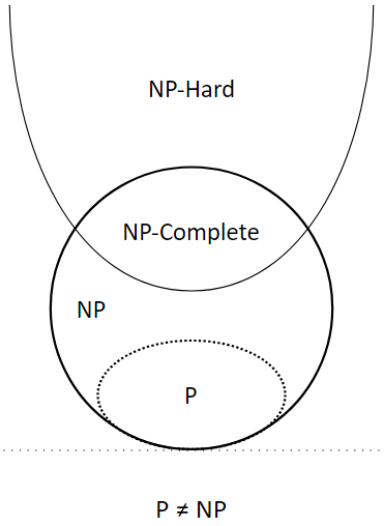
NP complete problems

***Disposition*:**

- Hvad er P og NP
- Languages and TMs
    - Church Turing
    - Formal Languages
    - Problems as languages
    - Representations (maps)
- Reduction
- NP-hard and NP-complete
- SAT til 3SAT
- 3SAT til INDEPENDENT SET
- NAESAT til 3COLORING
- Måske 3SAT til HAMILTON PATH

Hvad er P og NP

Vi vil gerne kunne klassificere hvilke problemer der kan og ikke kan løses af "efficient algorithms", dvs. algoritmer der i Worst Case bruger polynomiel tid, samt nogle andre egenskaber. Det er altså klasser af problemer. Ligesom i BerLog vil vi benytte os meget af reductioner for at finde frem til egenskaber ved problemer, og for at finde ud af hvilke klasser de tilhører. Det kommer jeg ind på lidt senere.



Klasserne P og NP vil vi bruge til at definere problemer som kan henholdsvis løses effektivt (polynomielt) og med exhaustive search (exponentielt).

At noget ikke har en efficient algorithm betyder selvfølgelig ikke at vi ikke kan løse problemet, vi har skam flere algoritmer til dette som f.eks. branch and bound, men det siger noget om hvad vi kan håbe på i worst case.

Languages and TMs

Dette leder os ind på sprog og Turing Maskiner. Sprog er vores måde at repræsentere problemer på, hvor et sprog er en eller anden delmængde af {0,1}*. Vi arbejder gerne med decision problems, da vi også kan omformulere optimeringsproblemer som decision problemer ved at tilføje Targets og Budgets, så vi beskriver JA-instanser af et problem som værende alle de bitstrenge i dets givne sprog L. Dvs. givet en bitstreng kan vi afgøre om det er en instans af 3COLORING som godt kan farves, iff den ligger i sproget.

Vi vil også gerne have en model for udregning, og dertil har vi vores troværdige Turing Maskine. Den består af et bånd der strækker sig uendeligt til højre. Båndet består af celler som alle indeholder enten et symbol fra alfabetet eller blank. TM'en har states og skifter states ud fra dets Transition Diagram delta når nålehovedet peger på en celle. Jeg skal nok spare jer for flere detaljer nu, så det vigtige er bare at vi påstår at denne konstruktion beskriver "beregnelighed" godt. Vi bruger dem nu til decision problems, hvor vi lægger input strengen på båndet, og vi siger så at en Turing Maskine bestemmer et sprog, hvis alle elementerne i sproget får Turing Maskinen til at gå i Accept.

Vi definere nu P klassen, til at være klassen af decision problemer (sprog) som kan bestemmes af en Turing Maskine, hvor den på alle input x terminere efter max p(|x|) skridt.

***Church-Turing Thesis**:*

> Any decision problem that can be solved by some mechanical procedure, can be solved by a Turing machine.

***Polynomial Church-Turing Thesis**:*

> A decision problem can be solved in polynomial time by using a reasonable sequential model of computation if and only if it can be solved in polynomial time by a Turing machine.

Ifølge ***Polynomial Church-Turing Thesis*** så er P robust over for vores MOC.

Vi har også det vi kalder Polynomial Time Computable Maps, som er funktioner der tager bitstrenge og retunerer bitstrenge. Det er funktioner som omdanner en repræsentation af et problem til en anden. Vi siger at to repræsentationer er Polynomisk Ækvivalente hvis der findes et map der kører i polynomisk tid imellem dem.

Dette giver os, at hvis vi har to ækvivalente repræsentationer, så hvis det ene sprog er i P, så er det andet også. Dette lægger også grunden for vores reductioner, og den måde vi kan overfører egenskaber på.

