

# Branch-and-bound

David Pisinger

Videregående algoritmik, DIKU (2005-06)

## Indhold

<b>1</b>	<b>Introduktion</b>	<b>5</b>
1.1	Gennemgående eksempler . . . . .	7
<b>2</b>	<b>Brute-force metoder</b>	<b>10</b>
<b>3</b>	<b>Divide and Conquer</b>	<b>11</b>
<b>4</b>	<b>Grænseværdier</b>	<b>15</b>
4.1	Eksempler på grænseværdier . . . . .	17
4.2	Grænseværditest . . . . .	21
4.3	Monotonitet af grænseværdier . . . . .	22
<b>5</b>	<b>Branch-and-bound</b>	<b>22</b>
5.1	Øvre grænseværdifunktion . . . . .	23
5.2	Nedre grænseværdi . . . . .	24
5.3	Søgestrategi . . . . .	24
5.4	Forgreningsregel . . . . .	26
5.5	Eksempler på branch-and-bound algoritmer . . . . .	26

<b>6</b>	<b>Kvalitet af grænseværdifunktionen</b>	<b>31</b>
6.1	Eksempler på dominans . . . . .	31
<b>7</b>	<b>Kritiske og Semikritiske delproblemer</b>	<b>34</b>
<b>8</b>	<b>Kunsten at designe en god branch-and-bound algoritme</b>	<b>36</b>
<b>9</b>	<b>Opgaver</b>	<b>38</b>
	<b>Index</b>	<b>42</b>

## Forord

Disse noter er skrevet til kurset *Videregående algoritmik* på Datalogisk Institut, Københavns Universitet.

Formålet med noterne er at give en samlet indføring til branch-and-bound paradigmet. I modsætning til majoriteten af tilsvarende tekster forudsætter noterne ikke kendskab til lineær programmering. Til gengæld bygger de videre på terminologi og principper fra Cormen m.fl. [7]. Da noterne er tiltænkt algoritmik-undervisning på 2.-3. studieår er noterne skrevet på dansk. Dette giver samtidig mulighed for at indføre en dansk terminologi på kurset. Nogle begreber har det dog ikke været muligt (eller hensigtsmæssigt) at oversætte til dansk, hvorfor de engelske betegnelser er bibeholdt.

Som inspiration er benyttet noterne “Branch-and-bound algorithms”, af Clausen [6], samt bøgerne “Integer Programming” af Wolsey [24] og “Elements of the Theory of Computation” af Lewis og Papadimitriou [16].

For at noterne også kan bruges som mini-opslagsbog, er der udarbejdet et *stikordsregister* sidst i noterne.

En stor tak til alle som har læst første version af disse noter, og som har bidraget til at forbedre dem.

### Kommentarer til 2. udgave

En række fejl, heriblandt figur 3 er blevet rettet. Kapitel 8 er blevet opdateret med nyeste litteratur omkring branch-and-bound paradigmet.

9. oktober 2005, DP.

## 1 Introduktion

Teorien om  $\mathcal{NP}$ -fuldstændighed giver os en velbegrunderet formodning om, at der er en lang række problemer, som vi ikke kan forvente at løse i polynomiel tid. Da problemerne dog kan være vigtige at løse for industri eller samfund, er det essentielt at finde løsningsmetoder til sådanne problemer. Selv om vi ikke kan garantere polynomiel køretid af sådanne algoritmer, kan man håbe på at køretiden er "rimelig" for de fleste instanser, som forekommer i praksis. Teorien om  $\mathcal{NP}$ -fuldstændighed udtaler sig kun om, at der *eksisterer* instanser med ubehagelig køretid, men den udelukker ikke at mange instanser kan løses i polynomiel tid.

Hvor vi i teorien om  $\mathcal{NP}$ -fuldstændighed fandt det hensigtsmæssigt at betragte *afgørlighedsproblemer*, vil man i praktiske anvendelser snarere betragte det tilhørende *optimeringsproblem*. F.eks. vil man for traveling salesman-problemet næppe være tilfreds med at få et ja-nej svar på, om der findes en Hamilton-kreds af længde højst  $k$  i grafen. Man vil snarere spørge efter den korteste Hamilton-kreds.

Formelt set kan et optimeringsproblem i *maksimeringsform* defineres som:

$$z = \max\{f(x) \mid x \in S\}. \quad (1)$$

Her angiver  $f(x)$  *objektfunktionen*, mens  $S$  er *løsningsrummet* og  $z$  er den *optimale løsningsværdi*. Ofte vil man også være interesseret i den *optimale løsning*, dvs. det  $x^*$  hvor  $z = f(x^*)$ . En række optimeringsproblemer som f.eks. traveling salesman-problemet er defineret i *minimeringsform*

$$z = \min\{f(x) \mid x \in S\}. \quad (2)$$

Man kan omforme et minimeringsproblem på formen (2) til et maksimeringsproblem ved at maksimere  $-f(x)$ . Vi vil i disse noter indføre alle definitioner for et maksimeringsproblem. Det overlades til læseren at definere de tilsvarende begreber for minimeringsproblemer.

Da klassen  $\mathcal{NP}$  kun er defineret for afgørlighedsproblemer, giver det ikke mening at snakke om  $\mathcal{NP}$ -fuldstændighed af optimeringsproblemer. Vi definerer derfor, at et optimeringsproblem er  *$\mathcal{NP}$ -hårdt*, hvis det tilhørende afgørlighedsproblem er  *$\mathcal{NP}$ -fuldstændigt*. Bemærk, at selv om man får forelagt den optimale løsning  $x^*$  for et  $\mathcal{NP}$ -hårdt optimeringsproblem, kender vi ikke en effektiv metode til at verificere at den givne løsning er optimal. Rent faktisk kan det kun bevises at en løsning er optimal ved at gennemløbe hele løsningsrummet.

Hvor polynomielle problemer typisk kan løses ved hjælp af en *konstruktiv* algoritme, der gradvist opbygger en løsning, må vi for  $\mathcal{NP}$ -hårde problemer ty til *søgebaserede* algoritmer, dvs. algoritmer som gennemløber hele eller dele af løsningsrummet for at finde den optimale løsning.

I disse noter vil vi betragte *branch-and-bound paradigmet*, som netop er en søgebaseret algoritme. Der findes en række danske betegnelser for branch-and-bound, f.eks. del-og-hersk, forgren-og-begræns. Ingen af disse betegnelser er dog slået igennem i stort omfang, hvorfor man typisk blot fastholder den engelske betegnelse.

Branch-and-bound er nok det mest benyttede værktøj til løsning af  $\mathcal{NP}$ -hårde kombinatoriske optimeringsproblemer. Branch-and-bound er dog et *paradigme* (dvs. en slags *skabelon*) hvor en række konkrete valg skal træffes for hvert enkelt optimeringsproblem. Som disse noter vil vise, er der en bred vifte af muligheder for hver komponent af branch-and-bound paradigmet, og det er en kunst at kombinere de rette komponenter til en vellykket algoritme.

Princippet i branch-and-bound er en total gennemøgning af løsningsrummet, hvor man bruger nogle matematiske overvejelser til at udelukke dele af løsningsrummet. En vel-designet branch-and-bound algoritme bør udelukke store dele af løsningsrummet, således at den for de fleste praktisk forekommende instanser kun skal gennemøge en lille del af løsningsrummet. I afsnit 2 vil vi først designe en simpel algoritme til løsning af kombinatoriske optimeringsproblemer baseret på total gennemøgning af løsningsrummet. I det følgende afsnit 3 benytter vi divide-and-conquer paradigmet til at gennemøge løsningsrummet, ved fortløbende at dele problemet op i mindre delproblemer og "samle" løsningerne op fra disse. For at undgå en gennemøgning af alle delløsningsrum benytter vi grænseværdier, som defineres i afsnit 4. En grænseværdi er et tal knyttet til et delløsningsrum, som siger noget om, hvor gode løsninger vi kan forvente at finde i det givne delløsningsrum. Hvis vi ved hjælp af grænseværdien for et delløsningsrum kan se, at vi ikke vil være i stand til at forbedre den nuværende løsning, kan vi forkaste delløsningsrummet. Dette er ideen i grænseværditesten, som beskrives i afsnit 4.2. Sætter vi disse grundelementer sammen, har vi branch-and-bound paradigmet, som beskrives i afsnit 5. Vi vil gerne have så stramme grænseværdier som muligt, idet disse vil gøre det muligt at bortskære større dele af løsningsrummet. I afsnit 6 opstiller vi en formel ramme for at sammenligne kvaliteten af grænseværdier. Afsnit 7 ser på, hvilke delproblemer vi altid vil skulle behandle uanset søgestrategi og initial løsning. Noterne afsluttes i afsnit 8 med forskellige tips til at designe vellykkede branch-and-bound algoritmer i praksis.

I disse noter vil vi benytte tre gennemgående optimeringsproblemer til at illustrere principperne. De første to er *maksimeringsproblemerne* knapsack-problemet og dense subgraph-problemet, mens det sidste problem er *minimeringsproblemet* traveling salesman-problemet. Vær opmærksom på, at alle definitioner skal “vendes om” når vi betragter et minimeringsproblem. Traveling salesman-problemet er med vilje medtaget for at træne læseren i denne proces. Alle eksempler med traveling salesman-problemet vil dog tydeligt være mærket med en overskrift som minder om, at det er et minimeringsproblem.

## 1.1 Gennemgående eksempler

### Eksempel 1 Knapsack-problemet

Knapsack-problemet kan defineres på følgende vis: Lad der være givet  $n$  genstande, som hver har en tilknyttet *profit*  $p_j$  og *vægt*  $w_j$ . Udvælg en delmængde af genstandene således, at den samlede profitsum bliver maksimeret uden, at den tilhørende vægtsum overstiger en given grænse  $c$ , kaldet *kapaciteten*.

Trods den simple struktur har knapsack-problemet utallige anvendelser. Det opstår i adskillige transportproblemer (f.eks. ladning af genstande i containere med vægtbegrænsning  $c$ ), udskæringsproblemer (f.eks. udskæring af tømmer som har længde  $c$  i mindre stykker  $w_j$  med forskellig salgspris  $p_j$ ), investering (der er en mængde kapital  $c$  til rådighed, som kan investeres i et antal projekter med profit  $p_j$  og pris  $w_j$ ) eller budgetlægning (afdelingens budget er  $c$ , og der skal udvælges et antal projekter, som har størst mulig samlet nytteværdi). For en komplet oversigt over eksakte algoritmer og approximationsalgoritmer for knapsack-problemet, se [13]. En samling af testinstanser for knapsack-problemet findes i [19].

Hvis vi bruger den binære variabel  $x_j$  til at angive om genstand  $j$  vælges eller ej, får vi følgende matematiske definition af problemet:

$$z = \max \left\{ \sum_{j=1}^n p_j x_j \mid \sum_{j=1}^n w_j x_j \leq c, x_j \in \{0, 1\} \right\}. \quad (3)$$

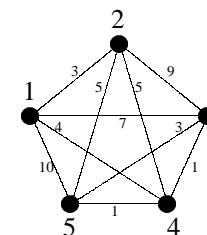
Bemærk, at hvis genstand  $j$  vælges, dvs.  $x_j = 1$ , så tælles  $p_j$  og  $w_j$  med i henholdsvis profitsummen og vægtsummen, mens hvis  $x_j = 0$  bidrager genstand  $j$  ikke til nogen af summerne. Det antages normalt, at alle koefficienter  $p_j$ ,  $w_j$  og  $c$  er positive heltal. Se evt. Cormen m.fl. [7] s. 382 for yderligere beskrivelse af problemet.

I det følgende eksempel er  $c = 9$ , og der er givet  $n = 7$  genstande med følgende profiler og vægte:

$j$	1	2	3	4	5	6	7
$p_j$	6	5	8	9	6	7	3
$w_j$	2	3	6	7	5	9	4

Den optimale løsning er at vælge genstandene 1 og 4, hvilket giver en optimal løsning på  $z = 15$ .

