KAPITEL III

LETTE OG SVÆRE PROBLEMTYPER

III.1 INDLEDNING.

I kapitlerne I og II har vi behandlet mange forskellige problemtyper. I begge kapitler har begreberne *god algoritme* og *god sætning* spillet en væsentlig rolle. Disse begreber dukkede op i den matematiske litteratur i 1960'erne, først og fremmest i arbejder af den canadiske matematiker J. Edmonds.

At der er en *god algoritme* for en problemtype betyder, at der eksisterer en potensfunktion, (f.eks. x^2 eller $7 \cdot x^{117}$), så algoritmen for ethvert problem af typen giver det korrekte svar i et antal skridt, som højst er potensfunktionen af størrelsen af problemet (f.eks. n^2 eller $7 \cdot n^{117}$, hvor n er størrelsen af problemet).

At der er en *god sætning* for en problemtype betyder, at der for ethvert muligt svar er en letforklarlig grund til, at svaret er korrekt.

Det er ikke klart, at de to begreber har noget med hinanden at gøre, bortset fra det fælles navn, som Edmonds gav dem. Men som vi har set, specielt i kapitel II, er et bevis for, at en god algoritme virkelig giver korrekt svar for en bestemt problem-type, ofte samtidig et bevis for en god sætning for problem-typen. Og omvendt er det svært at forestille sig et bevis for sætningen uden algoritmen.

Der var én undtagelse fra denne observation af at gode algoritmer og gode sætninger følges ad, nemlig den gode dualitetssætning i lineær programming (LP), hvor den tilsvarende simplex-algoritme i specielle tilfælde ikke opfører sig godt.

I dette kapitel vil vi belyse

> Edmonds' spørgsmål
>
> Eksisterer der en god algoritme for en problem-type når og kun når der eksisterer en god sætning ?

Dette spørgsmål har endnu ikke fundet sin endelige afklaring. Et væsentligt skridt fremad blev taget i begyndelsen af 1970'erne af den canadiske datalog S.A. Cook. Han stillede bl.a. et beslægtet spørgsmål, som i dag må betegnes som den teoretiske datalogi's vigtigste uløste problem.

> Cook's spørgsmål
>
> Eksisterer der en god algoritme for enhver problem-type med to svarmuligheder (JA og NEJ), hvor det kun forudsættes, at der i tilfælde af at svaret er JA er en letforklarlig grund til, at dette svar er korrekt.

Mens man regner med at svaret på Edmonds' spørgsmål har en vis chance for at være bekræftende, så regner man samtidig med, at svaret på Cook's spørgsmål ikke kan være det. Men begge spørgsmål er som sagt uafklarede.

De to spørgsmål, således som de er formulerede ovenfor, er ikke umiddelbart matematiske spørgsmål, men spørgsmål om matematik.

Spørgsmål om matematik har beskæftiget filosoffer og matematikere i tusinder af år. Et spørgsmål, som har været intenst studeret, er: "Er aritmetik konsistent ?" Hvilken garanti har vi for, at der ikke en dag kommer et geni og beviser, at 2+2 = 5 uden at der er fejl i beviset ?

Spørgsmålet om artimetikkens konsistens blev besvaret af den østrigske matematiker og logiker K. Gödel i 1931. Gödel omformede spørgsmålet fra at være et spørgsmål om aritmetik til at være et spørgsmål i aritmetik. Han kunne dermed genialt vise det overraskende svar, at dette spørgsmål ikke kan besvares: hvis aritmetik virkelig er konsistent, så er den aritmetiske version af dette udsagn en sand sætning, som ikke kan bevises med de metoder og teorier, vi benytter i aritmetikken !

Edmonds' og Cook's spørgsmål kan også gives en helt præcis matematisk formulering, dvs. de bliver til spørgsmål i matematik i stedet for spørgsmål om matematik. Den matematiske formulering giver mulighed for en mere præcis behandling af spørgsmålene. Cook kunne herved vise, at der eksisterer problem-typer, de såkaldte NP-*komplette*,

hvor eksistensen af en god algoritme for blot én NP-komplet problem-type medfører, at svaret på Cook's problem er bekræftende.