Reduction

A reduction r of L_1 to L_2 is a polynomial time computable map so that
$\forall\, x$: $x \in L_1$ iff $r(x) \in L_2$
We write $L_1 \leq L_2$ if L_1 reduces to L_2.
Med andre ord, vi tager en instans af L1 og udtrykker det med L2.
Intuition: Efficient software for L_2 can also be used to efficiently solve L_1.
Downward closure of P:

$$L_1 \leq L_2 \wedge L_2 \in \boldsymbol{P} \Rightarrow L_1 \in \boldsymbol{P}$$

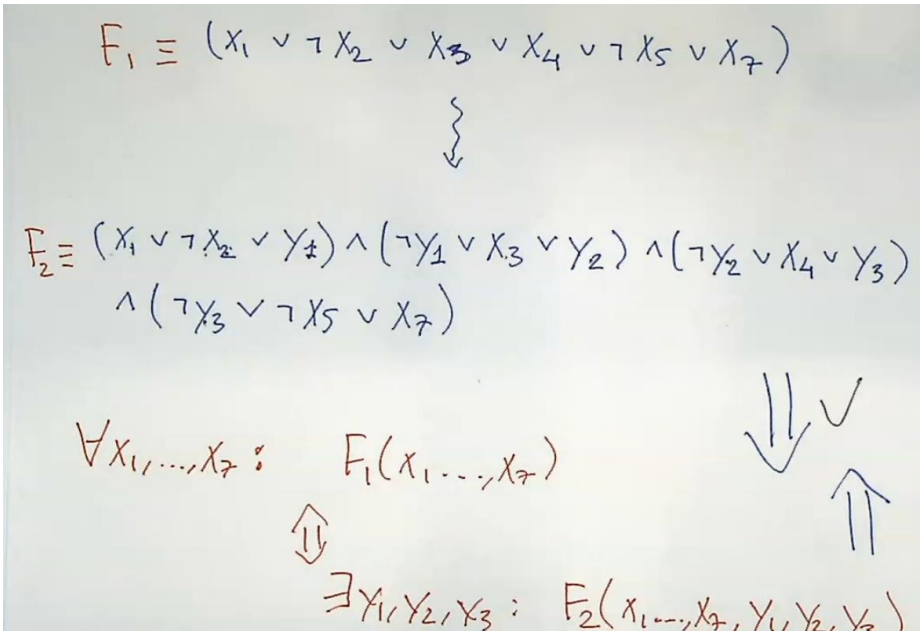
Vi siger at et sprog L er NP-hard iff for alle sprog L' i NP at du kan reducere **til** L.
Vi siger at et sprog L er NP-complete iff L er i NP og er NP-hard.
For at bevise NP-hardness omkring sprog, skal vi bare bruge et bevis for at 1 sprog er NP-hard, og resten er reductioner.

SAT til 3SAT

Vi kan ledt udregne at 3SAT er NP, da vi bare kan encode en instans af problemet som en bitstreng af alle variablerne, så vi kan tjekke det polynomielt. Vi ved også at SAT er NP-complete, ud fra Cook's Theorem. Derfor vil vi gerne reducere fra SAT til 3SAT, for at vise at 3SAT også er NP-hard. Som med enhver anden reduction, vil vi I dette tilfælde gerne lave en algoritme, altså en eller anden mapping, fra L1 til L2. Dvs. vi vil tage en instans fra SAT og omdanne til en instans af 3SAT, hvor det selvfølgelig skal overholde at hvis instansen er en JA-instans af SAT iff det er en JA-instans af 3SAT. Derfor skal vi bevise at vores reduction gælde begge veje bagefter.

Start med at lave clauses af mere end 3 variabler om til clauses af 3 variabler ved at tilføje en auxiliary variabel y_i i slutningen og NOTy_i i starten af den næste.

Nu gælder det at vores 3SAT 3CNF er sand iff der findes en tildeling af variablerne og de auxiliary variabler således at hvert clause bliver sandt.



Man "leder" sådan set bare clauses igennem indtil man finder der hvor den variabel der var sand i SAT CNF'en findes. Vi kunne også bare have set på CircuitSAT til SAT, fordi vi faktisk ender med en 3SAT instans.

Vi har nu at 3SAT er NP-complete (og hard).

3SAT til INDEPENDENT SET

INDEPENDENT SET problemet består i at vi går givet en graf G med nogle vertices og edges, og skal afgøre om der findes et set af vertices på størrelse mindst K, således at ingen kanter er forbundet med mere end 1 vertex fra settet.