### Eksempel 2 Dense subgraph-problemet



Givet en komplet vægtet graf  $G = (V, E, c)$  og et heltal  $k$ . *Dense subgraph-problemet* beder os udvælge en delmængde  $U \subseteq V$  af størrelse  $|U| = k$ , således at summen af kantvægte mellem knuder i  $U$  bliver maksimeret. Vi vil tænke på grafen som orienteret, dvs. mellem hvert par af knuder  $i$  og  $j$  findes to kanter, som har vægt  $c_{ij}$  henholdsvis  $c_{ji}$ . Selv om vi tillader, at  $c_{ij} \neq c_{ji}$  så kan man altid opnå en symmetrisk form  $c_{ij} = c_{ji}$  ved at dele summen  $c_{ij} + c_{ji}$  ligeligt på de to kanter (overvej!). Uden tab af generalitet kan vi antage, at alle kantvægte er ikke-negative, dvs.  $c_{ij} \geq 0$ , idet vi ellers kan lægge en konstant  $M$  til alle kantvægte for at opnå dette (se også opgave 4 side 39). I de fleste praktiske anvendelser vil  $c_{ii} = 0$ , men i det følgende kan  $c_{ii}$  også antage positive værdier.

Dense subgraph-problemet er en umiddelbar generalisering af *klike-problemet*, hvorfor det dukker op i mange sammenhænge indenfor grafteori. Praktiske anvendelser omfatter bl.a. lokalisering af sendemaster, således at trafik mellem masterne maksimeres, samt lokalisering af tankstationer, supermarkeder, brandstationer, el-ler hospitaler, således at de spredes mest muligt geografisk.

Kun få eksakte algoritmer er præsenteret for dense subgraph-problemet [8, 20], mens der er udviklet adskillige approximationsalgoritmer [4, 10, 11, 14, 21, 22].

Formelt kan problemet defineres som følgende maksimeringsproblem:

$$z = \max \left\{ \sum_{i \in U} \sum_{j \in U} c_{ij} \mid U \subseteq V, |U| = k \right\}. \quad (4)$$

Den følgende tabel angiver kantvægtene for en graf med 7 knuder, hvori der skal udvælges en delgraf  $U$  af størrelse  $k = 3$ .

$i \setminus j$	1	2	3	4	5	6	7
1	0	3	7	4	10	5	7
2	3	0	9	5	5	10	6
3	7	9	0	1	3	2	4
4	4	5	1	0	1	9	1
5	10	5	3	1	0	3	2
6	5	10	2	9	3	0	3
7	7	6	4	1	2	3	0

Den optimale løsning er at vælge knuderne  $U = \{2, 4, 6\}$ , hvilket giver en løsningsværdi på 48.

### Eksempel 3 Traveling salesman-problemet (minimeringsproblem)

Traveling salesman-problemet har utallige anvendelser, og er blevet studeret indgående i litteraturen, se f.eks. den omfattende bog [15]. Den p.t. bedste algoritme til løsning af Traveling salesman-problemet er *Concorde*-løseren [2] designet af Applegate, Bixby, Chvátal og Cook [3].

Vi vil betragte den *symmetriske* version af traveling salesman-problemet, der formelt kan defineres som følgende optimeringsproblem: Lad  $(V, E, d)$  være en vægтет graf, hvor  $d_{ij}$  for  $(i, j) \in E$  angiver afstanden mellem knuderne  $i$  og  $j$ . Da vi betragter den symmetriske variant gælder at  $d_{ij} = d_{ji}$ . Problemet er da at finde en Hamilton-kreds  $H$  i grafen, som har en minimal længde med hensyn til  $d$ . Hamilton-kredsen  $H$  er en delmængde af kanterne i  $E$ , hvorfor vi kan formulere problemet som:

$$z = \min \left\{ \sum_{(i,j) \in H} d_{ij} \mid H \subseteq E, H \text{ er en Hamilton-kreds} \right\}. \quad (5)$$

På det følgende kort er der markeret otte byer på Bornholm.



Afstanden mellem de otte byer er givet ved følgende tabel, og vi ønsker at finde den korteste Hamilton-kreds gennem byerne.

$i \setminus j$	1	2	3	4	5	6	7	8
1	0	11	24	25	30	29	15	15
2	11	0	13	20	32	37	17	17
3	24	13	0	16	30	39	29	22
4	25	20	16	0	15	23	18	12
5	30	32	30	15	0	9	23	15
6	29	37	39	23	9	0	14	21
7	15	17	29	18	23	14	0	7
8	15	17	22	12	15	21	7	0

Den optimale løsning er at besøge byerne (kuderne) i rækkefølgen:  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 1$ , hvilket giver en samlet længde af Hamilton-kredsen på  $z = 100$ .

## 2 Brute-force metoder

Som tidligere nævnt er branch-and-bound paradigmet en søgebaseret algoritme, som i værste fald gennemløber alle lovlige løsninger. Vi vil i dette afsnit skitsere

en første søgebaseret algoritme samt vurdere dens køretid. Algoritmen udnytter, at ethvert  $\mathcal{NP}$ -problem har et "kort" (dvs. polynomielt) *certifikat*, samt at ethvert certifikat kan *verificeres* i polynomielt tid. Vi kan derfor gennemløbe alle kombinationsmuligheder af certifikatet, og hver gang kontrollere om verifikationsalgoritmen returnerer "ja". Den samlede køretid bliver eksponentiel.

**Definition 1** Klassen  $\mathcal{EXP}$  er mængden af afgørbarhedsproblemer, som kan løses i eksponentiel tid på en deterministisk Turing-maskine. Formelt sagt siger vi, at et afgørbarhedsproblem  $L \in \mathcal{EXP}$ , hvis der findes et polynomium  $p(n)$ , således at enhver streng  $x$  af længde  $n$  kan afgøres i tiden  $2^{p(n)}$ .

**Sætning 1** Hvis  $L \in \mathcal{NP}$ , så gælder også at  $L \in \mathcal{EXP}$ .

**Bevis:** Antag at  $L \in \mathcal{NP}$ , så findes en verifikationsalgoritme  $A(x, y)$ , som kører i polynomielt tid  $p(|x|, |y|)$ . Længden af certifikatet  $y$  skal overholde at  $|y| \leq p_2(|x|)$  for et polynomium  $p_2$ , så køretiden af  $A(x, y)$  må også være polynomielt i  $|x|$ , dvs. begrænset af et polynomium  $p_1(|x|)$ .

Vi konstruerer nu en algoritme, som afgør  $L$  i eksponentiel tid. Uden tab af generalitet kan vi antage, at certifikatet  $y$  er en binær streng. Vi opremsner nu samtlige værdier af  $y$  i tiden  $2^{|y|} \leq 2^{p_2(|x|)}$ . For hver værdi af  $y$  anvender vi verifikationsalgoritmen  $A(x, y)$ . Hvis  $A(x, y) = 1$  for et givet  $y$  returnerer vores algoritme værdien 1. Hvis  $A(x, y) = 0$  for alle  $y$  returnerer algoritmen værdien 0.

Køretiden af algoritmen bliver  $O(2^{p_2(|x|)} p_1(|x|))$ , hvilket viser at algoritmen afgør  $L$  i eksponentiel tid.  $\square$

Løsningsmetoden i ovenstående bevis kaldes *brute-force opremsning* og trods dens dårlige køretid, er den grundlaget for de følgende algoritmer.

Fra beviset bemærker vi også, at løsningsrummet for et  $\mathcal{NP}$ -hårdt optimeringsproblem må være begrænset. Faktisk er det begrænset af  $2^{|y|}$ , hvor  $|y|$  er den binære længde af en løsning.

### 3 Divide and Conquer

Brute-force paradigmet fra forrige afsnit fungerede kun for  $\mathcal{NP}$ -fuldstændige afgørbarhedsproblemer. Når vi skal løse et  $\mathcal{NP}$ -hårdt optimeringsproblem på formlen

$$z = \max\{f(x) \mid x \in S\},$$

har vi ikke en verifikationsalgoritme til rådighed. I stedet kan vi anvende *divide-and-conquer* paradigmet fra Cormen m.fl. [7]. Vi ønsker at opdele problemet i en række mindre problemer, løse de mindre problemer, og samle informationen sammen til en optimal løsning. Der gælder

**Sætning 2** Lad  $S = S_1 \cup \dots \cup S_k$  være en opdeling af  $S$  i mindre mængder, og lad  $z_i = \max\{f(x) \mid x \in S_i\}$  være løsningsværdien svarende til den  $i$ 'te delmængde. Da gælder  $z = \max_{i=1, \dots, k} z_i$ .

**Bevis:**  $z = \max\{f(x) \mid x \in S_1 \cup \dots \cup S_k\} = \max_{i=1, \dots, k} \max\{f(x) \mid x \in S_i\} = \max_{i=1, \dots, k} z_i$ .  $\square$

Vi vil bruge betegnelsen *delproblem* til at betegne vores optimeringsproblem begrænset til delmængde  $S_i$ . I det følgende vil vi sprogligt ikke skelne mellem en delmængde og det tilhørende delproblem.

**Eksempel 4** *Knapsack-problemet*

Af pladshensyn vil vi betragte en reduceret udgave af knapsack-problemet fra eksempel 1, hvor der kun er  $n = 3$  genstande, og kapaciteten er  $c = 6$ .

$j$	1	2	3
$p_j$	6	8	3
$w_j$	2	6	4

Løsningsrummet er  $S = \{(x_1, \dots, x_n) \mid \sum_{j=1}^n w_j x_j \leq c, x_j \in \{0, 1\}\}$ .

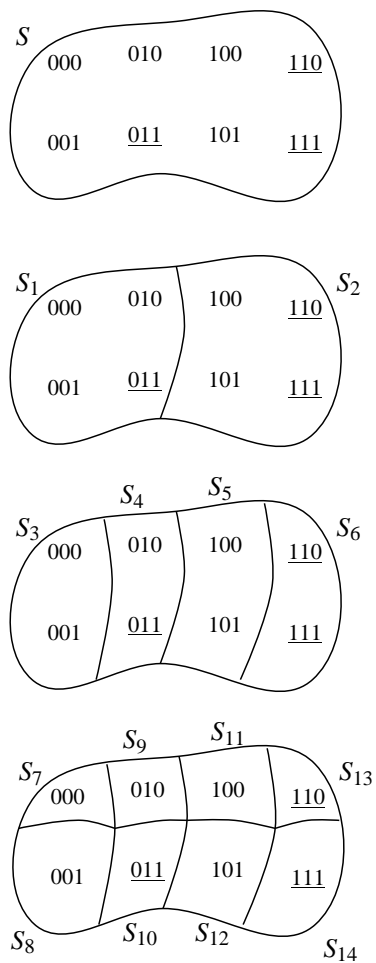
På figur 1 ses en opdeling af løsningsrummet  $S$  svarende til, om en given beslutningsvariabel  $x_i$  sættes til 0 eller 1. I den første opdeling får vi to delproblemer:

$$\begin{aligned} S_1 &= \{(x_1, \dots, x_n) \subseteq S \mid x_1 = 0\}, \\ S_2 &= \{(x_1, \dots, x_n) \subseteq S \mid x_1 = 1\}. \end{aligned} \tag{6}$$

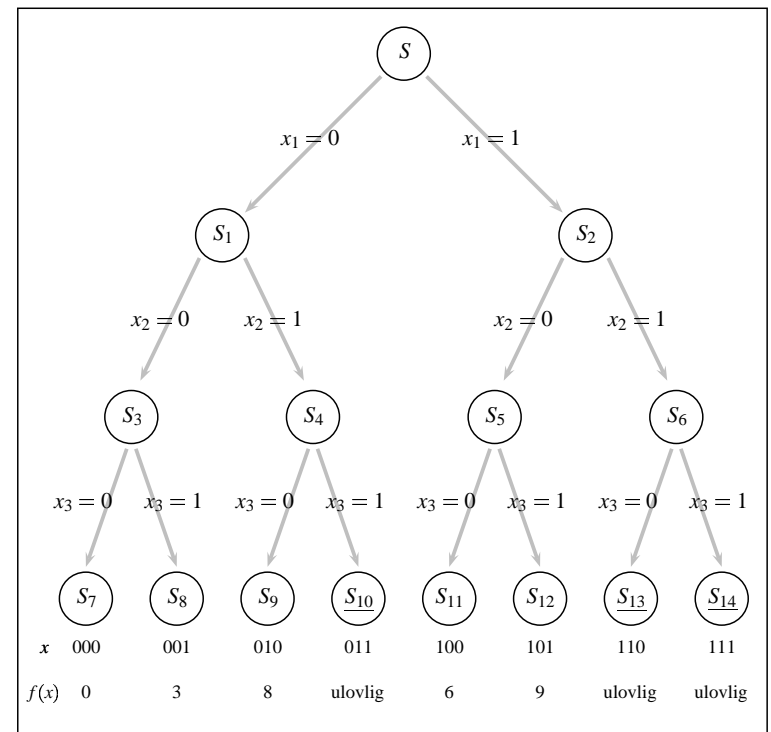
Tilsvarende får vi ved opdeling af  $S_1$  de to mængder  $S_3$  og  $S_4$  givet ved

$$\begin{aligned} S_3 &= \{(x_1, \dots, x_n) \subseteq S_1 \mid x_1 = 0, x_2 = 0\}, \\ S_4 &= \{(x_1, \dots, x_n) \subseteq S_1 \mid x_1 = 0, x_2 = 1\}. \end{aligned} \tag{7}$$

Og så fremdeles.



