

## 8. NP

### Ansatz

Problem ist leicht, wenn es in Polynomialzeit lösbar ist.

Worst-Case-Laufzeit des Algorithmus ist also  $\Theta\left(\sum_{i=0}^k a_i n^i\right) = \text{poly}(n)$  mit konstanten  $a_i, k$

Leicht zu lösende Problem:

- Sortieren eines Arrays
  - Breitensuche im Graphen
  - Minimale Spannbäume berechnen
  - ...
- Probleme mit leicht zu überprüfender Lösung:
- TSP
  - Faktorisieren
  - ...
- Unentscheidbare Probleme:
- Halteproblem
  - Code-Erreichbarkeit
  - ...

## Berechnungsprobleme vs. Entscheidungsprobleme

Berechnungsproblem:

- Gegeben: Problem  $P$
- Gesucht: Lösung  $S$
- Beispiel: Berechne kürzeste Pfade im Graphen

Entscheidungsproblem:

- Gegeben: Problem  $P$
- Gesucht: Hat  $P$  Eigenschaft  $E$ ? Antwort ist wahr/falsch
- Beispiel: Ist gerichteter Graph stark zusammenhängend?

Im Folgenden werden nur Entscheidungsprobleme betrachtet

Man kann jedes Berechnungs- in ein Entscheidungsproblem überführen, so dass Polynomialzeit-Lösung für Entscheidungsproblem auch Polynomialzeit-Lösung für Berechnungsproblem ergibt.

### Beispiel: Faktorisieren

## Faktorisierungsproblem (Berechnungsproblem):

- Gegeben:  $n$ -Bit Zahl  $N \geq 2$
- Gesucht: Primfaktoren von  $N$   
 $\Rightarrow$  Entscheidungsproblem:
- Gegeben:  $n$ -Bit Zahl  $N \geq 2$ , Zahl  $B$
- Gesucht: Ist kleinster Primfaktor von  $N$  maximal  $B$ ?

```
Factorize(N) // N>1
    WHILE N>1 DO
        p=computeFactor(N);
        print p;
        N=N/p;

computeFactor(N) // use decideFactor(N,B) as sub, N>1, computes prime factor of N
    L=1; U=N;
    WHILE L!=U DO
        M=L+floor((U-L)/2);
        IF decideFactor(N,M)==1 THEN U=M ELSE L=M+1;
    return L;

decideFactor(N,B)
    ...
    return d; // d==0 or d==1
```

- **sub** steht für Subroutine

In jeder Iteration wird Suchintervall um Hälfte reduziert, runden kann man ignorieren

Zu Beginn Intervalllänge  $N$ , also nach  $\Theta(\log_2 N) = \Theta(n)$  Iterationen fertig

Laufzeit  $\Theta(\log_2 N) = \Theta(n)$  Iterationen von **decideFactor**, in jeder Iteration konstanter Aufwand

Laufzeit **Factorize**:

In jeder Iteration wird Primfaktor  $p \geq 2$  abgespalten, also maximal  $\Theta(\log_2 N) = \Theta(n)$  Iterationen

Annahme der Laufzeit von **decideFactor**:  $\text{poly}(n)$

Gesamtlaufzeit  $\Theta(n^2 \cdot \text{poly}(n))$

## Berechnung durch Entscheidung

### Berechnungsproblem:

- Gegeben: Problem  $P$
- Gesucht: Lösung  $S$   
Kreiere daraus Entscheidungsproblem:

- Gegeben: Problem  $P$ , String  $s$
- Gesucht: Ist  $s$  Präfix der Binärdarstellung einer Lösung  $S$ ?

```

compute(P) // use decide(P,s) as sub
    s=""; // empty string
    IF decide(P,s)==0 THEN return "no solution";
    done=false;
    WHILE !done DO
        szero=decide(P,s+"0");
        sone =decide(P,s+"1");
        IF szero==0 AND sone==0 THEN // solution found
            done=true
        ELSE IF szero==1 THEN s=s+"0" ELSE s=s+"1";
    return "solution " + s;

decide(P,s)
    ...
    return d; // d==0/d==1

```

Sofern Bitlänge der Lösungen polynomiell beschränkt ist und `decide` in Polynomialzeit läuft, läuft `compute` auch in Polynomialzeit

Es wird bit-weise in richtige Richtung gesucht

Laufzeit:  $\Theta \left( 2 \cdot \max_S |S| + 1 \right)$  Iterationen von `decide`