INDEPENDENT SET

- Given: Undirected graph $G = (V, E)$, target K.
- Question: Does there exist an independent set I in G with $|I| \geq K$?

Vi starter med at anvende en gadget, nemlig en trekant. Vi vil gerne konstruere en instans af INDEPENDENT SET ud fra vores instans af 3SAT, hvor den skal have samme ”løsning”. Siden vi arbejder med 3SAT, laver vi en trekant for hver clause, hvor at vertices i trekanterne beskriver den tilsvarende literal fra vores clauses. Vi tegner nu kanter mellem alle literals som er modsat hindanden, dvs. x1 og NOTx1. De vertices vi vælger til at være en del af I svarer til at blive sat til ”true” i vores clause. Vi sætter vores target K til at være antal clauses M, for at sikre os at alle clauses har 1 true literal. Siden K = M, så skal vores set I bestå af 1 literal fra alle clauses, ellers ville det ikke være independent og/eller ikke være stort nok. Vores kanter mellem trekanterne sikre os consistency.

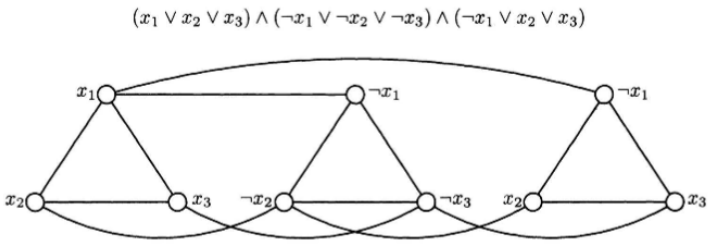


Figure 9-2. Reduction to INDEPENDENT SET.

Hvis vi får givet en JA-instans af 3SAT, konstruerer vi bare grafen som beskrevet, og sætter variablerne tilsvarende.

Den anden vej, givet et korrekt Independent Set, så sætter vi bare de literals der er i I til at være true.

NAESAT til 3COLORING

NAESAT er et special case af 3SAT, hvor det gælder at alle 3 variabler i dets clauses ikke må være ens. Dvs. de må ikke alle være false eller alle være true. Det gælder også at JA-instanser er lukket under komplementet, da komplementet til en JA-instans bare flipper alle variablerne og giver os noget andet der også opfylder Not All Equal, og at der må være mindst 1 sandt element i hver clause. Man kan lave en reduction fra CircuitSAT til NAESAT for at vise NP-hardness, men jeg skipper den del da man bare laver lidt om på hvordan CircuitSAT til SAT ser ud, da den ikke beskyttede imod at alle variabler i clauses med mindre end 3 variabler var true. Det er også klart nok at den ikke må være i P, da ellers ville 3SAT også være. Denne gang laver jeg en reduction fra NAESAT til 3COLORING, for at vise at 3COLORING er NP-hard. 3COLORING er problemet hvor man givet en graf med nogle kanter og vertices, skal finde ud af om der kan tildeles farver til vertices sådan at ingen kanter har samme farver på begge ender.

3-COLORING

- Given: Undirected graph $G = (V, E)$.
- Question: Does there exist a valid 3-coloring of G?
  (i.e. a function $c: V \to \{0,1,2\}$ such that $c(u) \neq c(v)$ for all $uv \in E$)

Vi har et eller antal clauses i vores NAESAT instans, som har et eller antal variabler. Vi vil gerne omdanne det til en instans af 3COLORING. Vi starter derfor med at definere farverne 0, 1 og 2 som værende ”true”, ”false” og ”?”. Dvs. 2 kan være begge.

Vi konstruere vores graf ved at starte med en vertex som har farven 2. Vi sammensætter nu en trekant for være variable i vores NAESAT instans, hvor 2 verticen er toppen for dem alle. Vi laver også en trekant for hver clause, hvis vertices repræsentere de variabler som bruges i det clause, og vi laver en kant fra de vertices ud til de variabler som bruges. Det betyder, at hvis vi kan finde en måde at tildele farverne på så det overholder 3COLORING, så må der være en måde at tildele variablerne værdier således at ”true” eller ”false” ikke fremkommer mere end 1 gang i vores clause trekanter. En JA-instans af 3COLORING konstrueret på denne måde vil derfor altid give en JA-instans af NAESAT.