Figur 1: Opdeling af løsningsrum. De understregede tal svarer til ulovlige løsninger, dvs. løsningsvektorer  $x$  hvor  $\sum_{j=1}^n w_j x_j > c$ .



Figur 2: Søgetræ for knapsack-problemet. De understregede mængder angiver ulovlige løsninger, dvs. løsningsvektorer  $x$  hvor  $\sum_{j=1}^n w_j x_j > c$ .

Sætning 2 viser, at vi kan finde en optimal løsning til f.eks.  $S_5$  som  $\max\{z_{11}, z_{12}\} = \max\{6, 9\} = 9$ , hvor  $z_{11}$  er den optimale løsning til  $S_{11}$ , og  $z_{12}$  er løsningen til  $S_{12}$ .

Man kan illustrere opdelingen af  $S$  med et søgetræ, som vist i figur 2. Hver knude i søgetræet svarer til et delproblem  $S_i$ . Hvis vi betragter to knuder  $i$  og  $j$  i søgetræet, hvor  $j$  ligger under  $i$ , så kalder vi  $j$  for et *underproblem* til  $i$ .

## 4 Grænseværdier

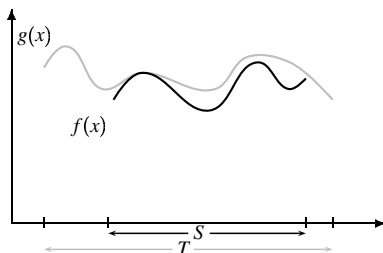
Vi betragter igen et optimeringsproblem  $P$  på formen  $z = \max\{f(x) \mid x \in S\}$ .  $S$  behøver ikke at være det originale løsningsrum for et problem, men kan godt være et delproblem  $S_i$  fremkommet ved brug af divide-and-conquer paradigmet. En *nedre grænseværdi*  $L$  er et reelt tal, som overholder at  $L \leq z$ . På samme måde kan vi definere en *øvre grænseværdi*  $U$  som et reelt tal, der overholder at  $U \geq z$ .

Enhver lovlig løsning  $x' \in S$  til problemet  $P$  er en nedre grænseværdi, idet der oplagt gælder at  $f(x') \leq z = \max\{f(x) \mid x \in S\}$ . Det er anderledes udfordrende at finde en øvre grænseværdi. Til dette formål har vi brug for at betragte en relaxering:

**Definition 2** Givet et problem  $P$  defineret som  $z = \max\{f(x) \mid x \in S\}$ . Problemet  $R$  givet ved  $z_R = \max\{g(x) \mid x \in T\}$  er en relaxering af  $P$ , hvis der gælder:

- (i)  $S \subseteq T$ ,
- (ii)  $g(x) \geq f(x)$  for alle  $x \in S$ .

Følgende figur illustrerer princippet i en relaxering. Vi maksimerer funktionen  $f$  over mængden  $S$ . Funktionen  $g$  må ikke ligge under  $f$  i hele definitionsmængden  $S$ , men der stilles ingen krav til  $g$  udenfor  $S$ .



**Sætning 3** Hvis  $R$  er en relaxering af  $P$ , så gælder der at  $z_R \geq z$ .

**Bevis:** Antag, at den optimale løsning for  $P$  er  $x^*$ , dvs. der gælder  $f(x^*) = z$ . Da  $x^* \in S$  har vi fra (ii) at  $g(x^*) \geq f(x^*)$ . Fra (i) ved vi endvidere at  $x^* \in T$ . Dermed må gælde at  $z_R = \max\{g(x) \mid x \in T\} \geq g(x^*) \geq f(x^*) = z$ .  $\square$

Ovenstående sætning giver os en opskrift til at bestemme øvre grænseværdier, idet vi for et problem  $P$  kan løse en tilhørende relaxering  $R$  og bestemme  $z_R$ . Da gælder at  $U := z_R$  er en øvre grænseværdi, idet vi har  $z_R \geq z = \max\{f(x) \mid x \in S\} \geq f(x)$  for ethvert  $x \in S$ .

Man kan altid finde en triviel relaxering ved at vælge  $g(x) = f(x)$  og  $T = S$ . Denne relaxering vil returnere den bedst tænkelige grænseværdi  $U = z$ , men den er lige så dyr at beregne, som at løse det originale problem. Man vil derfor normalt kræve, at en relaxering kan løses i polynomiel tid. Det er måske ikke videre intuitivt, at et problem kan blive lettere at løse ved at udvide løsningsrummet  $S$  til en større mængde  $T$ , men nedenstående eksempler vil vise, at dette rent faktisk ofte er tilfældet. Tilsvarende kan nogle problemer blive lettere at løse ved at modificere objektfunktionen.

For ethvert problem  $P$  vil der kunne defineres mange relaxeringer, som vil resultere i et antal forskellige grænseværdier. Det er en kunst at finde en relaxering, som giver gode grænseværdier, og som kan beregnes effektivt.

Divide-and-conquer paradigmet fra afsnit 3 kan også anvendes til at bestemme grænseværdier ved opdeling af problemet i mindre delproblemer. Idet vi definerer  $z_i = \max\{f(x) \mid x \in S_i\}$ , gælder følgende to sætninger:

**Sætning 4** Lad  $S = S_1 \cup \dots \cup S_k$  være en opdeling af  $S$  i mindre mængder, og lad  $L_i$  være en nedre grænseværdi for delmængde  $S_i$ . Da gælder at  $L = \max_{i=1, \dots, k} L_i$  er en nedre grænseværdi for  $S$ .

**Bevis:** Da  $L_i \leq z_i$  har vi  $L = \max_{i=1, \dots, k} L_i \leq \max_{i=1, \dots, k} z_i = z$ .  $\square$

**Sætning 5** Lad  $S = S_1 \cup \dots \cup S_k$  være en opdeling af  $S$  i mindre mængder, og lad  $U_i$  være en øvre grænseværdi for delmængde  $S_i$ . Da gælder at  $U = \max_{i=1, \dots, k} U_i$  er en øvre grænseværdi for  $S$ .

**Bevis:** Da  $z_i \leq \mathcal{U}_i$  har vi  $\mathcal{U} = \max_{i=1, \dots, k} \mathcal{U}_i \geq \max_{i=1, \dots, k} z_i = z$ .  $\square$

Endelig har vi sætningen:

**Sætning 6** *Betragt et maksimeringsproblem  $z = \max\{f(x) \mid x \in S\}$ , hvor  $f(x)$  returnerer en heltallig løsningsværdi for ethvert  $x \in S$ . Antag at  $\mathcal{U}$  er en øvre grænseværdi. Da er også  $\lfloor \mathcal{U} \rfloor$  en øvre grænseværdi.*

**Bevis:** Da  $f(x)$  er heltallig, må  $z \in \mathbb{Z}$ , og følgelig  $z = \lfloor z \rfloor$ . Da  $z \leq \mathcal{U}$  gælder  $\lfloor z \rfloor \leq \lfloor \mathcal{U} \rfloor$ , og dermed  $z \leq \lfloor \mathcal{U} \rfloor$ .  $\square$

## 4.1 Eksempler på grænseværdier

### Eksempel 5 Knapsack-problemet

For at finde øvre grænseværdier for knapsack-problemet betragter vi en simpel relaxering, hvor det er tilladt at medtage brøkdeler af genstandene. Dette problem kaldes det *fraktionelle knapsack-problem* i Cormen m.fl. [7], givet ved

$$\mathcal{U}_{\text{kp}}^1 = \max \left\{ \sum_{j=1}^n p_j x_j \mid \sum_{j=1}^n w_j x_j \leq c, 0 \leq x_j \leq 1 \right\}. \quad (8)$$

For at se, at der er tale om en relaxering, bemærker vi først at knapsack-problemet er defineret på formen (1) med

$$f(x) = \sum_{j=1}^n p_j x_j \quad , \quad S = \left\{ (x_1, \dots, x_n) \mid \sum_{j=1}^n w_j x_j \leq c, x_j \in \{0, 1\} \right\}.$$

Det fraktionelle knapsack-problem er defineret på samme form med

$$g(x) = \sum_{j=1}^n p_j x_j \quad , \quad T = \left\{ (x_1, \dots, x_n) \mid \sum_{j=1}^n w_j x_j \leq c, 0 \leq x_j \leq 1 \right\}.$$

Da  $f(x) = g(x)$  er kriterium (ii) i definition 2 overholdt. Endvidere er  $S \subseteq T$ , hvorfor kriterium (i) også er opfyldt.

Det fraktionelle knapsack-problem kan løses i polynomiel tid ved brug af den grådige algoritme, som beskrevet i Cormen m.fl. [7] s. 382. Først sorteres genstandene efter aftagende *effektivitet*  $p_j/w_j$ , således at

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \frac{p_3}{w_3} \geq \dots \geq \frac{p_n}{w_n}, \quad (9)$$

hvorpå rygsækken fyldes som følger: Genstandene 1, 2, 3, ... lægges i rygsækken, indtil man støder på den første genstand  $b$ , som der ikke er plads til. Den optimale løsning er da at medtage de første  $b-1$  genstande (dvs.  $x_j = 1$  for  $j = 1, \dots, b-1$ ), mens en brøkdel af genstand  $b$  medtages, således at hele kapaciteten udnyttes:

$$x_b = \frac{c - \sum_{j=1}^{b-1} w_j}{w_b}.$$

Ingen af genstandene efter  $b$  medtages (dvs.  $x_j = 0$  for  $j = b+1, \dots, n$ ). Dette giver os følgende direkte formel til at bestemme den øvre grænseværdi

$$\mathcal{U}_{\text{kp}}^1 = \sum_{j=1}^{b-1} p_j + p_b \frac{c - \sum_{j=1}^{b-1} w_j}{w_b}. \quad (10)$$

Grænseværdien kan findes i  $O(n \log n)$  tid, hvor den tungeste beregning er sorteringen (9).

Den grådige algoritme kan også bruges til at finde en nedre grænseværdi  $\mathcal{L}$ . Antag, at genstandene er sorteret efter (9) og betragt genstandene i rækkefølgen 1, 2, 3, ... Hvis der stadig er plads til en given genstand  $i$  medtages den i rygsækken, ellers fortsættes med genstand  $i+1$ .

For knapsack-problemet fra eksempel 1 bemærker vi, at genstandene allerede er sorteret efter aftagende profit-vægt forhold.

$j$	1	2	3	4	5	6	7
$p_j$	6	5	8	9	6	7	3
$w_j$	2	3	6	7	5	9	4

Da rygsækken har kapacitet  $c = 9$  fylder vi genstande 1, 2 i rygsækken og medtager en fraktionel del af genstand  $b = 3$ . Beslutningsvariablene bliver  $x_1 = x_2 = 1$ , og  $x_3 = \frac{9-5}{6} = \frac{2}{3}$ . Den øvre grænseværdi bliver

$$\mathcal{U}_{\text{kp}}^1 = 11 + 8 \cdot \frac{2}{3} = 16\frac{1}{3},$$

som ifølge sætning 6 kan rundes ned til  $\mathcal{U}_{\text{kp}}^1 = 16$ .

