Föreläsning 8+9: NP-problem

Formalisering av "rimlig tid"

- En algoritm som har körtid $O(n^k)$ för någon konstant k är rimligt snabb.
- En algoritm som har körtid $\Omega(c^n)$ för någon konstant $c > 1$ är för långsam.

Begreppet "effektiv algoritm" är alltså synonymt med "går i polynomisk tid" i den här kursen.
Är detta en rimlig uppdelning?

Formell definition av polynomiella problem

Ett *formellt språk* L är en mängd av strängar.
Exempel:
{"abc", "qwerty", "xyzzy"}
{binära strängar av udda längd}
{binära strängar som beskriver ett primtal}
{syntaktiskt korrekta C-program}
Ett språk kan beskrivas på olika sätt:

- Uppräkning av strängarna i språket.
- Uppsättning regler som definierar strängarna i språket (d.v.s. en *grammatik*).
- Algoritm som känner igen strängarna i språket.

Varje beslutsproblem motsvarar ett språk:
Språket som består av alla ja-instanser.
Omvänt säger vi att algoritmen A *känner igen* språket L om

$$A(x) = \text{Ja om } x \in L,$$
$$A(x) = \text{Nej om } x \notin L.$$

A tar *polynomisk tid* om det finns en konstant k så att $A(x)$ tar tid $O(|x|^k)$.
$\mathbf{P} = \{L : \exists A \text{ som löser } L \text{ i polynomisk tid}\}$
Dessa problem (språk) är de vi i kursen
betraktar som lätta.
Finns det svåra problem?: Två svar kan ges.
1. Det finns *artificiella* problem som man kan bevisa att de inte ligger i $\mathbf{P}$.
2. Det finns *naturliga* problem som man antar inte ligger i $\mathbf{P}$.

Exempel på (troligen) svåra problem

Schemaläggning:
KTH har, som vanligt, fått fler elever utan att ha tillgång till mer salar. Schemaläggarna måste ta hänsyn till ett stort antal villkor när de lägger schemat:

- Ingen sal får dubbelbokas
- Ingen lärare får dubbelbokas
- Ingen årskurs får dubbelbokas
- Övningar måste ligga efter föreläsningar etc
- 8-10 i Kista och 10-12 på campus går inte
- Jämn fördelning mellan olika läsveckor
- Lärare X arbetar bara onsdagar–fredagar

Går det att konstruera ett schema som uppfyller alla villkoren?

Satisfierbarhet:
I t.ex. AI och programverifikation kan man ofta beskriva ett problem som en logisk formel i n variabler där en variabeltilldelning som gör formeln sann t.ex. kan svara mot att ett program är korrekt.
Ex. Betrakta formeln

$$(x \vee y \vee \neg w) \wedge (\neg x \vee z) \wedge$$
$$(\neg y \vee w) \wedge (x \vee \neg w \vee \neg z)$$

Finns det en satisfierande variabeltilldelning?
Formeln ovan satisfieras om x och z är sanna medan y och w är falska.

Handelsresandeproblemet:
En handelsresande vill besöka alla större städer i Sverige för att sedan återkomma till sin hemstad. För att spara tid och pengar vill han inte åka längre än han måste.
Travelling Salesman Problem (TSP):
Givet en graf $G = (V, E)$ med vikter på kanterna, finns det någon tur av längd högst L som passerar alla hörnen och återkommer till starthörnet?
Detta visar sig vara relaterat till problemet
Givet en graf, innehåller den någon hamiltoncykel?
(En hamiltoncykel är en cyklisk delgraf som passerar alla hörn exakt en gång.)

Kappsäcksproblemet:
En fjällvandrare ska packa en ryggsäck och orkar inte bära mer än W kg. Det finns många saker han vill ha med, och varje sak har känd vikt och användbarhet:

Sak	Vikt	Nytta
Tält	10	100
Sovsäck	7	80
Kudde	0.5	10
Extra tröja	1	25
Tandborste	0.01	5
Rakhyvel	0.1	2
etc		

(Det går inte att dela på någon sak.)
Går det att välja ut saker av total vikt högst W kg så att nyttan blir minst U?

Graffärgning:
I vissa sammanhang är det naturligt att betrakta komponenterna i ett system som hörnen i en graf där varje komponent är i ett visst tillstånd. En konfliktsituation kan då uppstå om angränsande komponenter är i samma tillstånd.
Som grafproblem:
Går det att färga hörnen i grafen G med högst K färger så att ingen kant förbinder två hörn med samma färg?
Också om G är planär förblir problemet svårt.

