     Die Funktionsweise logischer Schaltkreise kann durch boole‘sche Formeln beschrieben werden

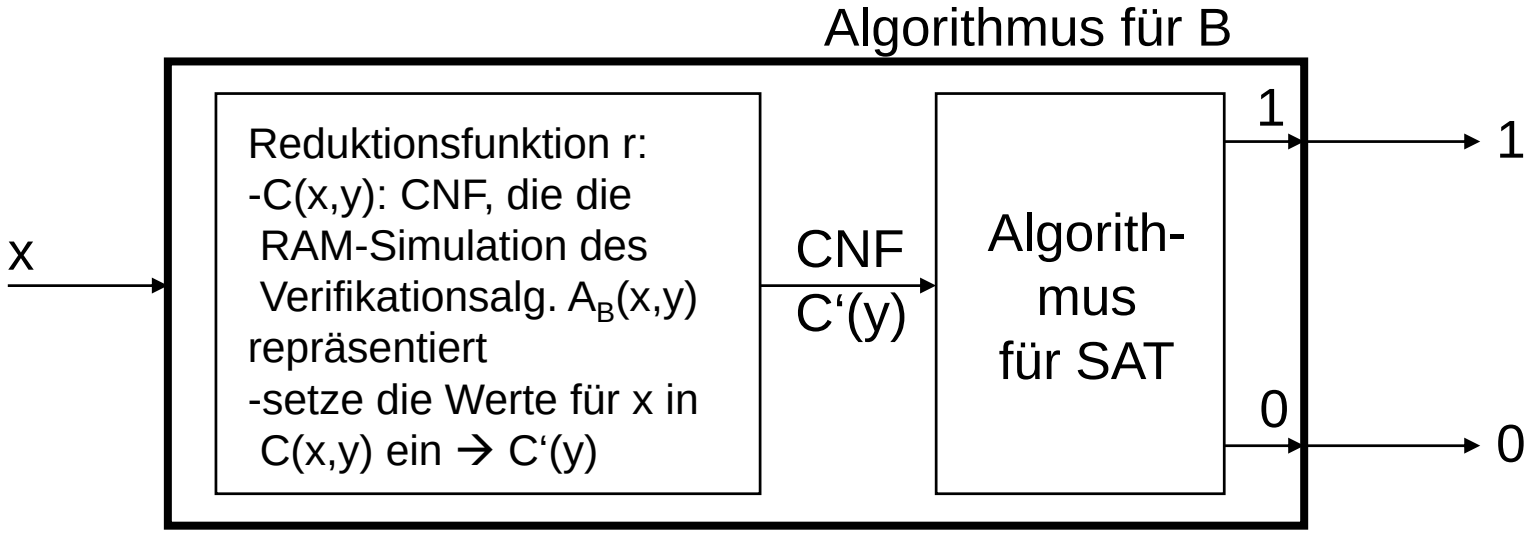
     Der Zustand einer RAM kann vollständig durch eine boole‘sche Formel in CNF beschrieben werden (modeliere Register, Akkumulator, Rechenwerk, Steuerwerk, etc.)

     Aus dem Zustand der RAM zum Zeitpunkt i kann der Zustand zum Zeitpunkt i+1 durch boole‘sche Formeln beschrieben werden.

     Eine vollständige Rechnung einer RAM mit t Schritten kann durch eine CNF polynomieller Länge beschrieben werden