En nedre grænseværdi findes ved at vælge genstande 1, 2, 7, der giver værdien  $\mathcal{L} = 14$ .

**Eksempel 6** Dens subgraph-problemet

Ved at sætte en parentes i objektfunktionen kan problemet skrives som

$$z = \max \left\{ \sum_{i \in U} \left( \sum_{j \in U} c_{ij} \right) \mid U \subseteq V, |U| = k \right\}. \quad (11)$$

Lad  $\bar{c}_i$  være en øvre grænse på enhver kantvægt-sum, der kan udgå fra knude  $i$ . Formelt kan dette defineres som

$$\bar{c}_i = \max \left\{ \sum_{j \in U} c_{ij} \mid U \subseteq V, |U| = k \right\}. \quad (12)$$

Da kan udtrykket inde i parentesen af (11) begrænses opadtil ved

$$\sum_{j \in U} c_{ij} \leq \bar{c}_i, \quad (13)$$

og vi får følgende øvre grænseværdi for dense subgraph-problemet

$$z_{\text{dsp}}^1 = \max \left\{ \sum_{i \in U} \bar{c}_i \mid U \subseteq V, |U| = k \right\}. \quad (14)$$

For at indse, at der er tale om en relaxering af det originale problem, bemærker vi, at det originale problem havde objektfunktion og løsningsrum

$$f(x) = \sum_{i \in U} \left( \sum_{j \in U} c_{ij} \right) \quad , \quad S = \{U \subseteq V, |U| = k\}.$$

Det nye problem har objektfunktion og løsningsrum

$$g(x) = \sum_{i \in U} \bar{c}_i \quad , \quad T = \{U \subseteq V, |U| = k\}.$$

Da  $S = T$  er kriterium (i) i definition 2 opfyldt. Tilsvarende har vi kriterie (ii) opfyldt, da  $g(x) \geq f(x)$  for alle  $x \in S$  på grund af (13).

Det relaxerede problem (14) kan løses i  $O(|V|^2)$  tid. Først finder vi for hver knude  $i \in V$  værdien  $\bar{c}_i$  givet ved (12). Dette problem består i at vælge de  $k$  største kantvægte, som udgår fra knude  $i$ . Dette kan gøres i  $O(|V|)$  tid for hver knude  $i$ , jf. Problem 9-1 side 194 i Cormen m.fl. [7]. I det relaxerede problem (14) skal vi igen vælge de  $k$  største tal blandt tallene  $\bar{c}_1, \bar{c}_2, \dots, \bar{c}_n$ , hvilket kan gøres i  $O(|V|)$  tid. Samlet får vi køretiden  $|V|O(|V|) + O(|V|)$ .

Som eksempel på grænseværdiberegningen kan vi betragte instansen fra eksempel 2:

$i \setminus j$	1	2	3	4	5	6	7
1	0	3	7	4	10	5	7
2	3	0	9	5	5	10	6
3	7	9	0	1	3	2	4
4	4	5	1	0	1	9	1
5	10	5	3	1	0	3	2
6	5	10	2	9	3	0	3
7	7	6	4	1	2	3	0

Her har vi  $n = 7$  knuder, hvoraf  $k = 3$  skal udvælges. Ved at vælge de  $k$  største tal i hver række finder vi  $\bar{c}_1 = 24, \bar{c}_2 = 25, \bar{c}_3 = 20, \bar{c}_4 = 18, \bar{c}_5 = 18, \bar{c}_6 = 24$  og  $\bar{c}_7 = 17$ . Dermed bliver en øvre grænseværdi  $z_{\text{dsp}}^1 = 73$ .

I opgave 9 vises en strammere version af oventående grænseværdi.

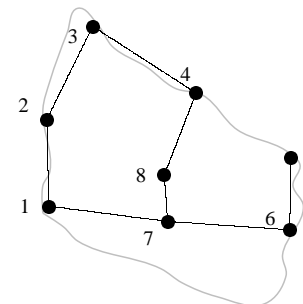
**Eksempel 7** Traveling salesman-problemet (minimeringsproblem)

Der er gennem tiden foreslået mange grænseværdier for traveling salesman-problemet.

En af de kønneste er baseret på 1-træ relaxering. Givet en vægteset graf  $(V, E, d)$  fremkommer et 1-træ ved at finde et udspændende træ på knuderne  $2, 3, \dots, n$  og herefter forbinde knude 1 med to vilkårlige kanter.

Et minimalt 1-træ findes ved at løse et mindste udspændende træ på knuderne  $2, 3, \dots, n$ , og herefter udvælge de to billigste kanter, som udgår fra knude 1 (overvej, hvorfor dette er et minimalt 1-træ).

For Bornholm-instansen fra eksempel 3 konstrueres et minimalt 1-træ ved at finde et mindste udspændende træ på knuderne  $2, 3, \dots, 8$ . De to billigste kanter, som udgår fra knude 1 er  $(1, 2)$  og  $(1, 7)$ , som tilføjes kantmængden. Samlet har vi følgende 1-træ, som koster 97 enheder:



Køretiden for at finde et mindste 1-træ domineres af, at vi skal finde et *mindste udspændende træ* (MST). Dette kan gøres i tiden  $O(|E|\log|V|)$  med Kruskals algoritme, jf. Cormen m.fl. [7] afsnit 23.2 side 568. Alternativt kan man bruge Prims algoritme, som har køretiden  $O(|E| + |V|\log|V|)$ , hvis man benytter Fibonacci hobe.

Man kan bruge 1-træer til at finde en nedre grænseværdi for traveling salesman-problemet givet ved

$$\mathcal{L}_{\text{TSP}}^1 = \min \left\{ \sum_{(i,j) \in H} d_{ij} \mid H \subseteq E, H \text{ er et 1-træ} \right\}. \quad (15)$$

For at indse at et minimalt 1-træ er en relaxering, bemærker vi, at det originale problem kan skrives på formen (1) med

$$f(x) = \sum_{(i,j) \in H} d_{ij}, \quad S = \left\{ H \subseteq E \mid H \text{ er en Hamilton-kreds} \right\}. \quad (16)$$

Tilsvarende har vi for problem (15) at

$$g(x) = \sum_{(i,j) \in H} d_{ij}, \quad T = \left\{ H \subseteq E \mid H \text{ er et 1-træ} \right\}.$$

Da både traveling salesman-problemet og det mindste 1-træ har samme objekt-funktion er kriterium (ii) opfyldt. Endvidere har vi, at en Hamilton-kreds også er et 1-træ, hvilket ses ved at fjerne knude 1 og bemærke, at de tilbageværende kanter udgør et træ (faktisk udgør de en vej, der naturligvis også er et træ). Da  $S \subseteq T$  har vi vist kriterium (i).

## 4.2 Grænseværditest

I forrige afsnit definerede vi en grænseværdifunktion som en funktion  $f$ , der for ethvert delproblem  $S_i$  returnerer et reelt tal  $\mathcal{U}$ , således at  $f(x) \leq \mathcal{U}$  for alle  $x \in S_i$ . Sagt i ord giver grænseværdifunktionen en garanti for, at man ikke kan finde en løsning i  $S_i$ , som er bedre end  $\mathcal{U}$ . Man kan bruge denne viden konstruktivt til at bortskære delproblemer. Hvis man for et delproblem  $S_i$  ved, at man ikke kan finde en bedre løsning end den nuværende nedre grænseværdi, så er der ingen grund til at undersøge delproblemet nærmere.

Lidt mere formelt har vi følgende *grænseværditest*:

**Sætning 7** Hvis et delproblem  $S_i$  har grænseværdi  $\mathcal{U}$ , således at

$$\mathcal{U}(S_i) \leq \mathcal{L}, \quad (17)$$

så kan vi forkaste  $S_i$ .

**Bevis:** Grænseværdien sikrer, at  $f(x) \leq \mathcal{U} \leq \mathcal{L}$  for alle  $x \in S_i$ . Så ingen løsning  $x$  i  $S_i$  vil have en bedre løsningsværdi end den allerede kendte  $\mathcal{L}$ .  $\square$

## 4.3 Monotonitet af grænseværdier

En øvre grænseværdifunktion siges at være *monoton*, hvis der gælder at  $\mathcal{U}(S_1) \leq \mathcal{U}(S_2)$ , når  $S_1 \subseteq S_2$ . Løst sagt skal grænseværdien blive mindre, når vi maksimerer over et mindre løsningsrum. Dette gælder oplagt for alle relaxeringer, hvor  $S = T$  i definition 2, og det vil også gælde for de fleste fornuftige relaxeringer, hvor  $S \neq T$ .

Hvis en grænseværdifunktion er monoton, vil den øvre grænseværdi aftage efterhånden, som vi bevæger os ned i søgetræet af delproblemer. Dette hænger sammen med den trivielle observation, at vi hver gang opdeler problemet i mindre delproblemer, således at ovenstående kriterium er opfyldt for en knude og dens underliggende knude.

## 5 Branch-and-bound

Vi er nu i stand til at skitsere en generisk branch-and-bound algoritme for et maksimeringsproblem på formen

$$z = \max\{f(x) \mid x \in S\}. \quad (18)$$

I den nedenstående algoritme angiver  $L$  en liste af delproblemer  $S_i$  givet ved de tilhørende løsningsrum. Listen  $L$  vil ofte være organiseret som en prioritetskø. Vi kalder delproblemerne i  $L$  for *åbne delproblemer*, mens delproblemer, der allerede er blevet behandlet, kaldes *lukkede delproblemer*. Endvidere vedligeholder vi en global nedre grænseværdi  $\mathcal{L}$  samt den tilhørende løsning  $x^*$ .

```

1   $\mathcal{L} := -\infty; L := \{S\}$ 
2  while  $L \neq \emptyset$ 
3    vælg et delproblem  $S_i$  fra  $L$ 
4     $L := L \setminus \{S_i\}$ 
5    if  $S_i \neq \emptyset$  then
6      find en øvre grænseværdi  $\mathcal{U}(S_i)$ 
7      if  $\mathcal{U}(S_i) > \mathcal{L}$  then
8        find en lovlig løsning  $x \in S_i$ 
9        if  $f(x) > \mathcal{L}$  then  $\mathcal{L} := f(x); x^* := x$ 
10       opdel  $S_i$  i delproblemer  $S_i^1, \dots, S_i^k$ 
11       tilføj delproblemerne til  $L$ , dvs. sæt  $L := L \cup \{S_i^1, \dots, S_i^k\}$ 
12     endif
13   endif
14 endwhile

```

En branch-and-bound algoritme for et maksimeringsproblem vil derfor bestå af følgende fire komponenter:

1. En *øvre grænseværdifunktion*, der for et givet delproblem returnerer en øvre grænse på værdien af den bedste løsning, som vi kan finde i delrummet.
2. En *nedre grænseværdi*, som på ethvert tidspunkt angiver den hidtil bedst kendte løsning.
3. En *søgestrategi*, som fastlægger en rækkefølge for behandlingen af delproblemer.
4. En *forgreningsregel*, som for alle delproblemer, der ikke kan forkastes af grænseværditesten, angiver, hvorledes det tilhørende løsningsrum skal opdeles. Dermed skabes to eller flere nye delproblemer.

## 5.1 Øvre grænseværdifunktion

En god grænseværdifunktion er afgørende for, at en branch-and-bound algoritme bliver vellykket. Jo strammere grænseværdier vi kan få, des mindre bliver søgetræet. Som beskrevet i afsnit 4 findes grænseværdier ved *relaksering* af optimeringsproblemet. Der findes en række standardteknikker, som kan benyttes i

denne sammenhæng, f.eks. *lineær-relaksering*, *Lagrange-relaksering*, *surrogat-relaksering* og *semidefinit-relaksering*. Det ville være for vidtgående at beskrive de nævnte teknikker i denne introduktion, men f.eks. Wolsey [24] beskriver mange af dem. Se dog opgave 7 for et eksempel på Lagrange-relaksering, og bemærk at lineær-relaksering blev benyttet i eksempel 5.