Et eksempel på en sådan NP-komplet problem-type er problemet om den handelsrejsende. Hvis der findes en god algoritme for blot denne ene problem-type, så er svaret på Cook's problem altså bekræftende. Da man ikke regner med, at svaret kan være bekræftende, betyder det, at man heller ikke kan regne med, at der eksisterer en god algoritme for problem-typen om den handelsrejsende !

Nogle matematikere har foreslået, at vi med Edmonds' og Cook's spørgsmål måske også står overfor spørgsmål, som ikke kan besvares. De fleste er dog ikke så pessimistiske, men ingen venter, at svarene ligger lige om hjørnet, selv om der udfoldes store anstrengelser verden over.

III.2 GODE ALGORITMER.

En *god* algoritme for en problem-type definerede vi i §III.1 som en algoritme, hvor antal skridt algoritmen skal gennemføre, højst er en potensfunktion i størrelsen af input. Denne definition udelukker bl.a. at antal skridt kan være en exponentialfunktion i størrelsen af input, jvf. Opgave 3.1.

Opgave 3.1 Lad $c \cdot x^a$ (a og c konstanter > 0) være en potensfunktion og lad $d \cdot b^x$ (b og d konstanter med d > 0 og b > 1) være en eksponentialfunktion. Vis, at $c \cdot x^a < d \cdot b^x$ for alle tilstrækkeligt store x.

(Vink: Vis, at $\ell n(c \cdot x^a) < \ell n(d \cdot b^x)$).

Til belysning af det rimelige i ovenstående definition af en god algoritme er følgende tabeller slående:

Tidsforbrug for computer med 10^6 skridt pr. sek.

Antal skridt \ størrelse n	10	20	30	40	50	60
n	0,00001 sek.	0,00002 sek.	0,00003 sek.	0,00004 sek.	0,00005 sek.	0,00006 sek.
n^2	0,0001 sek.	0,0004 sek.	0,0009 sek.	0,0016 sek.	0,0025 sek.	0,0036 sek.
n^3	0,001 sek.	0,008 sek.	0,027 sek.	0,064 sek.	0,125 sek.	0,216 sek.
n^5	0,1 sek.	3,2 sek.	24,3 sek.	1,7 min.	5,2 min.	13,0 min.
2^n	0,001 sek.	1,0 sek.	17,9 min.	12,7 dage	35,7 år	36600 år
3^n	0.059 sek.	58 min.	6.5 år	385500 år	$2 \cdot 10^{10}$ år	$1{,}3 \cdot 10^{15}$ år

Størrelse af største løsbare problem.

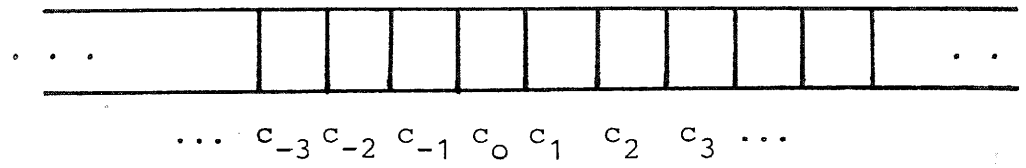
Antal skridt	Nuværende computer	Computer 100 gange hurtigere.	Computer 1000 gange hurtigere.
n	N_1	$100\ N_1$	$1000\ N_1$
n^2	N_2	$10\ N_2$	$31{,}6\ N_2$
n^5	N_3	$2{,}5 \cdot N_2$	$3{,}98 \cdot N_3$
2^n	N_4	$N_4 + 6{,}64$	$N_5 + 9{,}97$
3^n	N_5	$N_5 + 4{,}19$	$N_5 + 6{,}29$

Opgave 3.2 Kontrollér et udvalg af pladser i tabellerne.

Den givne definition af en god algoritme er lidt løs, idet det ikke præcis siges, hvad et skridt er. For at få dette præciseret er det nødvendigt først at få præciseret, hvad en algoritme er - vi kan ikke klare os med snakken i §I.1.