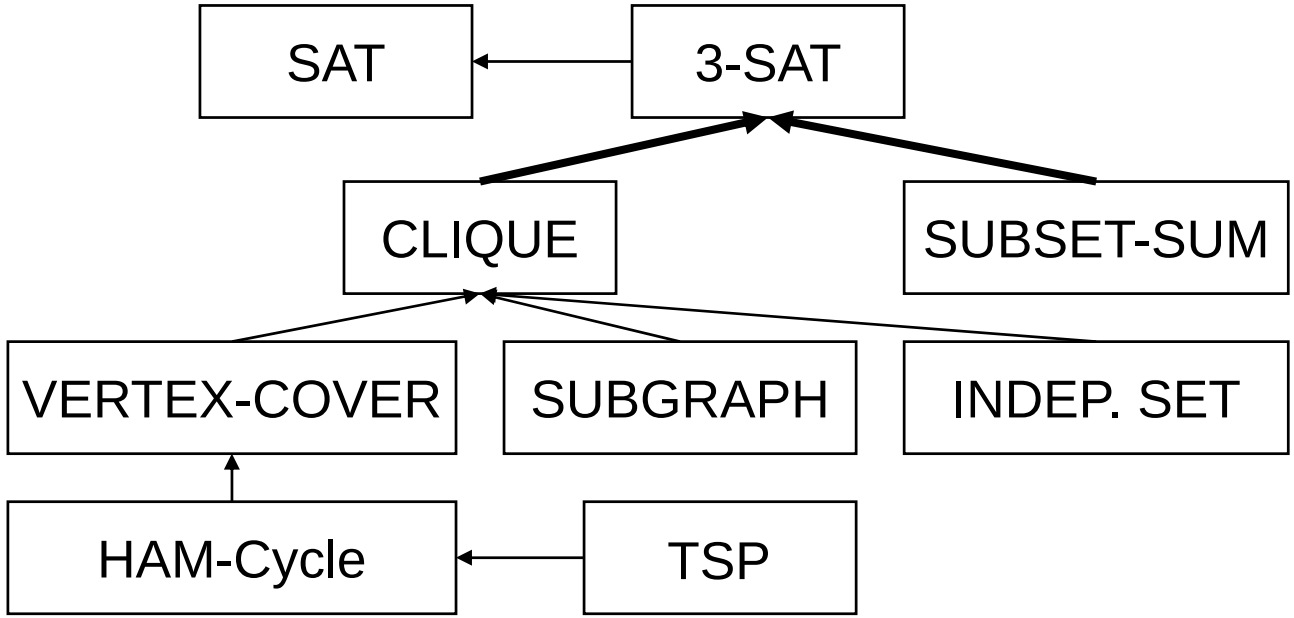
Das erste Problem in NPC

- Cook‘s Beweis (Fortsetzung)



- Ist $x \in B$:
  - so existiert ein y mit $A_B(x,y)=1$, für die Belegung y ist somit C‘(y)=1 => C‘(y) ist erfüllbar => $r(x)= C‘(y) \in SAT$
- Ist $x \notin B$:
  - so gilt für alle y: $A_B(x,y)=0$, für alle Belegungen ist somit C‘(y)=0 => C‘(y) ist nicht erfüllbar => $r(x)=C‘(y) \notin SAT$
- $|r(x)| = O(n^k)$ und r(x) kann aus x in $O(n^k)$ berechnet werden.

NP-vollständige Probleme



3-SAT: Jede Klausel hat maximal 3 Literale

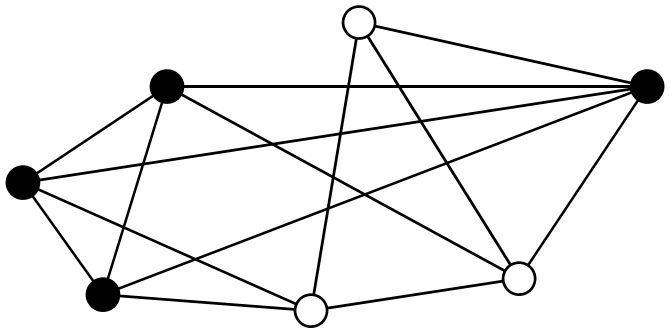
VERTEX-COVER: Knotenmenge V‘, |V‘| ≤ k, jede Kante ist zu mindestens einem Knoten in V‘ inzident

HAM-Cycle: Gibt es einen einfachen Kreis, der alle Knoten enthält

TSP: Traveling Salesperson Problem (kürzeste Rundtour in einem vollständigen, kantengewichteten Graphen)

Das CLIQUE-Problem

- CLIQUE = { <G,k> : G ist ein Graph mit einer k-Clique }
- Bsp.:



- CLIQUE ist NP-vollständig
  - Beweis Teil 1: Zeige CLIQUE $\in$ NP
    - Zertifikat: V‘ $\subseteq$ V: V‘ ist eine k-CLIQUE, es gilt |V‘| = O(|V|)
    - Verifikationsalgorithmus A(<G=(V,E),k>, V‘)

```
if( |V'| ≠ k ) return FALSE
for( each pair v,w ∈ V' )
  if( {v,w} ∉ E  ) return FALSE
return TRUE
```

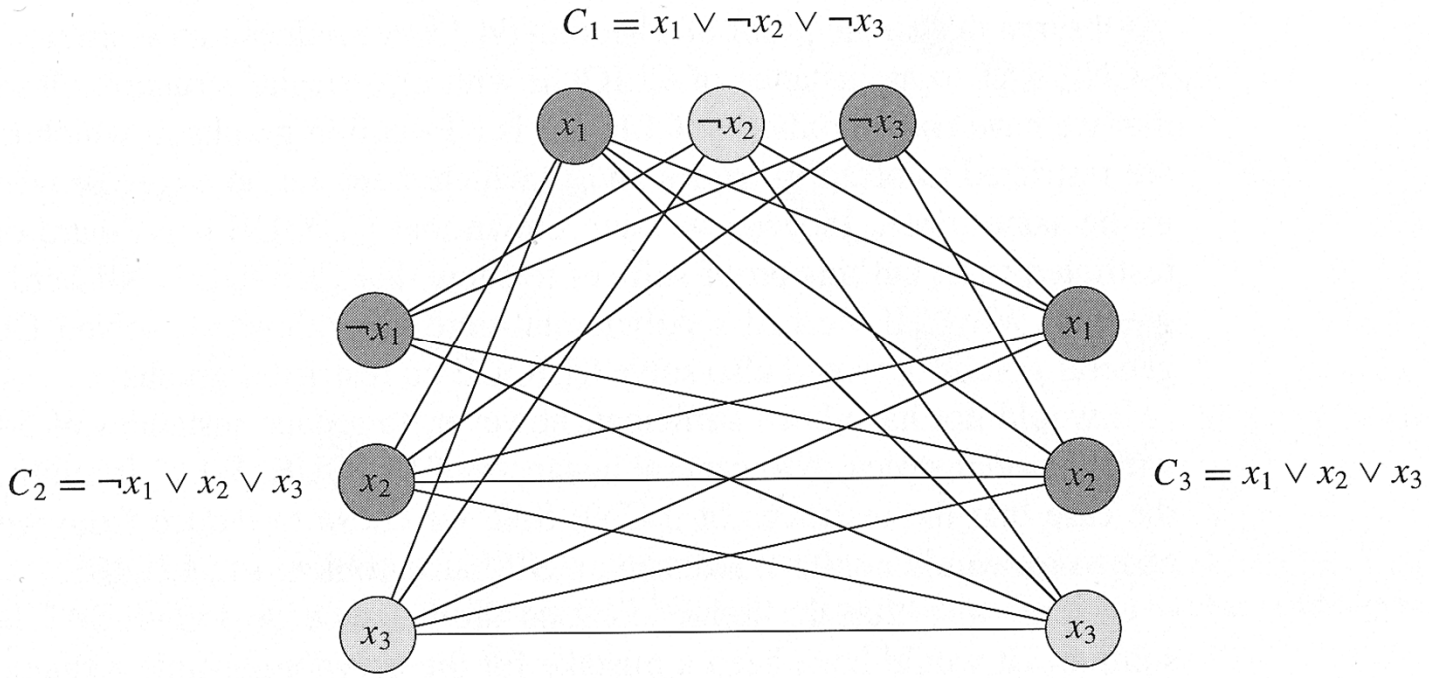
Das CLIQUE-Problem

- Beweis Teil 2: Zeige 3-SAT $\leq_p$ CLIQUE
  - Ziel: Wandle Eingabe x des 3-SAT Problems in Eingabe r(x) des CLIQUE-Problems mit x $\in$ 3-SAT <=> r(x) $\in$ CLIQUE
  - Eingabe des 3-SAT Problems: 3-CNF: $C_1 \wedge C_2 \wedge C_3 .... \wedge C_n$