## 5.2 Nedre grænseværdi

Jo bedre nedre grænseværdi vi kan finde, des flere delproblemer kan vi bortskære, og des hurtigere vil algoritmen terminere. Der findes en række metoder til opdatering af den nedre grænseværdi:

- Den simpleste metode er at lade branch-and-bound algoritmen kontrollere om det nuværende delproblem  $S_i$  kun indeholder en enkelt løsning  $x$ . I så fald er det relativt simpelt at kontrollere om løsningen er lovlig, samt om den tilhørende løsningsværdi er bedre end  $\mathcal{L}$ . I sidstnævnte tilfælde opdateres  $\mathcal{L}$ .
- I branch-and-bound algoritmen skal vi udregne en øvre grænseværdi for hvert delproblem. Denne udregnes ved at løse et relakeret problem. Ofte kan det relakerede problem omformes til en lovlig løsning for det originale problem.
- For hvert delproblem kan man bruge en heuristik til at finde en lovlig løsning. Denne kan være en *grådig algoritme*, eller den kan baseres på andre tilsvarende principper.
- Inden branch-and-bound algoritmen udføres, kan man bruge en del kræfter på at finde en god initial nedre grænseværdi. Da dette kun skal gøres een gang, behøver algoritmen ikke at være specielt hurtig, om end man ofte vil forvente at den er polynomiel.

## 5.3 Søgestrategi

Som tidligere nævnt angiver søgestrategien et kriterium for hvilket delproblem  $S_i$ , der er det næste, vi skal betragte i branch-and-bound algoritmen.

Man kan umiddelbart forestille sig en række forskellige hensigtsmæssige strategier: F.eks. kunne det være en fordel hurtigt at finde en lovlig løsning, da værdien

af denne efterfølgende kan bruges som nedre grænseværdi  $L$ . Man kunne også vælge at betragte de delproblemer først, som ser mest lovende ud. Til vurdering af hvilke delproblemer, der er mest lovende, kan man bruge den øvre grænseværdi. Endelig kunne man forestille sig, at man gerne vil holde antallet af åbne delproblemer nede. Derfor kunne en strategi være først at vælge de dårligste delproblemer (målt vha. deres øvre grænseværdi), idet disse formentlig hurtigt kan bortskæres med en grænseværditest.

Mere formelt har vi:

1. *Dybde-først-søgning*. Her vælges altid en knude på laveste niveau i søgetræet. Da bladknuderne i søgetræet (uden bortskæring med grænseværditest) svarer til lovlige løsninger, vil denne strategi hurtigt føre os til nogle lovlige løsninger. Dette giver os en nedre grænseværdi  $L$ , som vi kan bruge til at bortskære knuder med. Strategien er endvidere pladsmæssigt fordelagtig. Hvis vi antager, at hvert delproblem højst forgrenes i et konstant antal nye delproblemer, så vil der aldrig være mere end  $O(m)$  åbne delproblemer, når søgetræet har højde  $m$ .
2. *Bedste-først-søgning*. I denne søgestrategi betragter man de delproblemer først, som har den største øvre grænseværdi. Man håber her på at delproblemer med en god øvre grænseværdi også rummer de gode løsninger. Dermed vil vi formentlig finde en nedre grænseværdi  $L$  af god kvalitet. Bedste-først-søgning har endvidere den fordel, at vi betragter det mindst mulige søgetræ (se sætning 9 side 35).  
For at bruge bedste-først-søgning skal listen  $L$  af åbne delproblemer udvides til par af formen  $(S_i, \mathcal{U}_i)$ , hvor  $\mathcal{U}_i$  er en øvre grænseværdi for  $S_i$ . De øvre grænseværdier kan beregnes umiddelbart efter opdelingen af  $S_i$  i delproblemer (linie 10).
3. *Bredde-først-søgning*. Her vil man behandle alle delproblemer på et givet niveau i søgetræet, inden man fortsætter til næste niveau i søgetræet. Sagt med andre ord vil bredde-først-søgning udvælge det åbne delproblem som ligger højest i søgetræet. Argumentet for at bruge bredde-først-søgning kunne f.eks. være, at man ønsker samlet at behandle delproblemer, som har den samme mængde variable sat til en værdi. Ulempen ved bredde-først-søgning er, at søgetræet kan blive eksponentielt stort. For nogle problemer vil bredde-først-søgning svare til en slags dynamisk programmeringsalgoritme, som illustreret i opgave 3.

4. *Heuristisk styret søgning*. Hvis man kender en god heuristik (f.eks. en grådige algoritme) for det givne problem, kan det være hensigtsmæssigt at basere søgestrategien på denne algoritme. Branch-and-bound algoritmen vil da blive styret mod løsninger af god kvalitet, hvilket kan bruges til at opnå en god nedre grænseværdi  $L$ . Heuristisk styret søgning kan kombineres med alle ovenstående søgestrategier, idet man blandt ligeværdige kandidater vælger den knude som heuristisk set ser mest lovende ud.

## 5.4 Forgreningsregel

Forgreningsreglen definerer, hvorledes et delproblem  $S_i$  skal opdeles i et antal mindre delproblemer  $S_i^1, \dots, S_i^k$ . Forgreningsreglen vil typisk være problemspecifik, men ved valg af opdeling skal man tage følgende i betragtning:

- Forgrening sker ofte ved tilføjelse af ekstra begrænsninger. F.eks. kan en variabel sættes til en given værdi, en kant medtages eller fjernes, eller man kan tilføje en generel ulighed til problemet. Det er vigtigt at de ekstra begrænsninger ikke ændrer problemets natur for meget, idet man ellers skal udvikle helt nye grænseværdier for delproblemerne.
- Det er sædvanligvis ikke nogen god ide at opdele  $S_i$  i alt for mange delproblemer  $S_i^1, \dots, S_i^k$ . Hvis man i sit problem har en variabel  $x_i \in \{0, 1, \dots, 100\}$  vil det næppe være nogen god ide at generere et delproblem for hver af de ekstra begrænsninger  $x_i = 0, x_i = 1, x_i = 2, \dots, x_i = 100$ . En binær forgrening efter  $x_i < 50$ , og  $x_i \geq 50$  ville være bedre.
- Ved opdeling i delproblemer skal man tilstræbe at søgetræet vokser symmetrisk. F.eks. hvis det ene delproblem kun indeholder en enkelt løsning, eller det ene delproblem umiddelbart kan bortskæres vha. en grænseværditest, så har man reelt set ikke fået opdelt kernen af svære delproblemer.

## 5.5 Eksempler på branch-and-bound algoritmer

### Eksempel 8 Knapsack-problemet

Den simpleste måde at implementere en branch-and-bound algoritme på er ved brug af dybde-først-søgning, idet algoritmen da kan formuleres som en rekursiv procedure, hvor stakken bruges til at lagre alle delproblemer i  $L$ .

Vi vil illustrere dette princip for knapsack-problemet. Vi anvender heuristisk styret søgning med den grådige strategi som udvælgelseskriterie. Derfor sorteres alle genstande efter aftagende profit-vægt forhold (9) inden udførelsen af branch-and-bound-delen, således at vi først forgrener på den genstand, som har størst profit-vægt forhold. I tråd med den grådige strategi vælger vi altid at behandle delproblemet med  $x_i = 1$  inden delproblemet med  $x_i = 0$ .

Den rekursive branch-and-bound algoritme for knapsack-problemet bliver da:

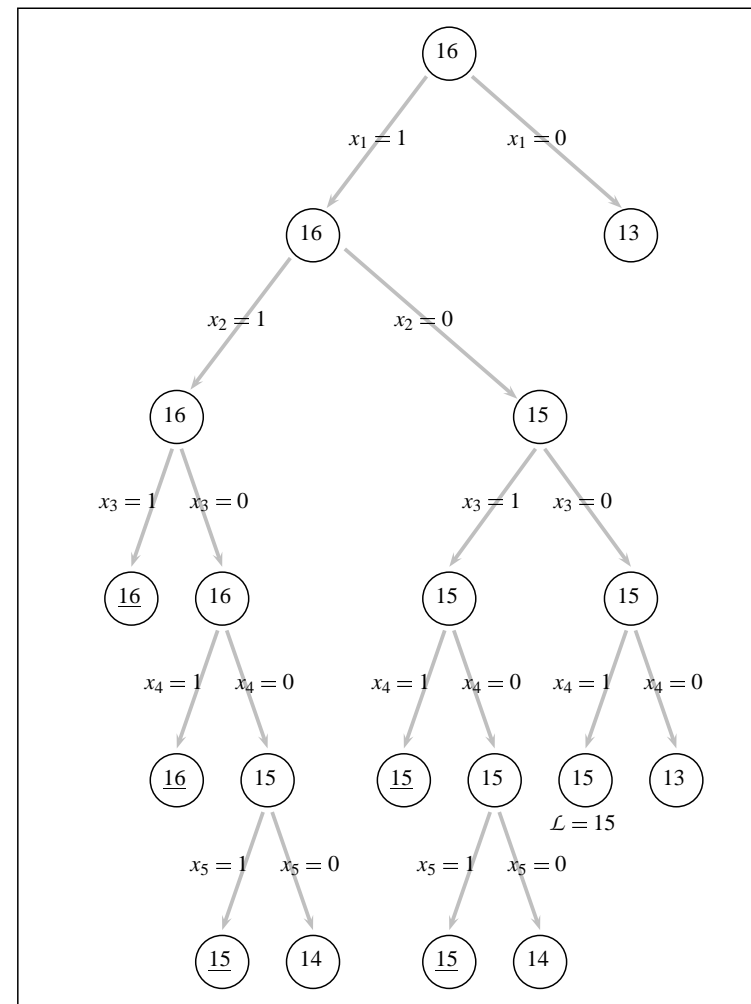
```

BRANCHBOUNDKNAPSACK( $\bar{p}, \bar{w}, i$ )
if  $\bar{w} > c$  then return
udregn  $\mathcal{U}_{kp}^1$  defineret på genstande  $\{i, \dots, n\}$  og med kapacitet  $c - \bar{w}$ .
en øvre grænseværdi for det betragtede delproblem er  $\mathcal{U} = \bar{p} + \lfloor \mathcal{U}_{kp}^1 \rfloor$ .
if  $\mathcal{U} > \mathcal{L}$  then
  if  $\bar{p} > \mathcal{L}$  then  $\mathcal{L} = \bar{p}$ ;  $x^* := x$ 
   $x_i := 1$ ; BRANCHBOUNDKNAPSACK( $\bar{p} + p_i, \bar{w} + w_i, i + 1$ )
   $x_i := 0$ ; BRANCHBOUNDKNAPSACK( $\bar{p}, \bar{w}, i + 1$ )
endif
  
```

Ved hvert kald til BRANCHBOUNDKNAPSACK er  $i$  den næste genstand, vi vil forgrene på, mens  $\bar{p} = \sum_{j=1}^{i-1} p_j x_j$  er profitsummen af de allerede valgte genstande, og  $\bar{w} = \sum_{j=1}^{i-1} w_j x_j$  er vægtsummen af de valgte genstande. Inden algoritmen køres skal man sætte  $x = (0, 0, \dots, 0)$ . Den nedre grænseværdi  $\mathcal{L}$  kan initielt sættes til 0, eller man kan benytte den grådige løsning fra eksempel 5. Herefter startes branch-and-bound algoritmen med kaldet

BRANCHBOUNDKNAPSACK(0, 0, 1).

Hvis vi anvender ovenstående branch-and-bound algoritme på instansen fra eksempel 1, fremkommer søgetræet vist på figur 3. Træet gennemløbes dybde-først fra venstre mod højre. I hver knude angiver tallet den pågældende øvre grænseværdi  $\mathcal{U}$ . Branch-and-bound algoritmen begynder med en initial nedre grænseværdi  $\mathcal{L} = 14$ , der er fundet med den grådige algoritme. På figuren er markeret, hvor den nedre grænseværdi opdateres til  $\mathcal{L} = 15$ . De understregede tal svarer til ulovlige løsninger, dvs. løsningsvektorer  $x$  hvor  $\sum_{j=1}^n w_j x_j > c$ .