Alla problemen kan alla lösas med *uttömmande prövning*:
Prova alla möjliga kombinationer av de ingående variablerna och kolla om någon löser problemet.
Nackdel:
För alla dessa problem ger det väldigt
dåliga tidskomplexiteter — typiskt $\Omega(n!)$ eller $\Omega(2^n)$.
Gemensamma egenskaper:
Alla problem delar följande egenskap:
Om svaret är "ja" går det givet en lösning lätt att verifiera att den är korrekt.
D.v.s. någon som har löst problemet kan snabbt övertyga mig om det.
Obs att omvändningen inte behöver gälla: Om det t.ex. inte finns något tillåtet schema behöver det inte finnas något enkelt sätt att bevisa detta.

Klassen NP

Komplexitetsklassen **NP** innehåller de ja/nej-problem där det till varje ja-lösning finns ett "bevis" som kan kontrolleras effektivt (d.v.s. polynomisk tid).
Några problem i **NP**:

- Är vektorn $a[1..N]$ sorterad?
- Är den logiska formeln Φ satisfierbar?
- Är talet N sammansatt?

Annan definition av NP

Man kan se uttömmande prövning som ett stort träd där varje lösning svarar mot ett löv
Höjden är polynomisk men antalet löv exponentiellt.
Om man i varje nod valde rätt stig skulle man kunna lösa problemet på polynomisk tid (förutsatt att det finns någon lösning).
Detta kan användas som definition av **NP**.

Olika formella definitioner av NP

Första definitionen:
A *verifierar* instansen x av problemet L om det finns ett certifikat y så att

$$A(x, y) = \text{Ja} \quad \Leftrightarrow \quad x \in L$$

D.v.s A avgör språket
$L = \{x \in \{0,1\}^* : \exists y \in \{0,1\}^* : A(x, y) = \text{Ja}\}$
NP $= \{L : \exists A$ som verifierar L i polynomisk tid$\}$
P $\subseteq$ **NP** eftersom alla problem som kan lösas snabbt också kan verifieras snabbt.
Andra definitionen:
En ickedeterministisk algoritm är en algoritm som gör slumpmässiga val i olika steg. Utfallet av en körning är slumpmässigt. A accepterar ett språk L om:

$$x \in L \Rightarrow A(x) = \text{Ja} \quad \text{med sannolikhet} > 0$$
$$x \notin L \Rightarrow A(x) = \text{Nej} \quad \text{med sannolikhet } 1$$

NP $= \{L : \exists$polynomisk icke-deterministisk alg. som avgör $L\}$

P kontra NP

Om något **NP**-fullständigt problem kan lösas på polynomisk tid så kan alla lösas på polynomisk tid.
P $\subseteq$ **NP** eftersom alla problem som kan lösas på polynomisk tid också kan verifieras snabbt.
Är P $\neq$ NP?
Nästan alla forskare tror det, men sedan frågan formulerades 1971 har ytterst små framsteg mot ett bevis gjorts.
Den bästa undre gränsen för **NP**-fullständiga problem är $\Omega(n)$ d.v.s. den triviala gränsen.

Olika typer av problem:

1. Beslutsproblem: $A(x) =$ Ja.
2. Optimeringsproblem: $A(x) = m$ Vanligen max/min.
3. Konstruktionsproblem: $A(x) =$ En struktur. Vanligen "lösningen" till problemet.

Några reduktioner:
Lösningen till beslutsproblem ger ofta lösningen till motsvarande optimerings- och konstruktionsproblem.
Ex:
KLICK

Indata: Graf G och ett tal K.
Problem: Finns det en **klick** av storlek K i G?
(Klick = Komplett delgraf.)

KLICK är ett svårt (**NP**-fullständigt) problem.

Motsvarande optimeringsproblem:
MAX-KLICK

Indata: Graf G. Problem: Vad är storleken på en maximal klick i G?

Motsvarande konstruktionsproblem:
KONSTRUERA-MAX-KLICK

Indata: Graf G. Problem: Hitta en klick av maximal storlek i G.