For at bevise det den anden vej, er vi givet en JA-instans af NAESAT. Vi starter som konstruktionen gjorde før, men i clause trekanten vælger vi bare 2 af variablerne som havde forskellige værdier, og farver dem efter dem. Og så får den sidste bare farven "?".

3SAT er en af de meget brugbare problemer til at reducere med for at vise at andre problemer er NP-hard. NAESAT er smart til denne reduction fordi 3COLORING har et element af "ikke alle må være ens". Man har derfor nogle gange lyst til at bruge andre problemer end 3SAT, hvis man ved noget specielt om problemet. Det sker også i tilfældet af HAMILTON PATH til TSP.

Måske 3SAT til HAMILTON PATH

Det starter egentlig med at vi reducere fra den brugbare 3SAT til HAMILTON PATH. Jeg har ikke tænkt mig at gå helt i dybden med den, da den er lidt lang.

Let $G = (V, E)$ be an undirected graph. A Hamiltonian path in G is a path in G that visits each node exactly once.

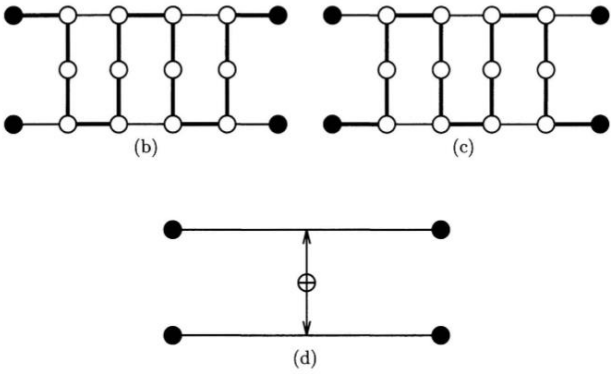
HAMILTON PATH

- Given: Undirected graph $G = (V, E)$.
- Question: Does there exist a Hamiltonian path in G?

Vi vil som sagt gerne omdanne noget fra 3SAT til noget fra HAMILTON PATH, hvor HAMILTON PATH problemet går ud på at lave en vej der besøger alle vertices præcis 1 gang. Konstruktionen består af flere lag, så vi starter småt. Vi vil gerne have en eller anden måde at tildele vores variabler sandhedsværdier. Dette gøres med en "choice gadget" kaldet vi det.

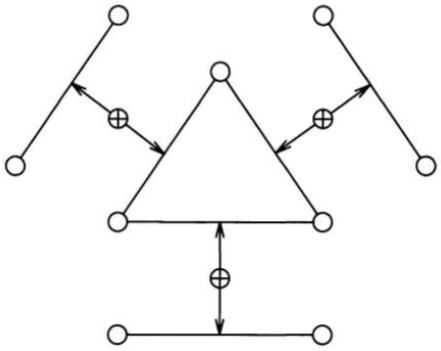
Denne gadget har 2 stier, som henholdsvist bestemmer om variablen er "true" eller "false. Vi laver derfor en lang kæde af disse for at tildele alle variablerne "true" eller "false.

Det næste skridt er at vi gerne vil sikre os at disse tildelinger bliver "udført" og overholdt i resten af grafen, så vi laver det vi kalder en "consistency gadget".

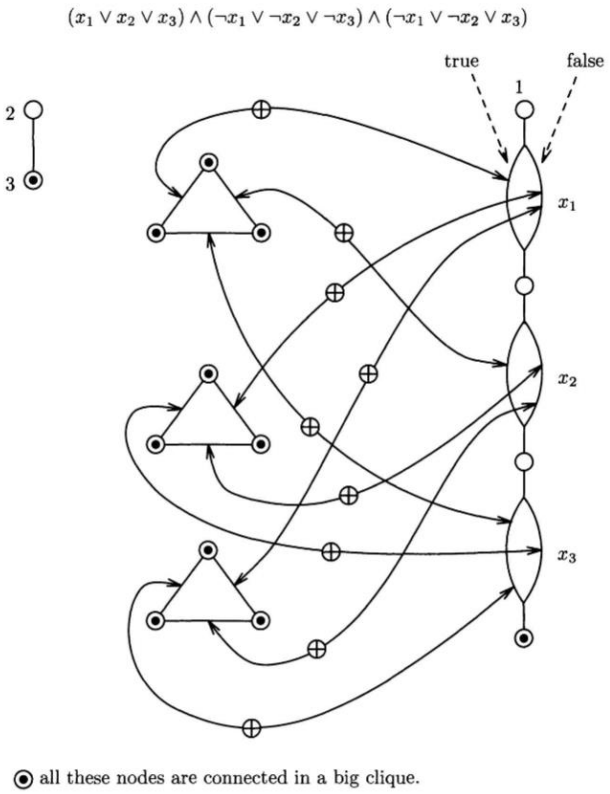


Denne gadget har til formål at holde grafen forbundet og at alle vertices bliver besøgt, men også at indgangspunktet bestemmer udgangspunktet. Som det kan ses på tegningen, så hvis du kommer ind øverst forlader du også gadgeten øverst. Vi bruger XOR tegnet for at gøre det simplere.