## Komplexitätsklassen P und NP

### Komplexitätsklasse P

Betrachte Entscheidungsproblem für Eigenschaft als Menge:

$L_E := \{P : P \text{ hat Eigenschaft } E\}$

$L$  kommt von "language"

Beispiel:  $L_{Sc} := \{G : G \text{ ist gerichtetet, stark zusammenhängender Graph}\}$

Komplexitätsklasse P:

Ein Entscheidungsproblem  $L_E$  ist genau dann in der Komplexitätsklasse P, wenn es einen Polynomialzeit-Algorithmus  $A_{L_E}$  mit Ausgabe 0/1 gibt, der stets korrekt entscheidet, ob eine Eingabe  $P$  die Eigenschaft  $E$  hat oder nicht, also  $P \in L_E \Leftrightarrow A_{L_E}(P) = 1$  für alle  $P$  gilt.

Eigentliche Definition: Algorithmus = Turing-Maschine und Problem-Universum =  $\{0, 1\}^*$

### Komplexitätsklasse NP

Das Prüfen einer vermeintlichen Lösung ist einfach für  $L_E$ :

- Gegeben: Problem  $P$  und vermeintliche Lösung  $S$

- Entscheide: Zeigt  $S$ , dass  $P$  Eigenschaft  $E$  hat oder nicht?
- $S$  dient als zusätzliche Entscheidungshilfe, heißt auch "witness", Zeuge, Zertifikat,... für  $P$

Technische Einschränkung:

Lösungen  $S$  sind von polynomieller Komplexität in Eingabeproblem  $P$ , meist: Lösungen  $S$  haben polynomielle Bitlänge (in Bitlänge von  $P$ )

## Beispiel

$L_{Fakt} := \{(N, B) : N \text{ hat Primfaktor} \leq B\}$

Gegenwärtig ist es unklar, wie in Polynomialzeit ohne Hilfe (und ohne Quantencomputer) entschieden werden kann, ob Eingabe  $(N, B)$  in  $L_{Fakt}$  ist oder nicht

Mit Hilfe ist das Entscheiden einfach:

Zeuge  $S$  zu  $P = (N, B)$  ist Faktor  $p$  von  $N$  mit  $1 < p \leq B$

```
verify(N,B,p) // check alleged solution
1 IF N>1 AND 1<p<=B AND p|N THEN return 1 else return 0;
```

Es wird nicht geprüft, ob  $p$  prim ist, wenn der zusammengesetzte Faktor in der Schranke  $B$  liegt, dann ist  $p$  erst recht ein Primfaktor

Es gibt keine falsche Hilfe für nicht-zugehörige Eingaben:

- Wenn  $(N, B) \in L_{Fakt}$ , dann gibt es ein  $S$ , das `verify` akzeptieren lässt
  - Wenn  $(N, B) \notin L_{Fakt}$ , dann gibt es kein  $S$ , das `verify` akzeptieren lässt
- Entscheidung mit Hilfe muss in beiden Fällen richtig sein

## NP (Nicht-deterministische Polynomialzeit)

Ein Entscheidungsproblem  $L_E$  ist in der Komplexitätsklasse NP gdw. es einen Polynomialzeit-Algorithmus  $A_{L_E}$  mit Ausgabe 0/1 gibt, der bei Eingabe eines Zeugen  $S_P$  für Eingabe  $P \in L_E$  bzw. für jede Eingabe  $S_P$  für Eingabe  $P \notin L_E$  stets korrekt entscheidet, ob eine Eingabe  $P$  die Eigenschaft  $E$  hat oder nicht, also  $P \in L_E \Leftrightarrow \exists S_P : A_{L_E}(P, S_P) = 1$  f.a.  $P$  gilt.

Äquivalent: F.a.  $P$  gilt  $P \notin L_E \Leftrightarrow \forall S_P : A_{L_E}(P, S_P) = 0$

Komplexität der Hilfseingabe  $S_P$  polynomiell in der von  $P$

## P vs. NP

Jedes Problem in P ist auch in NP: Algorithmus  $A_{L_E}$  entscheidet ohne Hilfe  $\rightsquigarrow P \subseteq NP$

Bis heute ist offen, ob auch  $NP \subseteq P$  gilt

## Mögliche Welten

Faktorisieren ist in NP, jedoch ist nicht klar, ob Faktorisieren auch in P liegt.