En præcis definition af hvad en algoritme er, blev givet af flere matematikere og logikere uafhængigt af hinanden i 1930'erne. Heldigvis viste de forskellige forslag sig at være ækvivalente. Den kendteste definition skyldes englænderen A.M. Turing. Han definerede begrebet algoritme vha. det, der siden er blevet kendt som en Turing-maskine.

En *Turing-maskine* består af endelig mange tilstande t_o , t_1 , t_2 , ..., t_n , ! , hvoraf t_o er *start-tilstanden* og ! er *slut-tilstanden*. Endvidere består Turing-maskinen af et endeligt alfabet a_1 , a_2 , ... , a_m og et dobbelt-uendeligt *bånd*

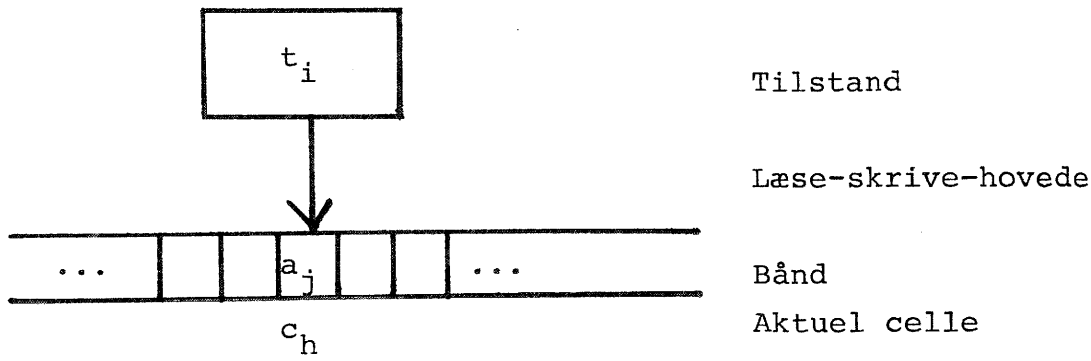


med celler ..., c_{-3} , c_{-2} , c_{-1} , c_o , c_1 , c_2 , c_3 , ...
Hver celle kan være enten tom eller indeholde ét bogstav fra det givne alfabet.

Input til Turing-maskinen er en udfyldning af endelig mange af båndets celler c_1 , c_2 , ... , c_i med bogstaver fra alfabetet. *Output* er det, der står på båndet, når maskinen kommer i slut-tilstanden.

Turing-maskinen starter i start-tilstanden t_o og læser indholdet i c_1 . I et vilkårligt *skridt* er den i en tilstand t_i og læser indeholdet a_j i celle c_h . Afhængig af t_i og a_j sker følgende: 1) indholdet i cellen udskiftes med et nyt bogstav (evt. det samme) eller cellen tømmes, 2) der skiftes tilstand (evt. bliver maskinen i uændret tilstand), og 3) der flyttes til en ny celle enten én til højre (H) eller én til venstre (V) for den aktuelle celle, eller man bliver i samme celle (S).

Så er maskinen klar til næste skridt.



En *problem-type* defineres som en vis mængde af ord i det givne alfabet. Til hvert ord tilknyttes et andet ord i alfabetet som svar. En *algoritme* for en problem-type er en Turing-maskine, som med et vilkårligt ord fra problem-typen som input efter et endeligt antal skridt kommer i slut-tilstanden med det korrekte svar som output.

Størrelsen af input er antal udfyldte celler i starten. Da også antal skridt nu er veldefineret, hvar begrebet *god algoritme* for en problem-type hermed fået en hel præcis betydning.

At det vi intuitivt forstår ved en algoritme netop indfanges af ovenstående præcise definition er ikke klart, men kan sandsynliggøres gennem eksempler.