    jede Klausel C_i: $l_i^1 \vee l_i^2 \vee l_i^3$ (l_i^j : j-te Literal der i-ten Klausel
  - Funktion r(x): baut aus 3-CNF einen Graph <G=(V,E),k>
    - Knoten V: v_i^j je ein Knoten pro Literal
    - Kanten E: v_i^r und v_j^s sind adjazent <=>
      1. zugehörige Literale gehören zu unterschiedlichen Klauseln, d.h. i $\neq$ j
      2. und zugehörige Literale können gleichzeitig erfüllt werden, d.h. $l_i^r \neq \neg\, l_j^s$
    - Clique-Größe: Anzahl der Klauseln, d.h. k= n
  - r(x) kann in $O(N^2)$ berechnet werden

Das CLIQUE-Problem

- Bsp.: Reduktionsfunktion r:
  - 3-CNF: $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$
  - Anzahl Klauseln: 3
  - Eingabe für CLIQUE: <G=(V,E), k>, k=3



Das CLIQUE-Problem

- Beweis Teil 2 (Fortsetzung):
  - Zeige $x \in$ 3-SAT => $r(x) \in$ CLIQUE
    - x ist erfüllbar, d.h. in jeder Klausel gibt es mindestens ein erfülltes Literal, d.h. $\forall\ 1 \leq i \leq n: \exists\ 1 \leq j \leq 3 : l_i^j = 1$
    - $V'' = \{ v_i^j \in V \mid l_i^j = 1 \}$, $V' \subseteq V''$: wähle pro Klausel genau ein Literal
    - $|V'| = k$, V' eine Clique, denn $\forall \{v_i^r, v_j^s\} \subseteq V'$:
      1. $i \neq j$ (pro Klausel wurde nur ein Literal gewählt)
      2. $l_i^r = 1$; $l_j^s = 1$ gemäß V'', also gilt $l_i^r \neq \neg\, l_j^s$
  - Zeige $r(x) \in$ CLIQUE => $x \in$ 3-SAT
    - Sei V' eine k-CLIQUE in r(x), L' die zugehörigen Literale
    - L' enthält aus jeder Klausel ein Literal, da $|V'|=k$ und Knoten zu Literalen innerhalb einer Klausel nicht adjazent sind.
    - Alle Literale aus L' können erfüllt werden, da Knoten zu inkonsistenten Literalen nicht adjazent sind.
    - Setze Variable $x_i = 1$ falls $x_i \in L'$, $x_i = 0$ falls $\neg x_i \in L'$, ansonsten beliebig.

      Die Belegung erfüllt die 3-CNF x.
  - Es gilt: r ist polynomzeit-berechenbar und $x \in$ 3-SAT <=> $r(x) \in$ CLIQUE, d.h 3-SAT $\leq_p$ CLIQUE

Das SUBSET-SUM Problem

- Problem (SUBSET-SUM)
  - geg.: Menge S von Zahlen, Zielwert t
  - Gibt es ein S‘ $\subseteq$ S mit $\sum_{s \in S'} s = t$

- Bsp.: t=300 S= { 3, 17, 39, 48, 103, 111, 113, 132, 254 }
  - S‘= { 17, 48, 103, 132 }

  - arithmetisches Problem, Größe der Zahlen muss bei der Komplexitätsanalyse berücksichtigt werden
- Theorem 34.15:

  SUBSET-SUM ist NP-vollständig
  - Beweis Teil 1: SUBSET-SUM $\in$ NP
    - Zertifikat y: Indizes der Elemente in S, die zu S‘ gehören
    - A(<S,t>, y): addiere die Elemente in S‘ und vergleiche mit t
      - Laufzeit: O(N)