1. Wahrscheinlichste Welt, Bild wird durch Quantum-Computer verfeinert:  
 $P \neq NP$ , Faktorisieren  $\notin P \rightsquigarrow$  Faktorisieren schwierig
2.  $P \neq NP$ , Faktorisieren  $\in P \rightsquigarrow$  Faktorisieren leicht
3.  $P = NP \rightsquigarrow$  Alle Probleme sind leicht

## NP-Vollständigkeit

### Ziel: Identifiziere schwierigsten Probleme in NP

NPC (NP-Complete): Klasse der NP-vollständigen Probleme

Eigenschaften:

1.  $NPC \subseteq NP$
2. Wenn  $P \neq NP$ , dann definitiv  $NPC \not\subseteq P$

### Reduktionen (Problemtransformationen)

Siehe [Berechnung durch Entscheidung](#) für Problemdefinition

Wenn das Entscheidungsproblem leicht ist, ist das Berechnungsproblem es auch

Das Entscheidungsproblem ist mindestens so schwierig wie das Berechnungsproblem

### Transfer auf NP-Entscheidungsprobleme

NP-Problem  $L_A$ :

- Gegeben: Problem  $P$
- Gesucht: Entscheidung

Reduktion auf NP-Problem  $L_B$ :

- Gegeben: Problem  $Q$
- Gesucht: Entscheidung

Die Reduktion von  $L_A$  auf  $L_B$  ist Polynomialzeit-Algorithmus  $R$ , sodass gilt:

$P \in L_A \Leftrightarrow R(P) \in L_B$  f.a.  $P$ , Schreibweise:  $L_A \leq L_B$

Die Reduktion transformiert Problem  $P$  in Problem  $Q = R(P)$ , sodass eine korrekte Entscheidung für  $Q$  automatisch eine korrekte Entscheidung für  $P$  liefert

```
decideA(P)
    Q=R(P);
    return decideB(Q);

decideB(Q)
    ...
    return d; // boolean
```

### NP-vollständige Probleme

Komplexitätsklasse NPC (NP-vollständige Probleme):

Alle Probleme  $L_C \in \text{NP}$ , sodass  $L_A \leq L_C$  f.a.  $L_A \in \text{NP}$

Zwei Bedingungen an  $L_C$ :

1.  $L_C \in \text{NP}$
2. jedes NP-Problem ist auf  $L_C$  reduzierbar ( $L_C$  ist NP-hart)

## Beispiel für Reduktion: Hamiltonscher Zyklus $\leq$ TSP

- HamCycle für  $G$ :  
Gibt es Tour (jeden Knoten einmal besuchen und zu Startknoten zurück) im Graphen  $G$  ?
- TSP für  $(G, B)$ :  
Gibt es Tour im Graphen  $G$  mit Gewicht maximal  $B$ ?  
Beide Probleme sind in NP  
Reduktion:  
Existierende Kanten bekommen Gewicht 0, vervollständige anschließend Graphen mit Kanten mit Gewicht 1, setze  $B = 0$   
Nun zu zeigen:  $G \in \text{HamCycle} \Leftrightarrow R(G) = (G^*, B) \in \text{TSP}$   
#TODO maybe add proof

## SAT: Die Mutter aller NP-vollständigen Probleme

Gegeben:

Boolesche Formel  $\phi$  aus  $\vee, \wedge, \neg$  in  $n$  Variablen  $x_1, x_2, \dots, x_n$

$\phi$  hat polynomielle Komplexität in  $n$

Gesucht:

Entscheide, ob  $\phi$  erfüllende Belegung hat oder nicht

SAT  $\in$  NP: Gegeben ist Belegung als Zeuge, werte Formel aus

## SAT ist NP-hart

#TODO elaborate further

Reduktion  $R(P)$  von  $L_A$  auf SAT berechnet:

$\phi_P(\text{alle Eingabebits}) = \text{gültiger Anfangszustand für } P \wedge \text{gültige Übergänge} \wedge \text{Endzustand mit } d=1$

Wenn  $P$  in  $L_A$  ist, gibt es eine Lösung  $S$ , die `verifyA` mit  $d = 1$  akzeptiert, dann gibt es aber auch eine erfüllende Belegung für "Rechenschritte"  $\phi_P$

Wenn  $P$  nicht in  $L_A$  ist, gibt es keine Lösung  $S$ , die `verifyA` akzeptiert, dann gibt es aber auch keine erfüllende Belegung für "Rechenschritte"  $\phi_P$