Eksempel 3.1 En Turing-maskine, som ud fra et ikke-negativt helt tal n (opskrevet i 10-tals-systemet) giver tallet n+1, kan gives ved følgende tabel, hvor der i hvert felt som funktion af bogstav og tilstand er angivet (nyt bogstav, ny celle, ny tilstand):

Alfabet \ Tilstand	t_o	t_1
0	(0, H, t_o)	(1, - , !)
1	(1, H, t_o)	(2, - , !)
2	(2, H, t_o)	(3, - , !)
3	(3, H, t_o)	(4, - , !)
4	(4, H, t_o)	(5, - , !)
5	(5, H, t_o)	(6, - , !)
6	(6, H, t_o)	(7, - , !)
7	(7, H, t_o)	(8, - , !)
8	(8, H, t_o)	(9, - , !)
9	(9, H, t_o)	(0, V, t_1)
tom	(tom, V, t_1)	(1, - , !)

Opgave 3.3 a) Hvorledes virker Turing-maskinen i Eksempel 3.3 med input 388, med input 389, og med input 999 ?

b) Hvor mange skridt tager Turing-maskinen i Eksempel 3.3 maksimalt med et tal med n cifre som input ?

III.3 GODE SÆTNINGER.

At der er en *god sætning* for en problem-type definerede vi i §III.1 til at betyde, at der for ethvert muligt svar er en letforklarlig grund til, at svaret er korrekt.

For at begrænse svar-mulighederne for problem-typer og for bedre at kunne sammenligne forskellige problem-typer indskrænker vi os til i resten af dette kapitel kun at betragte *afgørelses-problem-typer*, dvs. problem-typer med kun to mulige svar JA og NEJ. Problem-typen om en givet graf har en Euler-tur eller ej, er en sådan afgørelses-problem-type. Og f.eks. et minimum LP-problem kan formuleres som problemet for ethvert tal M at finde ud af, om der er en brugbar løsning med værdi $\leq$ M, og det bliver dermed også en afgørelses-problem-type. Der ligger således ingen væsentlig indskrænkning i kun at betragte afgørelses-problem-typer.

Cook definerede, at en afgørelses-problem-type tilhører klassen P, hvis der er en god algoritme for problem-typen til at finde det korrekte svar JA eller NEJ (bogstavet P står for *polynomial*). Han definerede endvidere, at en afgørelses-problem-type tilhører klassen NP, hvis det korrekte svar JA altid har en letforklarlig grund (bogstaverne NP står for *non-deterministic polynomial*. Grunden til dette navn forklares senere).

Hvad er en *letforklarlig grund* ? Det må være en grund, som kan efterprøves hurtigt, dvs. af en god algoritme. Vi ledes derfor til følgende mere præcise definition: En afgørelses-problem-type D tilhører NP hvis der eksisterer en algoritme A og en potensfunktion $c \cdot x^a$, således at svaret er JA for et problem d af typen D hvis og kun hvis der eksisterer et ord i(d), så A med input [i(d) og d] giver svaret JA i højst $c \cdot |d|^a$ skridt, hvor |d| er størrelsen af d.

Ordet i(d) er den letforklarlige grund, som eksisterer for et d med svar JA. Der er altså tale om, at algoritmen A med input [i(d) og d] hurtigt vil give svaret JA for d , men det kræver for et givet d , at vi først har gættet i(d). Her kommer det ikke-deterministiske element ind, idet der ingen anvisning gives på, hvordan i(d) skal findes. Hvis d har svar NEJ, så vil intet ord sammen med d som input til A givet svaret JA i højst $c \cdot |d|^a$ skridt.

Med den præcisering, vi tidligere gav af begrebet algoritme, får klasserne P og NP hermed en hel præcis betydning.

De fleste problem-typer tilhører NP. Spørger vi f.eks. i problemet for den handelsrejsende, om der eksisterer en rundrejse af længde $\leq$ L , så har svaret JA en letforklarlig begrundelse, nemlig fremvisningen af en rundrejse af længde $\leq$ L. Det lyder meget usandsynligt at alene det, at en problem-type tilhører NP skulle betyde, at den tilhører P. Men Cook bemærkede, at det synes vanskeligt at bevise, at NP $\neq$ P , og at det altså ikke kan udelukkes, at NP = P, selv om det næsten er for godt til at være sandt.

Med de indførte præcise definitioner kan spørgsmålene i §III.1 nu gives en matematisk formulering.

Cook's spørgsmål
Er NP = P ?