Antag att KLICK har en lösningsalgoritm $A(G, k)$ så att $A(G, k) =$ Ja om och endast om G har en klick av storlek k.
MAX-KLICK har lösningsalgoritmen $B(G)$:

```
(1)   for k ← to n
(2)       if A(G, k) = Ja
(3)           m ← k
(4)   return m
```

KONSTRUERA-MAX-KLICK har lösningsalgoritm $C(G)$:

```
(1)   m ← B(G)
(2)   S ← V
(3)   foreach v ∈ V
(4)       if B(G(S − {v})) = m
(5)           S ← S − {v}
(6)       return S
```

Polynomiska reduktioner (Karp-reduktioner)

Problemet Q kan *reduceras* till Q' om varje instans x av Q kan omvandlas till en instans x' av Q' så att $x \in Q$ om och endast om $x' \in Q'$.
Den här typen av reduktion kallas för
karpreduktion.
Om reduktionen tar polynomisk tid skriver vi

$$Q \leq_P Q'$$

eftersom Q inte kan vara svårare att lösa än Q'.
Viktiga konsekvenser av detta:

- Om $Q \leq_P Q'$ och $Q' \in \mathbf{P}$ så $Q \in \mathbf{P}$.
- Om $Q \leq_P Q'$ och $Q \notin \mathbf{P}$ så $Q' \notin \mathbf{P}$.

NP-fullständiga problem

Problemet Q är **NP**-fullständigt om

1. $Q \in \mathbf{NP}$
2. Q är **NP**-svårt: $Q' \leq_P Q$ för alla $Q' \in \mathbf{NP}$

Att ett problem är **NP**-fullständigt betyder i praktiken att det inte finns någon effektiv algoritm för det; nästan alla tror att $\mathbf{P} \neq \mathbf{NP}$.

Turingreduktioner

En turingreduktion $Q \leq_T Q'$ är en algoritm som löser Q med hjälp av anrop till en algoritm för Q':

```
Q(x)
(1)     ⋮
(2)     Q'(z)
(3)     ⋮
(4)     return svar
```

Turingreduktionen är liberalare än karpreduktionen; Q' behöver t.ex. inte vara ett beslutsproblem.
Om Q är **NP**-fullständigt och $Q \leq_T Q'$ säger vi ändå att Q' är **NP**-svårt.

Problem: För att visa att A är **NP**-fullständigt måste man visa att $X \leq_P A$ för alla $X \in \mathbf{NP}$. Hur är det möjligt?

Första NP-fullständiga problemet

SAT är följande problem:
Givet en logisk formel Φ, finns det någon variabeltilldelning som gör den sann?
Sats (Cook 1971). *SAT är **NP**-fullständigt.*
(I samma artikel som klassen **NP** definierades för första gången bevisades denna sats.)

Bevisskiss: Varje **NP**-problem motsvarar en icke-deterministisk algoritm A. Problemet är att avgöra om, givet indata x, det finns en beräkning som ger $A(x) = \text{Ja}$. Cook visade att man kan konstruera en logisk formel F så att $A(x)$ har accepterande beräkning $\Leftrightarrow$ F är satisfierbar.

För att visa att A är **NP**-fullständigt räcker det att visa att $SAT \leq_P A$.
Varför: Om $X \in \mathbf{NP}$ gäller $X \leq SAT$. Om $SAT \leq A$ också gäller får vi $X \leq A$!
Praktiskt tillvägagångssätt: Bilda riktad graf där $A \to B$ betyder $A \leq B$. $SAT \to A \to B \to C \to ...$ betyder då att $A, B, C, ...$ är **NP**-fullständiga.

För att visa att A är **NP**-fullständigt räcker det att hitta ett **NP**-fullständigt problem B så att $B \leq A$.

Om NP-fullständighetsbevis

Alla **NP**-fullständiga problem kan reduceras till varandra, men om man ska visa att ett problem är **NP**-fullständigt kan det ändå löna sig att utgå från rätt problem.
I praktiken finns det ofta enkla reduktioner från något av 5–10 kanoniska **NP**-fullständiga problem.
Det enskilda vanligaste problemet att utgå från är 3-SAT där vi har logiska formler på konjunktiv normalform med exakt 3 literaler i varje klausul.

3-SAT är NP-fullständigt

Att 3-SAT $\in$ **NP** är klart.
Vi ska reducera SAT till 3-SAT:
Givet en SAT-formel $\Phi = c_1 \wedge \cdots \wedge c_k$ bildar vi en ekvivalent 3-SAT-formel Φ_3 genom att ersätta varje klausul i Φ med en eller flera 3-SAT-klausuler.
Antag att c_i innehåller j literaler $l_1, \ldots l_j$. Klausuler som läggs till Φ_3:

$j = 3$ $\quad l_1 \vee l_2 \vee l_3$

$j = 2$ $\quad (l_1 \vee l_2 \vee y_i) \wedge (l_1 \vee l_2 \vee \neg y_i)$

$j = 1$ $\quad (l_1 \vee y_i \vee z_i) \wedge (l_1 \vee y_i \vee \neg z_i) \wedge$
$\qquad (l_1 \vee \neg y_i \vee z_i) \wedge (l_1 \vee \neg y_i \vee \neg z_i)$

$j > 3$ $\quad (l_1 \vee l_2 \vee y_i^1) \wedge (\neg y_i^1 \vee l_3 \vee y_i^2) \wedge$
$\qquad (\neg y_i^2 \vee l_4 \vee y_i^3) \wedge \cdots \wedge (\neg y_i^{j-3} \vee l_{j-1} \vee l_j)$

Φ_3 är per konstruktion satisfierbar precis då Φ är satisfierbar.