Figur 3: Branch-and-bound træ for knapsack-problemet

### Eksempel 9 Dens subgraph-problemet

For at vælge en forgreningsstrategi for dense subgraph-problemet kan man tage udgangspunkt i grænseværdien fra eksempel 6. Værdierne  $\bar{c}_i$  rummer en øvre grænseværdi for, hvor stor kantvægt-summen fra knude  $i$  kan blive. Denne øvre grænseværdi kan bruges til at designe en heuristisk baseret forgreningsstrategi. Vi kan antage, at knuder med store værdier af  $\bar{c}_i$  er attraktive, hvorfor vi først forgrener på disse knuder, hvor det første delproblem vælger knuden, mens det andet delproblem udelader knuden. Hvis vi følger det grådige princip, betragter vi først delproblemet, hvor en knude vælges.

$i \setminus j$	1	2	3	4	5	6	7
1	0	3	7	4	10	5	7
2	3	0	9	5	5	10	6
3	7	9	0	1	3	2	4
4	4	5	1	0	1	9	1
5	10	5	3	1	0	3	2
6	5	10	2	9	3	0	3
7	7	6	4	1	2	3	0

$V_3 \notin U$

$V_3 \in U$

$i \setminus j$	1	2	3	4	5	6	7
1	0	3	7	4	10	5	7
2	3	0	9	5	5	10	6
3	7	9	0	1	3	2	4
4	4	5	1	0	1	9	1
5	10	5	3	1	0	3	2
6	5	10	2	9	3	0	3
7	7	6	4	1	2	3	0

$i \setminus j$	1	2	3	4	5	6	7
1	14	3	7	4	10	5	7
2	3	18	9	5	5	10	6
3	7	9	0	1	3	2	4
4	4	5	1	2	1	9	1
5	10	5	3	1	6	3	2
6	5	10	2	9	3	4	3
7	7	6	4	1	2	3	8

Når en knude  $i$  forbydes i grafen, kan vi blot slette den tilhørende række og søjle. Når en knude  $i$  vælges i grafen, skal vi slette den tilhørende række og søjle i kantmatricen og samtidig modificere  $(c_{ij})$  matricen ved at sætte

$$c_{jj} := c_{jj} + c_{ij} + c_{ji},$$

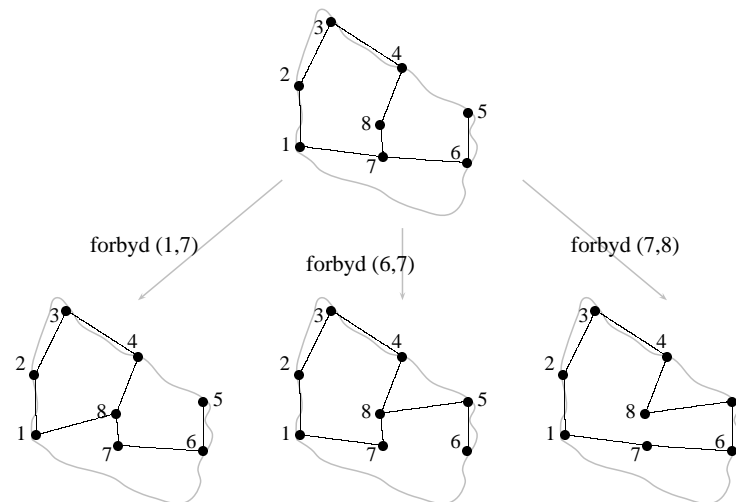
for hvert  $j = 1, \dots, n$ , hvor  $j \neq i$ . Denne modifikation af matricen betyder at hvis knude  $j$  på et senere tidspunkt vælges, vil man automatisk få bidraget  $c_{ij} + c_{ji}$  regnet med i  $c_{jj}$ .

### Eksempel 10 Traveling salesman-problemet (minimeringsproblem)

For traveling salesman-problemet kunne man i princippet vælge en forgreningsstrategi svarende til den for knapsack-problemet: først vælges en kant fra knude 1, derefter vælges en kant fra den knude, hvortil man når, etc. Erfaring viser dog, at dette ikke er nogen god forgreningsstrategi.

En langt bedre strategi er at tage udgangspunkt i det 1-træ, som blev fundet ved grænseværdiberegningen. Hvis alle knuder har valens 2, har vi fundet en Hamilton-kreds, og da vores øvre grænseværdi  $\mathcal{U}$  højst svarer til værdien af denne Hamilton-kreds, vil vi kunne bortsikre delproblemet med en grænseværditest. Ellers må mindst en knude have valens større end 2. Vælg denne knude og forbyd på skift de kanter, som udgår fra knuden. Bemærk, at denne forgreningsstrategi vil have varierende *forgreningsgrad* alt afhængig af den valgte knudes valens.

Forbud af en kant  $(i, j)$  er temmelig simpelt at implementere. Man sætter blot den tilhørende kantvægt  $d_{ij} = d_{ji} = \infty$ . Når branch-and-bound algoritmen vender tilbage fra underproblemet, sættes kantvægten til den gamle værdi.



I eksempel 7 fandt vi et 1-træ, hvor knude 7 havde valens 3. Derfor forbyder vi nu på skift kanterne  $(1, 7)$ ,  $(6, 7)$  og  $(7, 8)$ . I det første delproblem bliver prisen

af 1-træet  $\mathcal{L} = 97$ . I det andet delproblem finder vi  $\mathcal{L} = 98$ , mens i det sidste delproblem bliver  $\mathcal{L} = 105$ . Bemærk, at det sidstnævnte delproblem giver os en lovlig løsning, som vi kan bruge som øvre grænseværdi  $\mathcal{U}$ .

## 6 Kvalitet af grænseværdifunktionen

Antag for et givet maksimeringsproblem, at der findes to øvre grænseværdifunktioner  $\mathcal{U}_1$  og  $\mathcal{U}_2$ . Vi ønsker at kunne sammenligne kvaliteten af de to grænseværdifunktioner.

**Definition 3** Den øvre grænseværdifunktion  $\mathcal{U}_1$  dominerer den øvre grænseværdifunktion  $\mathcal{U}_2$ , hvis der gælder at:

- $\mathcal{U}_1 \leq \mathcal{U}_2$  for alle instanser
- $\mathcal{U}_1 < \mathcal{U}_2$  for mindst en instans

Med andre ord skal grænseværdifunktion  $\mathcal{U}_1$  returnere mindst lige så gode grænseværdier som  $\mathcal{U}_2$ , og der skal findes instanser, hvor  $\mathcal{U}_1$  giver skarpt bedre grænseværdier. Definitionen kræver ikke, at  $\mathcal{U}_1$  altid giver bedre grænseværdier end  $\mathcal{U}_2$ . Dette ville være meget svært at overholde.

### 6.1 Eksempler på dominans

**Eksempel 11** Knapsack-problemet

I eksempel 5 fandt vi en øvre grænseværdi for knapsack-problemet ved at løse det fraktionelle knapsack-problem. Vi vil nu præsentere en alternativ grænseværdi  $\mathcal{U}_{\text{kp}}^0$  givet ved relaxeringen

$$\mathcal{U}_{\text{kp}}^0 = \max \left\{ \sum_{j=1}^n p_j x_j \mid \sum_{j=1}^n w_j x_j \leq c, x_j \geq 0 \right\}. \quad (19)$$

Bemærk, at der er tale om en relaxering, idet vi har udvidet løsningsrummet til  $T = \{(x_1, \dots, x_n) \mid \sum_{j=1}^n w_j x_j \leq c, x_j \geq 0\}$ . Endvidere har begge problemer samme objektfunktion.

Antag, at genstandene er sorteret i aftagende profit-vægt forhold som givet ved (9). Grænseværdien  $\mathcal{U}_{\text{kp}}^0$  kan da udregnes som:

$$\mathcal{U}_{\text{kp}}^0 = \frac{c \cdot p_1}{w_1}. \quad (20)$$

Man vil typisk sortere genstandene i henhold til (9) før man kører en branch-and-bound algoritme. Grænseværdien  $\mathcal{U}_{\text{kp}}^0$  kan da findes i tiden  $O(1)$  inde i branch-and-bound delen.

Bemærk, at  $\mathcal{U}_{\text{kp}}^1$  dominerer  $\mathcal{U}_{\text{kp}}^0$ . Dette ses af, at  $\mathcal{U}_{\text{kp}}^1$  findes ved relaxering til løsningsrummet  $T' = \{(x_1, \dots, x_n) \mid \sum_{j=1}^n w_j x_j \leq c, 0 \leq x_j \leq 1\}$ , mens  $\mathcal{U}_{\text{kp}}^0$  findes ved relaxering til  $T$  defineret under (19). Da  $T' \subseteq T$  og begge relaxeringer bruger samme objektfunktion gælder  $\mathcal{U}_{\text{kp}}^1 \leq \mathcal{U}_{\text{kp}}^0$ .

Det er relativt simpelt at finde et eksempel på at  $\mathcal{U}_{\text{kp}}^1 < \mathcal{U}_{\text{kp}}^0$ . For instansen fra eksempel 1 er  $\mathcal{U}_{\text{kp}}^0 = 27$  mens  $\mathcal{U}_{\text{kp}}^1 = 16\frac{1}{3}$ .

**Eksempel 12** Traveling salesman-problemet (minimeringsproblem)

I eksempel 7 så vi, hvorledes 1-træ relaxeringen kan bruges til at finde den nedre grænseværdi  $\mathcal{L}_{\text{TSP}}^1$ . Vi vil her beskrive, hvorledes man kan stramme denne grænseværdi.

Princippet er at omformulere afstandsmatricen  $(d_{ij})$ , således at længden af en Hamilton-kreds er uændret, men 1-træ relaxeringen returnerer en strammere grænseværdi. Løst sagt, så vil vi "straffe" 1-træer, som ikke er en Hamilton-kreds. I en Hamilton-kreds vil der være netop to valgte kanter for hver knude.

Lad derfor  $v_i$  være valensen af en knude  $i$ , dvs. antallet af incidente kanter, som blev valgt ved løsning af 1-træ relaxeringen. Vi ønsker at "straffe" knuder med valens større end 2, mens knuder med valens mindre end 2 "belønnes". Derfor modificeres afstandsmatricen  $d$  på følgende vis:

$$d'_{ij} = d_{ij} + (v_i - 2) + (v_j - 2).$$

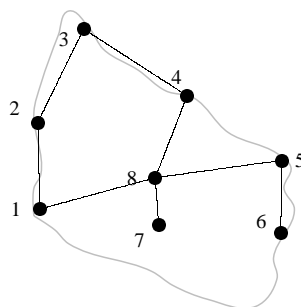
Den nye grænseværdi findes ved at bestemme et mindste 1-træ med hensyn til kantvægtene  $(d'_{ij})$ :

$$\mathcal{L}_{\text{TSP}}^2 = \min \left\{ \sum_{(i,j) \in H} d'_{ij} \mid H \subseteq E, H \text{ er et 1-træ} \right\}. \quad (21)$$

Hvis vi betragter 1-træ relaxeringen fra eksempel 7, så har knuderne følgende valens:  $v_1 = 2, v_2 = 2, v_3 = 2, v_4 = 2, v_5 = 1, v_6 = 2, v_7 = 3,$  og  $v_8 = 2$ . Dette betyder, at alle kanter, som er incidente med knude 7, gøres dyrere, mens alle kanter, som er incidente med knude 5, gøres billigere. Dermed får vi følgende modificerede afstandsmatrix ( $d'_{ij}$ ):

$i \setminus j$	1	2	3	4	5	6	7	8
1	0	11	24	25	29	29	16	15
2	11	0	13	20	31	37	18	17
3	24	13	0	16	29	39	30	22
4	25	20	16	0	14	23	19	12
5	29	31	29	14	0	8	23	14
6	29	37	39	23	8	0	15	21
7	16	18	30	19	23	15	0	8
8	15	17	22	12	14	21	8	0

Ved løsning af 1-træ relaxeringen for det modificerede problem findes  $x_{23} = x_{34} = x_{48} = x_{56} = x_{58} = x_{78} = x_{12} = x_{18} = 1$ , hvor de første seks kanter angiver det mindste udspændende træ blandt knuderne  $2, \dots, n$ . Samlet fås en nedre grænseværdi på  $\mathcal{L}_{\text{TSP}}^2 = 97$ .