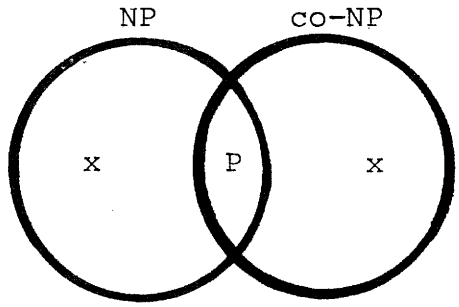
Svarende til definitionen af NP defineres en klasse co-NP blot ved at erstatte JA med NEJ. Dvs. en afgørelses-problem-type til-

hører klassen co-NP, hvis svaret NEJ altid har en letforklarlig grund. At der er en god sætning for en afgørelses-problem-type betyder altså, at den er med i både NP og co-NP. Edmonds' spørgsmål lyder derfor

> Edmonds' spørgsmål
>
> Er NP ∩ co-NP = P ?

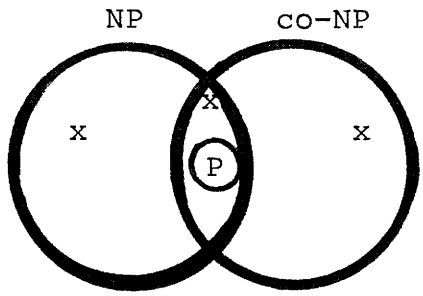
Med de præcise definitioner, vi har givet af klasserne i Edmonds' spørgsmål, er P en del af NP og co-NP. Endvidere kan bemærkes, at hvis svaret på Cook's spørgsmål er, at P = NP, så er også P = co-NP [Bevis: Hvis D er en problem-type i co-NP, så er den modsatte problem-type $\hat{D}$ (med svar JA for et problem i D hvis og kun hvis svaret er NEJ for det tilsvarende problem i $\hat{D}$) i NP og dermed i P. Men hvis $\hat{D}$ er i P, så er også D i P ■].

De mulige placeringer af de tre klasser er dermed (idet et kryds betegner "ikke-tom"):



JA til Edmonds

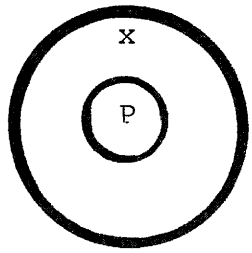
NEJ til Cook



NEJ til Edmonds

NEJ til Cook

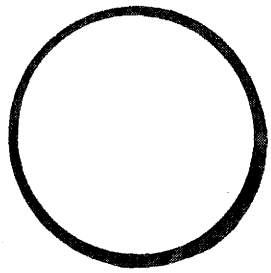
co-NP = NP



NEJ til Edmonds

NEJ til Cook

co-NP = NP = P



JA til Edmonds

JA til Cook

Den sidstnævnte mulighed må betragtes som usandsynlig. Den førstnævnte mulighed regnes blandt forskere som den mest sandsynlige.

At afgøre hvilken af de fire muligheder, der er den korrekte, er et af de mest udfordrende uløse problemer i den teoretiske datalogi.

III.4 NP-KOMPLETHED.

Cook's hovedresultat i forbindelse med problemet om NP = P er at vise, at der eksisterer en problem-type i NP, som er vanskeligere end alle andre. Denne problem-type kaldes *opfyldelighed*. Et eksempel på et problem af denne type er at afgøre, om det logiske udtryk

$(x_1$ eller $\bar{x}_2$ eller $\bar{x}_3)$ og $(x_1$ eller $x_2)$ og

$(\bar{x}_1$ eller x_2 eller $x_3)$ og $(\bar{x}_1$ eller $\bar{x}_2)$ og $(\bar{x}_1$ eller x_2 eller $\bar{x}_3)$

kan gøres sand ved at give de variable x_1 , x_2 , ... passende sandhedsværdier ($\bar{x}_i$ betegner "ikke x_i", som er sand hvis og kun hvis x_i er falsk). Svaret for problemet ovenfor er JA. En letforklarlig grund hertil er, at ($x_1 = x_3$ = falsk og x_2 = sand) gør én af alternativerne i hver parantes sand, og derfor gøres hele udtrykket sandt. Bemærk i denne sammenhæng, at det samlede antal muligheder for sandhedsværdier til n variable er 2^n.