Några NP-fullständiga grafproblem

Hamiltoncykel

Har grafen G en hamiltoncykel?
(D.v.s. en sluten stig som passerar alla hörn exakt en gång.)

Klick
Innehåller grafen G en klick av storlek K?
(En klick är en komplett delgraf.)

Hörntäckning
Givet en graf G och ett tal K, går det att välja ut K hörn i G så att alla kanter innehåller minst ett av de K hörnen?

Graffärgning
Givet en graf G och ett heltal K, går det att färga hörnen i G med K färger så att ingen kant blir monokromatisk?

Andra NP-fullständiga problem

Exakt täckning
Givet en uppsättning delmängder av en basmängd M. Går det att välja ut en mängd av delmängder så att varje element i M ligger i exakt en mängd?

Delsumma
Givet en mängd P av positiva tal och ett tal K. Finns det någon delmängd av talen i P vars summa är K?

Heltalsprogrammering
Givet en $m \times n$-matris A, en m-vektor b, en n-vektor c och ett tal K. Finns det en n-vektor x med heltal så att $Ax \leq b$ och $c \cdot x \geq K$?

Om man i sista exemplet släpper kravet att komponenterna i x är heltal blir problemet (linjärprogrammering) polynomiskt lösbart.

Ytterligare NP-fullständiga problem

Lådpackning
Givet en mängd vikter och k lådor som var och en klarar högst vikt B. Går det att lagra vikterna i lådorna?

Kappsäck
Givet en mängd S med element som har vikter w och värde v definierade. Går det att hitta en delmängd S' så att summan av vikterna $\leq W$ och summan av värdena $\geq B$?

TSP
Handelsresandeproblemet.

Reduktioner för flera av dessa problem finns i boken.
Två problem: OBEROENDE MÄNGD, HÖRNTÄCKNING
Oberoende mängd: Mängd noder utan kanter mellan dem.
Hörntäckning: Urval av noder som täcker in varje kant.
OBEROENDE MÄNGD
Indata: Graf G, tal K. Problem: Finns det oberoende mängd av storlek K?

HÖRNTÄCKNING
Indata: Graf G, tal K. Problem: Finns hörntäckning av storlek K?

OBEROENDE MÄNGD är NP-fullständigt

Vi reducerar från 3-CNF-SAT. Givet $\phi = C_1 \wedge C_2 \wedge ... \wedge C_m$ bildar vi en graf G med m st. trianglar. Om t.ex. $C_1 = l_1 \vee l_2 \vee l_3$ där $l_i = x_i$ eller $l_i = \bar{x}_i$ så sätter vi l_1, l_2, l_3 till noder i första triangeln. Gör på samma sätt för de övriga klausulerna och motsvarande trianglar. När trianglarna är klara drar vi kanter mellan alla par av noder märkta $x_i, \bar{x}_i$ för alla i. Detta ger grafen G som är en instans av OBEROENDE MÄNGD. Man kan se att (ϕ är satisfierbar) $\Leftrightarrow$ (G har oberoende mängd av storlek m).

Observation: Om $A \subseteq V$ är en oberoende mängd så är $V - A$ en hörntäckning och omvänt.

Givet instans $x = G, K$ till OBEROENDE MÄNGD skapas $R(x) = G, |V| - K$ till HÖRNTÄCKNING. Detta ger en reduktion OBEROENDE MÄNGD $\leq_P$ HÖRNTÄCKNING

Två andra problem: HANDELSRESANDEPROBLEMET(TSP), HAMILTON-KRETS

TSP

Indata: Viktad komplett graf G och ett tal K. Problem: Finns det Hamiltonkrets med vikt $\leq K$?
(Hamiltonkrets = Krets som besöker varje nod exakt en gång.)

HAMILTONKRETS

Indata: Graf G. Problem: Finns det en Hamiltonkrets?

Givet indata $x = G$ till HK skapas komplett graf G' med $w(e) = 1$ om $e \in G$ och $w(e) = 0$ om $e \notin G$. Sätt sedan $K = |V|$. Det ger en korrekt reduktion.

