P, NP and NPC

Lukas Jørgensen, 201206057

June 7, 2015

1 Disposition

- Hvilke problemer - Beslutningsproblemer, encoding
- Hvad er et sprog
- Hvad er et stand-in problem
- P og NP
- Reduktioner
- NPH og NPC

2 Noter

2.1 Problemer

Problemer kan være hvad som helst, alt fra 2+2 til "hvordan farver jeg denne graf med 3 farver".

I dette kursus har vi begrænset os til at se på beslutningsproblemer, det vil sige problemer hvor outputtet altid er "ja" eller "nej" instanser. Dette kan også beskrives ved binære strenge over $\{0,1\}$. Grunden til dette er, at vi da kan opfatte problemerne som "sprog" hvor der gælder at:

$$x \in L \text{ iff } x \text{ er en "ja" instans til problemet.} \tag{1}$$

Udover at begrænse typen af problemer, har vi også valgt at begrænse os til at se på inputs der er bitstrenge over $\{0,1\}^*$. Selvom vi har denne begrænsning, er vores inputs stadig meget generelle, da dette allerede gøres i computere, hvor man f.eks. kan encode tekst v.h.a. ASCII. Ethvert naturligt tal kan da encodes i binære bitstrenge, sådan som det allerede gøres i computere, de kan dog ikke repræsentere reelle tal.

2.2 Stand-in problemer

For at vise at restriktionen med at vi kun kigger på beslutningsproblemer, stadig er ret generel, kigger vi på såkaldte stand-in problemer. Disse stand-in problemer er beslutningsproblemer der kan bruges til at løsede egentlige problemer.

Tjek dette

2.2.1 Problemer med bitstrenge som output

Hvis vi har nogle problemer hvor outputtet ikke består af "ja" eller "nej", men i stedet består af en arbitrær bitstreng, kan vi konstruerer et stand-in sprog for dette problem.

Et stand-in sprog for en given funktion $f : \{0,1\}^* \to \{0,1\}^*$ kan skrives således:

$$L_f = \{\langle x, b(j), y\rangle | f(x)_j = y\} \tag{2}$$

Således kan man udregne outputtet en bit ad gangen.

2.2.2 Optimeringsproblemer

Hvis vi har givet et optimerings problem:

> **OPT:** "Givet en inputstreng der definerer et sæt mulige løsninger F og en objective funktion f, find $x \in F$ der maksimerer $f(x)$".

a Dette er dog ikke et beslutningsproblem, og vi (tror) ikke at vi kan tjekke det i polynomiel tid. I stedet kan vi udtrykke dette problem som et beslutningsproblem vha. følgende stand-in sprog:

> $\mathbf{L_{OPT}}$: "Givet en inputstreng der definerer F, f og en target værdi $v \in Q$, bestem hvorvidt der er en løsning $x \in F$ således $f(x) \geq v$".

Dette er dog ikke en perfekt "stand-in" da L_{OPT} sagtens kan have en effektiv løsning, uden at OPT har det. Omvendt, kan så ved vi at hvis L_{OPT} er svær at løse, så ved vi også at OPT er det. (Hvis vi kan finde den maksimale værdi i polynomiel tid, kan vi også bestemme hvorvidt der er en løsning der større end v i polynomiel tid.)

Derfor kan vi ikke bruge L_{OPT} til at vise at et problem har en effektiv løsning, men vi kan bruge det til at argumentere for at et givent problem er svært.

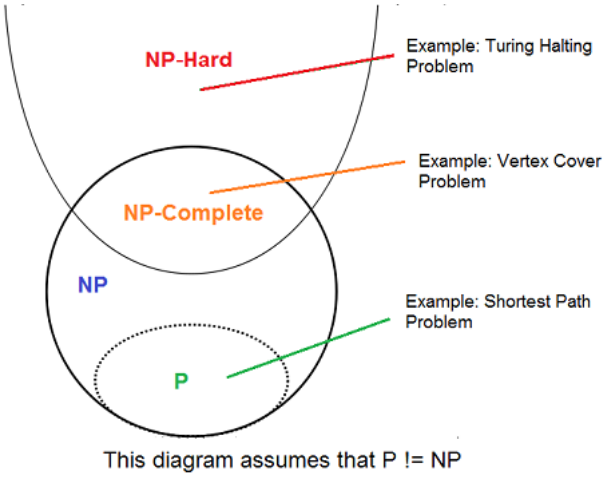

Figure 1: En figur der visualiserer P, NP, NPC og NP-Hård

2.3 P og NP

2.3.1 P

Kompleksitetsklassen P er klassen af beslutningsproblemer, der kan blive bestemt af en deterministisk Turing Maskine, hvor antallet af "steps" maskinen udfører for et eller andet input x er $\rho(x)$ for et arbitrært polynomie ρ. Dvs. det er polynomielt i inputtet.

Dette kan også beskrives som ethvert problem med worst-case kørselstid $O(n^k)$ for et givent k.

Formelt defineres P som:

$$P = \{L \subseteq \{0,1\}^* | \exists \text{ TM } M_L \text{ der afgører L i polynomiel tid.}\} \tag{3}$$

Det skal dog siges, at i praksis kan man godt komme ud for at et teoretisk "svært" problem, godt kan være lettere end et teoretisk "let" problem. Når man ser på problemerne teoretisk, tager man ikke højde for konstanter eller forventet kørselstid som kan betyde at et problem kan være lettere at løse i de praktiske tilfælde frem for det rent teoretiske.