Som antydet ovenfor er opfyldelighed en problem-type i NP. Vha. den præcise definition på, hvad NP er, kunne Cook vise, at

> Theorem 3.1 (Cook 1971)
>
> a) Et problem fra en vilkårlig problem-type i NP kan formuleres som et opfyldeligheds-problem, hvis størrelse højst er en potensfunktion (afhængig af problem-typen) af størrelsen af problemet.
>
> b) Hvis der eksisterer en god algoritme for opfyldeligheds-problem-typen, så eksisterer der gode algoritmer for alle problem-typer i NP.

b) følger direkte af a). Vi skal ikke vise a), men blot sige, at beviset går ud på at "simulere" den Turing-maskine, der benyttes til at prøve den letforklarlige grund, vha. et

logisk udtryk. Et godt "gæt" i(d) for et problem d med svar JA eksisterer når og kun når der eksisterer sandhedsværdier, som gør det logiske udtryk sandt.

En problem-type i NP, for hvilken eksistensen af en god algoritme medfører, at NP = P, kaldes NP-*komplet*. Sætning 3.1 b) viser altså, at opfyldeligheds-problem-typen er NP-komplet.

Den amerikanske matematiker R.M. Karp viste i 1972, at opfyldelighed ikke er enestående som et NP-komplet problem. Denne problem-type kan nemlig reduceres til en række andre problem-typer, således at hvis der er en god algoritme for én af disse andre problem-typer, så er der også for opfyldelighed, dvs. NP = P. Alle disse andre problem-typer er derfor også NP-komplette.

Af problem-typer, vi har mødt, som er NP-komplette, kan nævnes:

Postbud-problemet for blandede grafer;
Korteste veje med negative længder tilladt;
Den handelsrejsendes problem.

Listen over kendte NP-komplette problemer er meget lang. En problem-type, som er NP-komplet, betragtes normalt som umulig at løse vha. en god algoritme, idet man som sagt regner med at NP ≠ P.

På den anden side er det at finde en god algoritme for blot ét eneste NP-komplet problem en ønskedrøm, som med ét slag vil betyde en meget kraftig effektivisering af mulighederne for anvendelse af computere.

↓ Lad os slutte med at vise en reduktion af opfyldelighed til et grafteoretisk problem. Vi ønsker at undersøge, om

$$U = c_1 \text{ og } c_2 \text{ og } \ldots \text{ og } c_p$$

kan gøres sand, idet hvert c_i består visse af

$$x_1, x_2, \ldots, x_n, \bar{x}_1 = x_{n+1}, \bar{x}_2 = x_{n+2}, \ldots, \bar{x}_n = x_{2n}$$

med eller imellem.

Dan en graf G med punkter (i,j), hvor $i = 1,2,\ldots,2n$ og $j = 1,2,\ldots,p$. Forbind (i,j) og (r,s) med en kant i G hvis

$$(x_i \in c_j) \text{ og } (x_r \in c_s) \text{ og } j \neq s \text{ og } (x_i \neq \overline{x}_r).$$

Der gælder nu, at G indeholder en komplet p-graf, dvs. p punkter som alle er forbundet med en kant to og to, hvis og kun hvis der er sandhedsværdier, som gør U sand (overvej !). Endvidere er $|P(G)| = 2n\,p \leq 2 \cdot (U\text{'s størrelse})^2$. Kunne vi altså finde en god algoritme for den grafteoretiske *klike-problem-type* (dvs. finde ud af om der eksisterer komplette delgrafer af givne størrelser i givne grafer), så kunne vi også løse opfyldelighed med en god algoritme, og dermed vise, at NP = P.

† Klike-problem-typen er altså også NP-komplet.

III.5 Lineær Programmering.

For lineær programmering (LP) er dualitetssætningen en god sætning, mens den sædvanlige algoritme, simplex-algoritmen, ikke er en god algoritme. I praksis fungerer den dog godt, men der kan gives eksempler, hvor antal skridt er eksponentielt stort i størrelsen af input.