Delgrafsisomorfi är NP-fullständigt

Givet två oriktade grafer G_1 och G_2, är G_1 en delgraf till G_2?
Problemet tillhör uppenbarligen **NP**.
Bevis för att problemet är **NP**-svårt:
Reducera från Hamilton Cycle.
En graf $G = (V, E)$ innehåller nämligen en hamiltoncykel om och endast om den innehåller en delgraf som är isomorf med cykeln $C_{|V|}$ som har lika många noder som G.

DELSUMMA är NP-fullständigt

Vi reducerar från 3-CNF-SAT igen. Tag t.ex. $\phi = (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee \bar{x}_3)$.
Vi skapar tal $y_1, .., M$ enligt följande mönster:

	x_1	x_2	x_3	C_1	c_2
y_1	1	0	0	0	1
z_1	1	0	0	1	0
y_2	0	1	0	0	1
z_2	0	1	0	1	0
y_3	0	0	1	1	0
z_3	0	0	1	0	1
g_1	0	0	0	1	0
h_1	0	0	0	1	0
g_2	0	0	0	0	1
h_2	0	0	0	0	1
M	1	1	1	3	3

Vi får följande instans av DELSUMMA: Kan M fås som delsumma av talen $y_1, ..., h_2$?
Om svaret är JA så vet vi att exakt 1 av talen y_1, z_1 kan ingå i delsumman. Om y_1 ingår så sätter vi x_1 till sann. Om z_1 ingår så sätter vi x_1 till falsk. Gör samma sak för x_2 och x_3. Eftersom C_1 har kolumnsumman 3 så måste det finnas något y_i eller z_i i delsumman så att x_i eller $\bar{x}_i$ ingår i C_1. Så C_1 blir satisfierad. Samma gäller för C_2.
Omvänt: Om ϕ går att satisfiera väljer vi y_1 om x_1 är sann i satisfieringen och z_1 om x_1 är falsk i satisfieringen. Gör samma för x_2 och x_3. Välj sedan g och h tal så att summan i c_1 och c_2 indexen blir 3. Vi får då summan M.

Partition är NP-fullständigt

Givet en mängd S av positiva tal.
Kan S delas upp i två disjunkta delar S_1 och S_2 så att $\sum_{x \in S_1} x = \sum_{x \in S_2} x$?
Problemet ligger i **NP** ty givet en disjunkt uppdelning går det snabbt att kolla om den uppfyller villkoren.
Vi reducerar från Subset sum:
En instans där är tal $p_1, p_2, \ldots, p_n$ och K; ur instansen skapar vi partitionsinstansen

$$p_1, p_2, \ldots p_n, P - 2K$$

(där $P = \sum p_i$) om $K \leq P/2$ och

$$p_1, p_2, \ldots p_n, 2K - P$$

annars. En jämn partitionering av denna instans finns precis då Subset sum-instansen är lösbar.

0/1-programmering är NP-fullständigt

Givet en $m \times n$-matris A och en m-vektor b.
Finns det en n-vektor x med heltal $\in \{0, 1\}$ så att $Ax \leq b$?
Problemet tillhör **NP** ty givet en lösning x tar det $O(n^2)$ tid att kolla att $Ax \leq b$.
För att bevisa att problemet är **NP**-svårt
reducerar vi 3-CNF-SAT till det:

En instans av 3-CNF-SAT är en formel Φ över n variabler. Till x_i i Φ låter vi svara $y_i \in \{0, 1\}$ där 1 är sant och 0 falskt.
För varje klausul $c_j = l_1 \vee l_2 \vee l_3$ inför vi en olikhet

$$T(l_1) + T(l_2) + T(l_3) \geq 1$$

där $T(x_i) = y_i$ och $T(\neg x_i) = (1 - y_i)$.

0/1-programmering, forts.

Exempel:
Klausulen $x_1 \vee \neg x_2 \vee x_3$ ger olikheten
$y_1 + (1 - y_2) + y_3 \geq 1$.
Tolkning: Vänsterledet är sant (d.v.s. minst 1) om minst en av literalerna är sann (d.v.s. har värdet 1).
Formeln Φ omvandlas alltså till en uppsättning linjära olikheter som kan uppfyllas om och endast om Φ är satisfierbar.
$\Rightarrow$ 0/1-programmering är **NP**-fullständigt.
(Ofta vill man i tillämpningar maximera $c \cdot x$ givet att $Ax \leq b$ där x är en $\{0, 1\}$-vektor. Det är också **NP**-svårt.)