## SAT $\leq$ 3SAT

Boolesche Formeln in konjunktiver Normalform (KNF) mit jeweils 3 Literalen:

$$\phi(x_1, x_2, x_3, x_4) = (\neg x_2 \wedge x_3 \wedge x_4) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_4 \wedge x_3 \wedge x_4)$$

KNF = Und-Verknüpfung von Klauseln, Klausel = Oder-Verknüpfung

Klausel besteht aus 3 Literalen  $x_j \in \{x_j, \neg x_j\}$

Falls weniger Literale in Klausel, transformiere:

$(x_j) = (x_j \wedge x_j \wedge x_j), (x_j \wedge x_k) = (x_j \wedge x_k \wedge x_k)$

3SAT:

Gegeben:

Boolesche 3KNF-Formel  $\phi$  in  $n$  Variablen  $x_1, x_2, \dots, x_n$ ,  $\phi$  hat polynomielle Komplexität in  $n$

Gesucht:

Entscheide, ob  $\phi$  erfüllende Belegung hat oder nicht

SAT: Boolesche Formel  $\sigma$  aus  $\vee, \wedge, \neg$  in  $n$  Variablen  $y_1, y_2, \dots, y_n$  ( $\sigma$  polynomielle Komplexität in  $n$ )

lässt sich in Polynomialzeit überführen zu

3SAT: 3KNF-Formel  $\phi$  in  $poly(n)$  Variablen  $x_1, x_2, \dots, x_{poly(n)}$  ( $\phi$  polynomielle Komplexität in  $n$ )

sodass  $\sigma$  erfüllbar ist gdw.  $\phi$  erfüllbar ist  $\rightsquigarrow \text{SAT} \leq 3\text{SAT}$

## 3-Färbbarkeit von Graphen

3COLORING für  $G$ :

Gibt es eine Knotenfärbung im Graphen  $G$  mit 3 Farben, sodass benachbarte Knoten nie die gleiche Farbe haben?

3COLORING  $\in$  NP:

Gegeben Färbung, durchlaufe Knoten und prüfe jeweils Farbe der Nachbarknoten

## 3SAT $\leq$ 3COLORING

#TODO add proof maybe

## Einer für alle, alle für einen

Wenn Problem  $L_B$  NP-vollständig ist und  $L_B \leq l_C$  für  $L_C \in \text{NP}$  gilt, dann ist auch  $L_C$  NP-vollständig.

$L_A \leq L_B$  per Reduktion  $R_{AB}$ ,  $L_B \leq L_C$  per Reduktion  $R_{BC} \rightsquigarrow L_A \leq L_C$  per Reduktion  $R_{AC} := R_{BC} \circ R_{AB}$  (Hintereinanderausführung)

Also folgt aus  $3\text{SAT} \leq 3\text{COLORING}$  und  $3\text{COLORING} \in \text{NP}$  auch, dass 3COLORING NP-vollständig ist.

## NPC – eine Auswahl

- SAT: Ist Formel  $\phi$  erfüllbar?
- 3SAT: Ist Formel  $\phi$  in 3KNF erfüllbar?
- 3COLORING: Ist Graph mit 3 Farben kantenkonsistent färbbar?
- HamCycle: Gibt es eine Tour im Graphen?
- TSP: Gibt es eine Tour im Graphen, mit Gesamtgewicht  $\leq B$ ?

- VertexCover: Gibt es im Graphen eine Knotenmenge der Größe  $\leq B$ , sodass jede Kante an einem der Knoten hängt?
- IndependentSet: Gibt es im Graphen Knotenmenge der Größe  $\geq B$ , sodass kein Knotenpaar durch eine Kante verbunden ist?
- Knapsack: Für Gegenstände mit Wert und Volumen, gibt es eine Auswahl mit Gesamtwert  $\geq W$ , aber Gesamtvolumen  $\leq V$ ?
- ...

## P vs. NP vs. NPC

Für jedes NP-vollständige Problem  $L_C$  gilt:  $L_C \in P \Leftrightarrow P=NP$ .

Wenn es also einen Polynomialzeit-Algorithmus für ein  $L_C \in NPC$  gibt, dann gibt es einen Polynomialzeit-Algorithmus für jedes Problem in NP.