LP blev derfor i mange år betragtet som et muligt modeksempel til Edmonds' formodning om, at gode sætninger og gode algoritmer følges ad.

Det vakte derfor meget stor opsigt, da der i 1979 i et russisk tidsskrift udkom en artikel med titlen: "En polynom algoritme for LP" skrevet af en i Vesten ukendt russisk matematiker L.G. Khachian. Det gav i amerikanske aviser anledning til store avisoverskrifter af typen: "Discovery rocks World of mathematics and computers", "Mathematicians amazed by Russian's discovery", "The Russian Genius who has rocked the computer world", osv.

Når den nye algoritme vakte så stor opsigt blandt matematikere, skyldtes det, at man nu fik vist, at der for LP virkelig er en god algoritme (hvilket styrker tilliden til Edmonds' formodning), og det skyldtes for det andet håbet om, at den nye algoritme ikke blot teoretisk,men også i praksis ville vise sig mere effektiv end simplex-algoritmen.

Dette sidste håb blev dog ret hurtigt gjort til skamme. Selv om den nye algoritme er en god algoritme, viste den sig i praksis på de fleste LP-problemer at fungere dårligere end simplex-algoritmen.

Den nye algoritme virker på en noget anden måde end simplex-algoritmen. Simplex-algoritmen fører fra hjørne til hjørne af det brugbare område, mens den nye algoritme fører til mindre og mindre ellipsoider (eller i to dimensioner ellipser), som alle indeholder den optimale løsning. Som konsekvens heraf kaldes metoden for *ellipsoide-metoden*.

Ellipsoide-metoden finder altås ikke umiddelbart en præcis optimal løsning, men kommer vilkårligt tæt på én. Desuden skal der undervejs uddrages kvadratrødder, og man bliver således også af den grund nødt til at regne med tilnærmelse, dvs. med et vist

antal decimaler.

Hvordan skal det så forstås, at ellipsoide-metoden er en god algoritme ? Lad os antage, at vi betragter et LP-problem med n variable og m begrænsninger

$$\text{Find min } \Sigma c_i x_i$$

$$\text{ufa. } \Sigma a_{ij} x_i \geqq b_j \quad j=1,\ldots,m$$

$$x_i \geqq 0 \quad i=1,\ldots,n$$

hvor alle koefficienter c_i , a_{ij} og b_j er heltallige. Hvad er størrelsen af dette problem ? Vi kunne ved første øjekast fristes til at sige $m \cdot n + m+n$ (dette er det samlede antal koefficienter), men dette er ikke helt nok, hvis disse koefficienter kan have mange cifre. Det vil være mere rimeligt i størrelsen af problemet også at indregne det samlede antal cifre i koefficienterne forskellige fra 0 (i f.eks. 10-talssystemet). Som størrelsen af LP-problemet kunne vi således angive

$$L = m \cdot n + m+n + \sum_{a_{ij} \neq 0} \lceil \log|a_{ij}| \rceil + \sum_{b_j \neq 0} \lceil \log|b_j| \rceil + \sum_{c_i \neq 0} \lceil \log|c_i| \rceil$$

hvor $\lceil a \rceil$ for et reelt a er det mindste hele tal $> a$.

Også i forbindelse med simplex-algoritmen kommer antal cifre i koefficienterne til at spille en rolle. Vi har set, at selv om alle koefficienter er heltallige, behøver koordinaterne til et optimalt punkt ikke at være det, men disse koordinater er dog altid rationale, dvs. brøker. For at angive et præcist optimalt punkt skal vi altså angive tæller og nævner for alle koordinater. Antal cifre i disse tal afhænger af antal cifre i de oprindelige koefficienter, dvs. af L. Så i enhver algoritme for LP kommer L til at indgå i antallet af skridt.

I et LP-problem som ovenfor kan man på forhånd begrænse størrelsen af nævneren i koordinaterne i en optimal løsning som funktion af L (lad os sige, at alle nævnere er $\leq$ N). Når vi så i ellipsoide-algoritmen er nået indenfor en afstand $< \frac{0,5}{N}$ af optimum, så fås det virkelige optimum ved at tage nærmeste hele tal i tællerne af brøker med mindst N som nævner.