For at vise, at  $\mathcal{L}_{\text{TSP}}^2$  er en lovlig nedre grænseværdi, skal vi godtgøre, at der er tale om en relaxering. Løsningsrummet  $S = \{H \subseteq E, H \text{ er en Hamilton-kreds}\}$

er en delmængde af  $T = \{H \subseteq E, H \text{ er et 1-træ}\}$ , som bemærket i slutningen af eksempel 7. Objektfunktionen kan skrives:

$$g(x) = \sum_{(i,j) \in H} d'_{ij} = \sum_{(i,j) \in H} (d_{ij} + (v_i - 2) + (v_j - 2)). \quad (22)$$

Men for enhver Hamilton-kreds har vi

$$\begin{aligned} g(x) &= \sum_{(i,j) \in H} (d_{ij} + (v_i - 2) + (v_j - 2)) \\ &= \sum_{(i,j) \in H} d_{ij} + \sum_{(i,j) \in H} (v_i - 2) + \sum_{(i,j) \in H} (v_j - 2) \\ &= \sum_{(i,j) \in H} d_{ij} + 0 + 0 = f(x), \end{aligned} \quad (23)$$

hvor  $f(x)$  angiver den originale objektfunktion som defineret i (16). Bemærk, at summerne  $\sum_{(i,j) \in H} (v_j - 2)$  bliver nul for enhver Hamilton-kreds, idet kanterne i  $H$  vil gå igennem hver knude netop en gang, og summen af valenserne er netop  $2|H|$ . Vi kan nu se, at problemet (21) har den egenskab, at  $g(x) = f(x)$  for alle løsninger  $x \in S$ .

Man kan gentage iterationsprocessen et antal gange. For ovenstående eksempel giver de efterfølgende iterationer grænseværdierne 98, 99, 99, 100.

Lad  $\mathcal{L}_{\text{TSP}}^3 = \max_{i=1, \dots, k} \mathcal{L}_{\text{TSP}}^2(i)$  betegne den bedste grænseværdi fundet ved ovenstående metode med  $k$  iterationer. Da gælder, at  $\mathcal{L}_{\text{TSP}}^3$  dominerer  $\mathcal{L}_{\text{TSP}}^1$ . Det ses nemt at  $\mathcal{L}_{\text{TSP}}^3 \leq \mathcal{L}_{\text{TSP}}^1$ , idet den første iteration foregår med den originale afstandsmatrix, således at vi finder samme grænseværdi som  $\mathcal{L}_{\text{TSP}}^1$ . Endvidere viser ovenstående eksempel, at der findes en instans, hvor  $\mathcal{L}_{\text{TSP}}^3 > \mathcal{L}_{\text{TSP}}^1$ .

## 7 Kritiske og Semikritiske delproblemer

Vi betragter som tidligere et optimeringsproblem på formen  $z = \max\{f(x) \mid x \in S\}$ . Lad der være givet en forgreningsregel som opdeler  $S$  i mindre delproblemer  $S_1, \dots, S_m$ , og en øvre grænseværdi-funktion  $\mathcal{U}(S_i)$ . Et delproblem  $S_i$  siges at være *kritisk* hvis og kun hvis der gælder at

$$\mathcal{U}(S_i) > z. \quad (24)$$

hvor  $z$  er den optimale løsning.

Det er oplagt, at uanset hvilken søgestrategi og startløsning  $\mathcal{L}$  vi benytter, så skal alle kritiske delproblemer behandles af branch-and-bound algoritmen.

På tilsvarende måde siger vi, at et delproblem  $S_i$  er *semikritisk* hvis og kun hvis, der gælder at

$$\mathcal{U}(S_i) \geq z. \quad (25)$$

**Sætning 8** Hvis  $z$  er kendt fra starten, vil enhver branch-and-bound algoritme kun gennemsøge de kritiske delproblemer (svarende til den valgte forgreningsregel og grænseværdi-funktion).

**Bevis:** Et delproblem er kritisk, hvis  $\mathcal{U}(S_i) > z$ . Hvis vi bruger den nedre grænseværdi  $\mathcal{L} = z$  i alle delproblemer, kan vi forkaste alle delproblemer med  $\mathcal{U}(S_i) \leq z$ .  $\square$

Det interessante ved ovenstående sætning er, at hvis vi kender den optimale løsning fra starten, er der ingen forskel på bedste-først-, dybde-først-, eller bredde-først-søgning. Man kan derfor anvende den mest pladsbesparende variant.

**Sætning 9** Uanset den initiale nedre grænseværdi vil bedste-først søgestrategien kun behandle de semikritiske delproblemer.

**Bevis:** Vi beviser dette indirekte. Antag, at bedste-først søgestrategien behandlede et delproblem  $S_i$ , der ikke var semikritisk, dvs. at  $\mathcal{U}(S_i) < z$ . Dette kan kun lade sig gøre, hvis vi på det givne tidspunkt har  $\mathcal{L} < z$ , idet vi ellers ville forkaste delproblemet med grænseværditesten (17). Da vi bruger bedste-først-søgning må der gælde, at  $\mathcal{U}(S_i) \geq \mathcal{U}(S_j)$  for alle andre delproblemer  $S_j \in \mathcal{L}$ . Men så har vi for alle delproblemer i  $\mathcal{L}$ , at

$$\mathcal{U}(S_j) \leq \mathcal{U}(S_i) < z,$$

dvs.  $z$  kan ikke findes i nogen af løsningsrummene, så  $z$  findes ikke.  $\square$

Sætningen siger med andre ord, at bedste-først-søgning sikrer, at den optimale løsning  $z$  vil blive fundet, inden man betragter delproblemer  $S_i$  med  $\mathcal{U}(S_i) < z$ .

## 8 Kunsten at designe en god branch-and-bound algoritme

Da branch-and-bound paradigmet bruges til løsning af  $\mathcal{NP}$ -hårde optimeringsproblemer, kan man sjældent matematisk bevise, at en given branch-and-bound algoritme er bedre end andre. I stedet må man foretage en empirisk afprøvning med typiske datasæt for at godtgøre, at en algoritme er godt designet.

Følgende tommelfingerregler kan benyttes ved design af algoritmer:

- Det formentlig vigtigste valg i design-processen er *grænseværdifunktionen*. Her skal man tilstræbe den strammest mulige grænseværdi, som kan opnås i polynomiel tid. Da søgetræet vokser eksponentielt vil enhver polynomiel grænseværdi, som effektivt kan bortskære store dele af søgerummet, asymptotisk set kunne betale sig. Erfaringsmæssigt skal grænseværdifunktionen give grænseværdier, som ligger 1 – 2% fra optimum, for at den kan bruges til noget. Hvis grænseværdien ligger 20 – 30% fra optimum, kan man lige så godt bruge *brute-force oprensning*.
- Når først grænseværdifunktionen er valgt, skal man vælge forgreningsreglen, således at de ekstra begrænsninger i delproblemerne kan håndteres effektivt af grænseværdifunktionen.
- Det er en god ide at lægge mange kræfter i at konstruere en god startløsning. Den ideelle situation er at finde en nær-optimal løsning med heuristikker, således at branch-and-bound algoritmen kun bruges til at bevise optimalitet af denne løsning.
- I praksis vil man ofte have en tidsfrist for, hvad brugerne vil acceptere som "rimelig" køretid. Samtidig vil brugeren have et antal instanser, som ønskes løst i denne tid. I en sådan situation er det kunsten at afveje nytten af stramme (men tidsmæssigt dyre) grænseværdier overfor løsere (men tidsmæssigt billige) grænseværdier.
- Branch-and-bound egner sig til *parallel beregning*, dvs. at mange processorer deltager i løsning af et optimeringsproblem. Ideelt set kan  $M$  processorer gennemsøge løsningsrummet  $M$  gange hurtigere end på en enkelt processor. Dette gælder dog kun såfremt, der ikke spildes en væsentlig del af cpu-tiden på kommunikation mellem processorerne, og såfremt processorernes

arbejde ikke “overlapper”. Se Clausen [6] eller Xu og Lau [25] for en dybere diskussion af parallelle algoritmer. Der findes et antal softwarepakker til rådighed, som kan gøre algoritmeudviklingen hurtig, se f.eks. [23].

- Der er i de seneste år præsenteret en del forbedringer til branch-and-bound metoden, heriblandt *strong branching* af Linderoth og Savelsbergh [17], *local branching* af Fischetti og Lodi [9], *pseudocost branching* af Benichou m.fl. [5], samt endelig *reliability branching* af Achterberg, Koch og Martin [1].

*Local branching* har til formål at fastholde søgningen i områder med gode løsninger. Hvis algoritmen allerede har fundet en god kandidat-løsning, vil local branching forhindre branch-and-bound algoritmen i at søge langt væk fra denne løsning. Dette gøres ved at tilføje en ny begrænsning til problemet som sætter en øvre grænse på antallet af variable der må ændres i forhold til kandidat-løsningen. Først når hele det tilhørende løsningsrum er blevet undersøgt, undersøges løsninger som afviger mere fra den givne kandidat-løsning.

*Strong branching* er en teknik til at udvælge en god forgrenings-variabel i branch-and-bound algoritmen. Strong branching undersøger et antal kandidat-variable specificeret af brugeren. For hver kandidat-variabel testes samtlige forgreninger og de tilhørende grænseværdi udregnes. Baseret på disse stikprøver vælges en af de foreslåede kandidat-variable til den egentlige forgrening i branch-and-bound algoritmen.

*Pseudocost branching* vedligeholder information om hvor succesfuldt det har været at forgrene på hver enkelt variabel. *Reliability branching* kombinerer ideerne fra pseudocost branching og strong branching. Achterberg, Koch og Martin [1] sammenligner en række af de bedste branching strategier eksperimentelt.

- Der findes en række open-source pakker tilgængelige på nettet som gør det nemmere at udvikle en branch-and-bound algoritme. Disse omfatter bl.a. *COIN* [18] og *ABACUS* [12]. Begge er specielt velegnet til at udvikle *branch-and-cut* og *branch-and-price* algoritmer.
- Som med alt andet her i livet er *erfaring* en uvurderlig hjælp. Jo flere branch-and-bound algoritmer man har designet, des bedre har man føling for, hvordan de enkelte komponenter skal kombineres for at skabe en god algoritme.

## 9 Opgaver

**Opgave 1** I afsnit 1 blev det nævnt at teorien om  $\mathcal{NP}$ -fuldstændighed ikke udelukker, at mange instanser for et  $\mathcal{NP}$ -hårdt optimeringsproblem kan løses i polynomiel tid.

Betragt *subset-sum-problemet* defineret på en mængde heltal  $S = \{s_1, s_2, \dots, s_n\}$  samt heltallet  $t$ . Subset-sum-problemet i optimeringsversionen søger en delmængde  $S' \subseteq S$ , så  $\sum_{j \in S'} s_j$  kommer så tæt på  $t$  som muligt uden at overskride  $t$ .

- 1 Vis, at subset-sum-problemet kan løses i  $O(nt)$  tid vha. dynamisk programmering
- 2 I Cormen m.fl. [7] afsnit 34.5.5 vises, at subset-sum-problemet i afgørlighedsversionen er  $\mathcal{NP}$ -fuldstændigt ved reduktion fra 3CNF-satisfiability. Opskriv et udtryk for størrelsesordenen af  $t$  ved denne reduktion i forhold til input-størrelsen af 3CNF-satisfiability.
- 3 Siger reduktionen fra 3CNF-satisfiability noget om sværheden af at løse subset-sum for små værdier af  $t$ ? Er subset-sum “svært” at løse i praksis, hvis de fleste realistiske problemer omfatter moderat små værdier af  $t$ ?

**Opgave 2** Bevis, at hvis den optimale løsning  $z$  kendes fra starten af en branch-and-bound algoritme, vil alle søgestrategier besøge lige mange knuder.