Das SUBSET-SUM Problem

- Beweis Teil 2: 3-SAT $\leq_p$ SUBSET-SUM
  - Ziel: Wandle Eingabe x des 3-SAT Problems in Eingabe r(x) des SUBSET-SUM Problems, so dass x erfüllbar ist g.d.w. r(x) eine Teilsumme mit Wert t hat.
  - Sei F eine logischer Ausdruck in 3-CNF
    - Entferne alle Klauseln die Variable und ihr Komplement enthalten (sind sowieso immer erfüllt)
    - Entferne alle Variablen, die in keiner Klausel vorkommen (spielen bzgl. der Erfüllbarkeit keine Rolle)
  - F bestehe aus den Variablen $x_1,...,x_n$ und den Klauseln $C_1,...,C_k$
  - Reduktionsfunktion r:
    - verwende Zahlen im Zehnersystem mit n+k Stellen
    - die ersten n Stellen repräsentieren Variablen, die folgenden k Stellen repräsentieren Klauseln
    - konstruiere je zwei Zahlen v_i, v_i‘ und s_j,s_j‘ für Variablen und Klauseln:

Das SUBSET-SUM Problem

- Beweis Teil 2: (Fortsetzung)
  - v_i: 1 an Stelle x_i, 1 an Stellen aller Klauseln, die x_i enthalten
  - v_i‘: 1 an Stelle x_i, 1 an Stellen aller Klauseln, die $\neg x_i$ enthalten
  - s_j: 1 an Stelle C_j s_j‘: 2 an Stelle C_j t: $\underbrace{111...1}_{n \text{ mal}}\underbrace{444...4}_{k \text{ mal}}$

- Beispiel: 3-CNF:

$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge$ C_1

$(\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge$ C_2

$(\neg x_1 \vee \neg x_2 \vee x_3) \wedge$ C_3

$(x_1 \vee x_2 \vee x_3)$ C_4

VAR:	x1	x2	x3	C1	C2	C3	C4
v1	1	0	0	1	0	0	1
v1‘	1	0	0	0	1	1	0
v2	0	1	0	0	0	0	1
v2‘	0	1	0	1	1	1	0
v3	0	0	1	0	0	1	1
v3‘	0	0	1	1	1	0	0
s1	0	0	0	1	0	0	0
s1‘	0	0	0	2	0	0	0
s2	0	0	0	0	1	0	0
s2‘	0	0	0	0	2	0	0
s3	0	0	0	0	0	1	0
s3‘	0	0	0	0	0	2	0
s4	0	0	0	0	0	0	1
s4‘	0	0	0	0	0	0	2
t	1	1	1	4	4	4	4

Das SUBSET-SUM Problem

- Beweis Teil 2: (Fortsetzung)
  - r(x) hat Länge $O((n+k)^2)$ und kann in $O((n+k)^2)$ Zeit berechnet werden
  - Zeige x $\in$ 3-SAT => r(x) $\in$ SUBSET-SUM

    x ist erfüllbar; sei B eine Belegung, die x erfüllt.

    Wähle S‘ = { v_i | x_i = 1 in B } $\cup$ {v_i‘ | x_i = 0 in B}, sei $t' = \sum_{s \in S'} s$

    t‘ hat an den ersten n Stellen eine 1, da entweder

    $v_i \in$ S‘ oder v_i‘$\in$ S‘

    t‘ hat an den hinteren k Stellen eine 1, 2 oder 3, da jede Klausel erfüllt ist (daher > 0) und jede Klausel max. 3 Literale hat (daher $\leq$ 3)

    Erweitere S‘ um Variablen s_i und/oder s_i‘, so dass an den hinteren k Stellen je eine 4 steht.

    Offensichtlich gilt $\sum_{s \in S'} s = t$.

Das SUBSET-SUM Problem

- Beweis Teil 2: (Fortsetzung)
  - Zeige: r(x) $\in$ SUBSET-SUM => x $\in$ 3-SAT

    Sei S‘ die Teilmenge der Zahlen mit $\sum_{s \in S'} s = t$

    Um in der Summe an der i-ten Stelle eine 1 zu erhalten, muss gelten: entweder $v_i \in S'$ oder $v_i' \in S'$.

    Wähle die Belegung x_i=1 falls $v_i \in S'$, x_i=0 falls $v_i' \in S'$.