Ellipsoide-algoritmen finder altså også i et endeligt antal skridt den præcise løsning til LP-problemet ovenfor, og når vi siger, at ellipsoide-algoritmen er god, betyder det, at denne præcise løsning findes i et antal skridt, som højst er en for ellipsoide-metoden fast potensfunktion i L.

I efteråret 1984 vakte det betydelig opsigt blandt matematikere, at en amerikansk matematiker N. Karmarkar på Bell Laboratories udviklede en ny god algoritme for LP, kaldet *projektions-algoritmen*. Der er igen tale om en algoritme, som regner med tilnærmelse, idet der også her skal uddrages kvadratrødder. Men antallet af skridt er betydeligt mindre end ved ellipsoide-algoritmen.

Det hævdes, at projektions-algoritmen ikke blot er god i teorien, men også er langt overlegen den sædvanlige simplex-algoritme i praksis (hvad ellipsoide-metoden jo ikke var). For problemer med adskilllige tusinde variable siges projektions-algoritmen at være 50 gange hurtigere end standard-versioner af simplex-algoritmen. Om dette virkelig er tilfældet undersøges i øjeblikket grundigt. Hvis det er rigtigt, er der tale om et nyt gennembrud af stor betydning for løsning af mange praktiske problemer vha. computere.

§II.6 LITTERATUR.

Begreberne "god algoritme" og "god sætning" er først beskrevet uformelt i den af J. Edmonds i §I.7 nævnte artikel.

Den mere formelle behandling samt definitioner af P , NP , co-NP og NP-komplette problem-typer blev givet i

> S.A. Cook: The complexity of theorem proving procedures, Proc. 3rd ACM Symp. on the theory of computing, ACM (1971), side 151-158.

Konsekvenserne af Cook's arbejde for kombinatoriske problemer, bl.a. den handelsrejsende, blev gjort klar i

> R.M. Karp: Reducibility among combinatorial problems, i "Complexity of Computer Computations" (ed. Miller & Thatcher), Plenum Press (1972), side 85-103.

En meget grundig og velskrevet gennemgang af Cook's og Karp's teorier, samt et katalog over 300 NP-komplette problem-typer, findes i

> M.R. Garey & D.S. Johnson: Computers and intractability. A guide to the theory of NP-completeness, Freeman 1979.

Cook's sætning findes bevist ret simpelt i bøgerne

> S. Even: Graph algorithms, Computer Science Press 1979.

og

> C.H. Papadimitriou & K. Steiglitz: Combinatorial optimization. Algorithms and complexity, Prentice-Hall 1982.

Ellipsoide-algoritmen beskrives også i bogen af Papadimitriou & Steiglitz, og i den i §II.6 nævnte bog af Chvátal. En artikel herom er

> Näslund: Hacijan's algorithm, Normat 29 (1981), side 162-169.

En glimrende beskrivelse af ellipsoide-metoden og dens konsekvenser er

L. Lovász: A new linear programming algorithm - better or worse than the simplex method ? Math. Intelligencer 2 (1979-80), side 141-148.

I samme tidsskrift samme år er der en morsom artikel om bl.a. pressens behandling af ellipsoide-metoden:

E.L. Lawler: The great mathematical sputnik of 1979, Math. Intelligencer 2 (1979-80), side 191-198.

Turing-maskiner beskrives i bl.a.

B.A. Trakhtenbrot: Algorithms and automatic computing machines, D.C. Heath 1963.

H.R. Lewis & C.H. Papadimitriou: Elements of the theory of computation, Prentice Hall 1981.

Gödel's teori findes godt beskrevet i

E. Nagel & J.R. Newman: Gödel's proof, Routledge & Kegan Paul Ltd. 1959.

Kamarkar har offentliggjort den nye projektions-algoritme for LP i et ungarsk tidsskrift :

N.Karmarkar : A new polynomial-time algorithm for linear programming, Combinatorica 4 (1984), side 373-395.