2.3.2 NP

Kompleksitetsklassen NP klassen af beslutningsproblemer for hvilket "ja" instanserne kan verificeres i polynomiel tid. Altså, tager løsningen til disse problemer eksponentiel tid, men man kan verificere en given løsning til problemet i polynomiel tid.

Formelt defineres NP således:

$$\forall x : x \in L \Leftrightarrow \{\exists y \in \{0,1\}^* : |y| \leq \rho(|x|) \wedge \langle x, y\rangle \in L'\} \tag{4}$$

Denne definition kan forstås således: vi har et instans af et problem $x \in L$, vi har da sættet af binære strenge med længde $\leq \rho(|x)$ der repræsenterer potentielle løsninger til problemet der er associeret med instansen x. Med denne fortolkning, betyder $\langle x, y\rangle \in L'$ at den potentielle løsning y er en korrekt løsning til problemet.

2.4 Reduktioner

2.4.1 Polynomial time computable maps

Et polynomial time computable map, er en funktion $f : \{0,1\}^* \to \{0,1\}^*$, hvor der gælder at:

$$\forall x : |f(x)| \leq \rho(|x|) \tag{5}$$

$$L_f \in P \tag{6}$$

For et arbitrært polynomium ρ.

D.v.s at længden af funktionens output er polynomielt på inputtet og sproget for funktionen f skal have en polynomiel kørselstid.

Dette kan også ses som en måde at oversætte fra en repræsentation eller model, til en anden i polynomiel tid, vha. en funktion f.

2.4.2 Polynomielt ækvivalente repræsentationer

To repræsentationer π_1 og π_2 er polynomielt ækvivalente hvis der findes polynomial time computable maps r_1 og r_2 der oversætter mellem de to repræsentationer.

D.v.s. for alle $x \in S$ hvor S er instanserne for et beslutningsproblem $f : S \to \{yes, no\}$ (f.eks. sættet af all endelige directed graphs).

$$\pi_1(x) = r_1(\pi_2(x)) \wedge \pi_2(x) = r_2(\pi_1(x)) \tag{7}$$

2.4.3 Reduktioner

Vi siger at L_1 *reducerer til* L_2 ($L_1 \leq L_2$) hvis der er en polynomial time computable funktion r således at for alle $x \in \{0,1\}^*$ så har vi at $x \in L_1 iff r(x) \in L_2$.

2.5 NP-Hard og NPC

2.5.1 NP-hard

Et sprog er `NP-hard` hvis der gælder:

$$\forall L' \in \text{ NP } : L' \leq L \tag{8}$$

Altså gælder der, at ethvert sprog L' i NP kan reduceres til L. Altså er algoritmen til at løse `NP-hard` problemet så generel, at den kan bruges til at løse ethvert andet problem i NP. Derfor siger man også at det intuitivt set er mindre sandsynligt at et `NP-hard` problem er i P, da det vil medføre at $P = NP$.

NP-hard problemer er ikke nødvendigvis i P, og hvis de er i P kaldes de typisk `NP-Complete` i stedet.

2.5.2 NPC

`NP-Complete` (NPC) er den klasse af problemer der både er `NP-hard` og ligger i NP. Vi kan definerer dette formelt således:

$$\text{NPC} = \{L \in \{0,1\}^* \| L \in NP \wedge (\forall L' \in \text{NP} : L' \leq L)\} \tag{9}$$

NPC-problemer er interessante, fordi vi kan ved hjælp af disse problemer, bevise at et givent problem ikke ligger i P medmindre $P = NP$ og derved undgå at bruge tid på at forsøge at finde en algoritme i P til et problem der (formentlig) ikke har en effektiv løsning.

3 Beviser

3.1 Proposition 3 - transitivitet af reduktioner

Hvis $L_1 \leq L_2$ og $L_2 \leq L_3$ så gælder der at $L_1 \leq L_2$ (transitivitet af reduktioner).

Da $L_1 \leq L_2$ og $L_2 \leq L_3$ så ved vi at vi har to polynomial time computable maps r_1 og r_2, hvor der gælder:

- For ethvert arbitrært x er $x \in L_1$ iff $r_1(x) \in L_2$
- For ethvert arbitrært y er $y \in L_2$iff $r_2(x) \in L_3$

Heraf kan vi så se at for alle x, har vi at $x \in L_1$ iff $r_2(r_1(x)) \in L_3$. Da vi blot bruger to polynomial time computable maps efter hinanden, vil det tage $\rho_1(|x|) + \rho_2(|r_1(x))$ tid, hvilket stadig er polynomielt så derfor har vi at $L_1 \leq L_3$.

3.2 Proposition 5 - NP-hard $\in P \implies P = NP$

Lad L være et NP-hard sprog. Hvis $P \neq NP$ så gælder der at $L \notin P$

Siden L er `NP-Hard`, gælder der at $\forall L' \in NP : L' \leq L$. Hvis vi derimod antager at $L \in P$ så gælder der at alle $L' \in NP$ er i P og derfor gælder det at $P = NP$.

3.3 Proposition 6

Lad $L \in NPC$ så er $P = NP$ iff $L \in P$

Fra proposition 5 ved vi at $P \neq NP$ hvis $L \notin P$, men vi ved ikke direkte herfra at det er "hvis og kun hvis".

Vi kan derimod udlede dette fra at $L \in NP$. Da vi fra proposition 5 ved at $L \notin P$ og vi ved at sproget er i NP, så må det betyde at $P \neq NP$ iff $L \notin P$ og derved at $P = NP$ iff $L \in P$.