    Für jede Klausel C_j gilt: die n+j-te Stelle ist 4, da $s_j + s_j' = 3$ gibt es ein v_i oder v_i' mit einer 1 an der n+j-ten Stelle.

    Angenommen, es ist ein v_i:

    x_i =1 und kommt in C_j vor, damit ist C_j erfüllt.

    Angenommen, es ist ein v_i':

    x_i =0 und $\neg x_i$ kommt in C_j vor, damit ist C_j erfüllt.
- Es gilt: r ist polynomzeit-berechenbar und x $\in$ 3-SAT <=> r(x) $\in$ SUBSET-SUM, d.h 3-SAT $\leq_p$ SUBSET-SUM.

7.4 Noch mehr NP-vollständige Probleme

- Beispiele aus
  - Garey/Johnson, Computers and Intractability (1979)

- Graphen:
  - Aufteilung in kantendisjunkte Dreiecke
  - Aufteilung in weniger als k kantendisjunkte Bäume
  - Gibt es einen bipartiten Subgraph mit mehr als K Kanten?
  - Gibt es einen planaren Subgraph mit mehr als K Kanten?
  - Enthält ein Graph G einen gegebenen Subgraph H?
  - Gibt es einen Spannbaum, in dem jeder Knotengrad $< K$ ist?
  - Gibt es einen Spannbaum, in dem die Summe über allen paarweisen Distanzen zwischen Knoten $< K$ ist?

Noch mehr NP-vollständige Probleme

- Netzwerk-Design
  - (K-th SHORTEST PATH)

    Geg. Graph mit positiven Kantengewichten, Start- und Zielknoten, zwei Zahlen K und B

    Gibt es K kantendisjunkte Wege von s nach t, jeder mit Länge < B?

    Das Problem ist polynomzeit-reduzierbar, es ist unbekannt, ob es in NP ist.

  - QUADRATIC ASSIGNMENT

    Geg. n Objekte mit paarweisen Abstandskosten c_{ij}, m Slots mit paarweisen Abständen d_{ij}

    Gibt es eine Zuordnung f:{1,...,n} →{1,...,m} der Objekte zu den Slots mit $\sum_{\{i,j\}\subseteq\{1,\ldots,n\}} c_{ij} d_{f(i)f(j)} \leq B$

Noch mehr NP-vollständige Probleme

- Mengen und Partitionen
  - MINIMUM COVER

    Geg. eine Sammlung C von Teilmengen über eine endl. Menge S, eine positive Zahl K

    Gibt es ein C‘ $\subseteq$ C mit |C‘| $\leq$ K, die S überdeckt, d.h. $\bigcup_{c \in C'} c = S$

  - BIN PACKING

    Geg. Menge U von Objekten mit positiver Größe, eine positive Behältergröße B, eine positive Zahl K

    Gibt es eine Aufteilung von U in $\leq$ K Teilmengen, so dass für jede Teilmenge U_i gilt: $\sum_{u \in U_i} u \leq B$

Noch mehr NP-vollständige Probleme

- Zeichenketten
  - SHORTEST COMMON SUPERSEQUENCE

    Geg. eine Menge R von Strings, eine positive Zahl K

    Gibt es einen String s mit $|s| \leq K$, der alle $r \in R$ als Teilsequenz (mit Unterbrechungen) enthält, d.h. $s = w_0 r_0 w_1 r_1 ... w_k r_k w_{k+1}$ mit $r_0 r_1 ... r_k = r$?
  - SHORTEST COMMON SUPERSTRING

    Geg. eine Menge R von Strings, eine positive Zahl K

    Gibt es einen String s mit $|s| \leq K$, der alle $r \in R$ als Teilstring (ohne Unterbrechungen) enthält, d.h. $s=w_0 r w_1$?
  - LONGEST COMMON SUBSEQUENCE

    Geg. eine Menge R von Strings, eine positive Zahl K

    Gibt es einen String s mit $|s| \geq K$, der eine Teilsequenz aller Strings in R ist?