#TODO maybe add proof

## Approximation

NPC-Probleme sind vermutlich nicht effizient lösbar, aber eventuell leicht approximierbar

```
3SAT-Approx( $\phi$ ,  $n$ )
    A[]=ALLOC( $n$ ); // assignment for variables
    FOR i=1 TO  $n$  DO
        A[i]=true resp. A[i]= false with probability 1/2
    return A;
```

#TODO add explanation

## 2-Färbbarkeit und 2SAT in P

### 2-Färbbarkeit von Graphen ist relativ einfach

Idee:

- Farbe eines Knoten bestimmt eindeutig Farben der Nachbarknoten
  - Prüfe jeweils, ob Färbung Widerspruch erzeugt
- Ansatz:
- Beginne mit einem Knoten und beliebiger Farbe
  - Durchlaufe den Graphen per BFS, färbe Knoten und identifiziere eventuelle Widersprüche

```
2ColoringSub( $G, s, col$ ) //  $G=(V, E)$ ,  $s$  node
     $s.color=col$ ; newQueue( $Q$ ); enqueue( $Q, s$ );
    WHILE !isEmpty( $Q$ ) DO
```



```

u=dequeue(Q);
IF u.color==BLACK THEN nextcol=RED // change colour
ELSE nextcol=BLACK;
FOREACH v in adj(G,u) DO
    IF v.color==u.color THEN return 0; // check for contradiction
    If v.color==WHITE THEN // only accept nodes without colour
        v.color=nextcol;
        enqueue(Q,v);
return 1; // no contradiction

```

Zunächst nur für zusammenhängenden Graphen mit vorgegebenem Startknoten und vorgegebener Startfarbe

Man muss eventuell mit anderem Startknoten nochmal starten, wie ist die Farbe zu wählen?

## Von gerichtet zu ungerichtet

Betrachte ungerichteten Graphen, Lösungsmenge ändert sich nicht

Bei Neustart keine Kante zwischen Zusammenhangskomponenten:

Jede individuelle 2-Färbung der Zusammenhangskomponenten kann zu 2-Färbung des Graphen kombiniert werden

```

2Coloring(G) // G=(V,E) undirected graph
FOREACH u in V Do u.color=WHITE;
FOREACH u in V DO
    IF u.color==WHITE THEN
        IF 2ColoringSub(G,u,BLACK)==0 THEN return 0;
return 1;

```

Algorithmus findet ohne zusätzlichen Aufwand auch Färbung

Laufzeit  $\Theta(|V| + |E|)$

## 2-SAT

Keine Symmetrie zwischen Belegungen und Farben bei 2-Färbbarkeit

## Implikationsgraph aus 2-SAT

Konstruiere aus Formel  $\phi$  (gerichteten) Implikationsgraphen  $G = (V, E)$ :

1. Knotenmenge  $V$  besteht aus Literalen  $x_1, \neg x_1, x_2, \neg x_2, \dots, x_n, \neg x_n$
2. Für jede Klausel  $(x_j \wedge x_k)$  nehme Kanten  $(\neg x_j, x_k)$  und  $(\neg x_k, x_j)$  auf

## Starke Zusammenhangskomponenten im Implikationsgraphen

Formel ist erfüllbar gdw. in keiner Zusammenhangskomponenten  $x_j, \neg x_j$  für ein  $j$  liegen

#TODO maybe add proof

## Erfüllende Belegung berechnen

Annahme: kein  $x_j$  und  $\neg x_j$  in gleicher SCC

SCC-dag:

Graph mit Superknoten aus allen Knoten einer SCC

Es gibt eine Kante zwischen SCCs, wenn es eine Kante für zwei Knoten aus den SCCs

1. Sortiere SCC-dag topologisch
2. Erfüllende Belegung (wohldefiniert, da kein  $x_j$  und  $\neg x_j$  in gleicher SCC):

$x_j = \text{true}$ , wenn  $x_j$  in SCC nach SCC mit  $\neg x_j$

$x_j = \text{false}$ , wenn  $\neg x_j$  in SCC nach SCC mit  $x_j$

#TODO maybe add proof

## MAX-2SAT

Gegeben: 2SAT-Formel  $\phi$ , Zahl  $k$

Gesucht: Gibt es eine Belegung, die mindestens  $k$  Klauseln erfüllt?

MAX-2SAT  $\in$  NPC

MAX-2SAT  $\in$  NP:

Gegeben ist eine Belegung als Zeuge; prüfe, ob mindestens  $k$  Klauseln erfüllt werden

3SAT  $\leq$  MAX-2SAT

#TODO maybe add proof