Nu vil vi godt repræsentere vores clauses. Dette gør vi med vores "constraint" gadget.



Trekanten er vores 3SAT clause, hvor hver kant er en af variablerne i det clause. Hvis kanten er ”taget” i HAMILTON stigen, så betyder det at variablen er ”false”. Siden det er en trekant, og at kun 2 kanter på tages fordi ellers besøger vi en vertex mere end 1 gang, så har vi altså en utaget kant som betyder at den ene variabel i hvert fald er sand. Vi forbinder hver kant med vores choice gadget for de tilsvarende variabler, hvilket betyder at hvis vi sætter f.eks. x1 til true over i choice, så kan vi ikke tage den tilsvarende kant som jo betød at den var false.



Nu vil vi så godt forbinde slut punktet for vores choice kæde med all kanter i clause trekanterne. Disse skal så også forbindes med en slut node. Vores sti starter altså fra node 1 og slutter i node 2.

Hvis der er en korrekt HAMILTON PATH for denne graf, så må der også være tildelinger til alle clauses som gør at mindst 1 variabel er sand.

Hvis vi allerede har en sand 3SAT instans så kører vi bare igennem choice kæden og tildeler de samme værdier.

HAMILTON PATH til TSP består så bare af at vi sætter alle kanter i grafen som eksistere i HAMILTON tilfældet til at have Distance 1, og alle ikke eksisterende kanter til at have Distance 2. Vores Budget er så n+1, dvs. hele vejen rundt og så kanten tilbage til start. Hvis sådan en vej findes, så er det også en HAMILTON PATH.

Noter:

Alle SAT kan omdannes til 3SAT.

3SAT er NP-hard.

Resolution.

Davis-Putnam (nogen forklar mig den)

2CNF (2SAT and connectivity, MAX2SAT er P)

NAESAT

CircuitSAT <= SAT (3SAT)

I SAT til 3SAT er det så lokale transformationer? Hvordan sætter man dem sammen?

Man bruger resolution til at slette en variable men bevarer løsningen, man bruger det i David Putnam algoritmen

2SAT er i P fordi der er et polynomisk upper bound på hvor stort det bliver efter Resolution (Resolve på 2CNF giver et 2CNF), så David Putnam kan løse den i polynomisk tid. Dette gælder så ikke 3SAT. Resolve på 3CNF giver et 4CNF, Resolve på 4CNF giver et 6CNF.

Spørg Alex om 2CNF graf tings

MAX2SAT er NP complete (reduction from 3SAT to MAX2SAT)

NAESAT (Klart fordi ellers ville 3SAT være P, CircuitSAT <= NAESAT). Når man laver CircuitSAT til 3SAT så er alle 3clauses NAE, så man skal bare tilføje Z til de andre.

NAESAT til 3COLORING tjek noten

Independent Set er ret simpel, og den måde vi har defineret Clique og Vertex Cover som Independent Set problemer gør at de også følger let derfra. For Clique, bare lav det samme på komplementgrafen. Usikker på Vertex Cover.

3SAT to HAMILTON PATH er nice nok.

Corollary: TSP (D) is **NP**-complete.

Proof: We shall reduce HAMILTON PATH to it. Given a graph G with n nodes, we shall design a distance matrix d_{ij} and a budget B such that there is a tour of length B or less if and only if G has a Hamilton path. There are n cities, one for each node of the graph. The distance between two cities i and j is 1 if there is an edge $[i, j]$ in G, and 2 otherwise. Finally, $B = n + 1$. The proof that this works is left to the reader. □

3SAT to TRIPARTITE (EXACT SET COVER, SET COVER og SET PACKING)

SubsetSum medfører at KnapSack er NP hard. Reducer fra EXACT COVER til SUBSET SUM, hvor man ser Sets som integers.