Noch mehr NP-vollständige Probleme

- Scheduling
  - MULTIPROCESSOR SCHEDULING

    Geg. Anzahl m von Prozessoren, Menge T von Aufgaben, eine positive Bearbeitungsdauer z(t) für jede Aufgabe und eine positive Zahl D (Deadline)

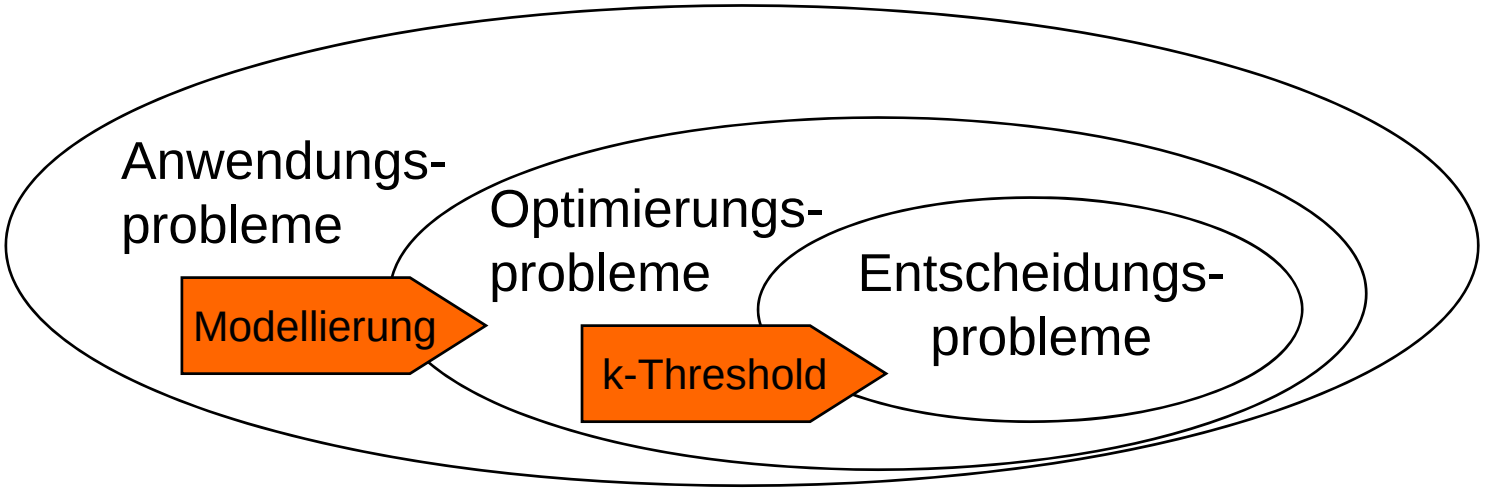
    Können die Aufgaben aus T auf die m Prozessoren verteilt werden, so dass alle Aufgaben bis zum Zeitpunkt D bearbeitet sind?
  - TIMETABLE DESIGN (informell)

    Gibt es einen gültigen Stundenplan für H Arbeitsperioden, C Handwerker und T Aufgaben
- Packungsprobleme
  - KNAPSACK

    Geg. eine Menge U von Objekten mit einer Größe s(u) > 0 und einem Wert v(u) > 0, zwei positive Zahlen B und K

    Gibt es eine Teilmenge U‘ von U mit $\sum_{u \in U'} s(u) \leq B$ und $\sum_{u \in U'} v(u) \geq K$

Komplexität von Anwendungsproblemen



- Kompexitätsaussagen erfolgen über Entscheidungsprobleme
- Optimierungsprobleme:
  - k-Threshold-Problem (+ binäre Suche) definiert einen klaren Bezug zu Entscheidungsproblem
  - Komplexitätsaussage hat uneingeschränkte Gültigkeit
- Anwendungsprobleme:
  - Modellierung (Was sind die Freiheitsgrade, Was ist eine Energiefunktion, etc.) definiert ein Optimierungsproblem
  - Komplexitätsaussage hat nur eingeschränkte Gültigkeit

  „Problem X ist NP-schwer unter einer gegebenen Modellierung“