**Opgave 3** Betragt *knapsack-problemet*, som blev defineret i eksempel 1.

- Brug bredde-først-søgning for instansen fra eksempel 1, hvor du benytter grænseværdi  $U_{kr}^0$  til at bortskære delproblemer.
- Vi definerer *dominans af delproblemer* som følger: Betragt to delproblemer  $S_1$  og  $S_2$ , hvor begge problemer rummer præcis den samme mængde frie variable. Lad endvidere  $\bar{p}_1$  være profit-summen svarende til de variable, der er blevet låst fast til en værdi ved forgrening i  $S_1$ , og  $\bar{w}_1$  være den tilhørende vægtsum. På samme måde definerer vi  $\bar{p}_2$  og  $\bar{w}_2$  for delproblem  $S_2$ . Vi siger, at  $S_1$  dominerer  $S_2$ , hvis

$$\bar{p}_1 \geq \bar{p}_2 \text{ og } \bar{w}_1 \leq \bar{w}_2.$$

Vis, at hvis  $S_1$  dominerer  $S_2$ , så kan  $S_2$  forkastes uden, at vi går glip af en optimal løsningsværdi.

- Brug dominans i bredde-først-søgningen fra første del af opgaven.
- Kan man give en øvre grænseværdi for køretiden af algoritmen?
- Sammenlign bredde-først-søgning, hvor man bruger dominans, med *dynamisk programmering*.

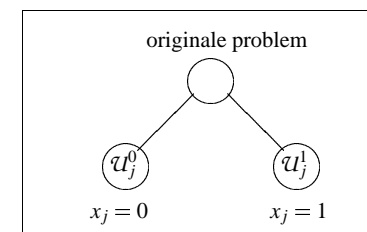
**Opgave 4** I *dense subgraph-problemet* blev det antaget at alle kantvægte er ikke-negative. Vis, at hvis der findes negative kantvægte, så kan problemet transformeres til et ækvivalent problem uden negative kantvægte. Ækvivalent betyder her, at man vil finde den samme løsning  $U$ , men ikke nødvendigvis den samme løsningsværdi.

**Opgave 5** I definitionen af et optimeringsproblem (1) er vi kun interesseret i en enkelt løsning, som maksimerer objektfunktionen. Der findes dog anvendelser, hvor man har behov for at finde samtlige optimale løsninger. Hvorledes skal branch-and-bound paradigmet ændres, således at man kan bestemme samtlige optimale løsninger?

**Opgave 6** Betragt et optimeringsproblem på formen (1), hvor vi antager at alle beslutningsvariable er 0-1 variable. Det vil tydeligvis være en fordel, hvis man på forhånd kunne låse nogle af variablene fast på deres optimale værdi. Dermed ville branch-and-bound algoritmen kunne nøjes med at gennemløbe et mindre søgetræ, således at køretiden reduceres.

Variabelreduktion kan ses som et specialtilfælde af branch-and-bound algoritmen. Hvis der er  $n$  beslutningsvariable  $x_1, \dots, x_n$ , vil man gennemløbe disse for  $i = 1, \dots, n$  og hver gang anbringe variabel  $x_i$  i rodknuden af branch-and-bound algoritmen. Algoritmen vil da forgrene på to delproblemer  $S_i^0$  (hvor  $x_i = 0$ ) og  $S_i^1$  (hvor  $x_i = 1$ ). Grænseværdierne for de to delproblemer bestemmes til  $\mathcal{U}_i^0$  og  $\mathcal{U}_i^1$ .

- Bevis, at hvis  $\mathcal{U}_i^0 < \mathcal{L}$  for en given nedre grænseværdi  $\mathcal{L}$ , så kan vi låse værdien af  $x_i$  til 1. Bevis tilsvarende, at hvis  $\mathcal{U}_i^1 < \mathcal{L}$ , så kan vi låse værdien af  $x_i$  til 0.
- Det kunne være ønskværdigt at bruge kriteriet  $\mathcal{U}_i^0 \leq \mathcal{L}$  til at låse værdien af  $x_i$  til 1 (og tilsvarende ved låsning til 0). Under hvilke omstændigheder kan man benytte denne strammere test?



Figur 4: Forgrelning på de to forskellige værdier af  $x_j$ .

- Anvend begge reduktionsprincipper på knapsack-problemet fra eksempel 1, hvor  $\mathcal{L} = 14$ .
- Generaliser princippet til problemer, hvor beslutningsvariablene ikke nødvendigvis er 0-1 variable.

**Opgave 7** *Lagrange-relaksering*. Betragt knapsack-problemet

$$z = \max \left\{ \sum_{j=1}^n p_j x_j \mid \sum_{j=1}^n w_j x_j \leq c, x_j \in \{0, 1\} \right\}. \quad (26)$$

For et givet reelt tal  $\lambda \geq 0$  sletter vi begrænsningen  $\sum_{j=1}^n w_j x_j \leq c$  og tilføjer i stedet "straffen"  $-\lambda(\sum_{j=1}^n w_j x_j - c)$  i objektfunktionen. Dermed får vi problemet

$$z_{\text{LR}} = \max \left\{ \sum_{j=1}^n p_j x_j - \lambda \left( \sum_{j=1}^n w_j x_j - c \right) \mid x_j \in \{0, 1\} \right\}. \quad (27)$$

- Vis at (27) er en relaxering af (26) for ethvert  $\lambda \geq 0$ .
- For instansen fra eksempel 1 aftegn  $z_{\text{LR}}$  som funktion af  $\lambda$  i intervallet  $[0, 5]$ . For hvilken værdi af  $\lambda$  fås den strammeste grænseværdi?
- Betragt knapsack-problemet i generel form (26), og lad  $b$  være givet som i (10). Vis at  $\lambda = p_b/w_b$  fører til den strammeste grænseværdi. Vi vil betegne denne grænseværdi  $\mathcal{U}_{\text{kp}}^2$ .
- Gælder der, at  $\mathcal{U}_{\text{kp}}^2$  dominerer  $\mathcal{U}_{\text{kp}}^1$  eller vice versa?

**Opgave 8** *Assignment-problemet* kan defineres som følger:

$$z = \min \left\{ \sum_{j=1}^n c_{ij}x_{ij} \mid \sum_{i=1}^n x_{ij} = 1, \sum_{j=1}^n x_{ij} = 1, x_{ij} \in \{0, 1\} \right\}. \quad (28)$$

Sagt i ord er der givet et antal job  $j = 1, \dots, n$  og et antal medarbejdere  $i = 1, \dots, n$ . Endvidere er der en omkostning  $c_{ij}$  ved at lade medarbejder  $i$  udføre job  $j$ . Problemet er at tildele hver medarbejder netop et job, således at de samlede udgifter minimeres. Assignment-problemet kan løses i polynomiel tid.

- Brug assignment-problemet til at udlede en nedre grænseværdi for traveling salesman-problemet.
- Bevis, at der er tale om en relaxering.
- Hvis vi kalder den fundne grænseværdi  $\mathcal{L}_{\text{TSP}}^4$ . Gælder der da, at  $\mathcal{L}_{\text{TSP}}^4$  dominerer  $\mathcal{L}_{\text{TSP}}^1$ ?

**Opgave 9** Vi betragter igen dense subgraph-problemet fra eksempel 2 og 6. Vi udleder nu en ny grænseværdi  $\mathcal{U}_{\text{DSP}}^2$ , der beregnes som følger. Først finder vi

$$\tilde{c}_i = c_{ii} + \max \left\{ \sum_{j \in U} c_{ij} \mid U \subseteq V, |U| = k - 1 \right\}, \quad (29)$$

og derefter

$$\mathcal{U}_{\text{DSP}}^2 = \max \left\{ \sum_{i \in U} \tilde{c}_i \mid U \subseteq V, |U| = k \right\}. \quad (30)$$

- Vis at  $\mathcal{U}_{\text{DSP}}^2$  er en lovlig grænseværdi.
- Vis at  $\mathcal{U}_{\text{DSP}}^2$  dominerer  $\mathcal{U}_{\text{DSP}}^1$ .

**Opgave 10** Bevis, at hvis antallet af semikritiske delproblemer i søgetræet for en given branch-and-bound algoritme er polynomielt begrænset, så tilhører problemet klassen  $\mathcal{P}$ .

**Opgave 11** I afsnit 4.3 definerede vi *monotonitet af grænseværdi*.

- Vis, at grænseværdierne  $\mathcal{U}_{\text{KP}}^0, \mathcal{U}_{\text{KP}}^1, \mathcal{U}_{\text{KP}}^2$  er monotone.
- Hvilke af grænseværdierne  $\mathcal{L}_{\text{TSP}}^1, \mathcal{L}_{\text{TSP}}^3, \mathcal{L}_{\text{TSP}}^4$  er monotone?

## Indeks

- 1-træ, 30
- 1-træ relaxering, 20, 32
- 3CNF-satisfiability, 38
- ABACUS, 37
- afgørlighedsproblem, 5
- assignment-problemet, 41
- bedste-først-søgning, 25, 35
- branch-and-bound, 22
- branch-and-bound paradigmet, 6
- branch-and-cut, 37
- branch-and-price, 37
- branching
  - local, 37
  - pseudocost, 37
  - reliability, 37
  - strong, 37
- bredde-først-søgning, 25, 35, 38
- brute-force opremsning, 11, 36
- certifikat, 11
- COIN, 37
- Concorde, 9
- del-og-hersk, 6
- delproblem, 12, 15
- dense subgraph-problemet, 8, 19, 29, 39, 41
- divide-and-conquer, 12, 16
- dominans af delproblemer, 38
- dominerer, 31
- dybde-først-søgning, 25, 35
- dynamisk programmering, 39
- effektivitet, 17
- eksempel, 1:7, 2:8, 3:9, 4:12, 5:17, 6:19, 7:20, 8:26, 9:29, 10:30, 11:31, 12:32
- erfaring, 37
- EX $\mathcal{P}$  problem, 11
- forgren-og-begræns, 6
- forgreningsgrad, 30
- forgreningsregel, 23, 26
- fraktionelle knapsack-problem, 17
- grådig algoritme, 24
- grænseværdifunktion, 36
- grænseværditest, 21
- Hamilton-kreds, 5
- heltallig løsningsværdi, 17
- heuristisk styret søgning, 26
- kapacitet, 7
- klike-problemet, 8
- knapsack-problemet, 7, 12, 17, 26, 31, 38
- konstruktiv algoritme, 6
- kritisk delproblem, 34
- Lagrange-relaxering, 24, 40
- lineær-relaxering, 17, 24
- local branching, 37
- lukkede delproblemer, 22
- løsningsrum, 5
- løsningsværdi delmængde  $S_i$ , 12
- maksimeringsform, 5
- maksimeringsproblem, 7
- mindste udspændende træ, 20, 21
- minimalt 1-træ, 20

minimeringsform, 5  
 minimieringsproblem, 7  
 monoton, 22  
 monotonitet af grænseværdi, 22, 41  
 MST, 20, 21  
 nedre grænseværdi, 15, 23, 24  
 $\mathcal{L}_{TSP}^1$ , 21, 32, 34, 41  
 $\mathcal{L}_{TSP}^2$ , 32–34  
 $\mathcal{L}_{TSP}^3$ , 34, 41  
 $\mathcal{L}_{TSP}^4$ , 41  
 $\mathcal{N}(\mathcal{P})$ -fuldstændigt problem, 5  
 $\mathcal{N}(\mathcal{P})$ -hårdt problem, 5  
 objektfunktion, 5  
 optimal løsning, 5  
   finde alle løsninger, 39  
 optimal løsningsværdi, 5  
 optimeringsproblem, 5  
 paradigme, 6  
 parallel beregning, 36  
 profit, 7  
 pseudocost branching, 37  
 reduktion af variable, 39  
 relaxering, 15, 23  
 reliability branching, 37  
 semidefinit-relaxering, 24  
 semikritisk delproblem, 35, 41  
 skabelon, 6  
 strong branching, 37  
 subset-sum-problem, 38  
 surrogat-relaxering, 24  
 symmetrisk traveling salesman-problem,  
   9  
 sætning, 1:11, 2:12, 4:16, 3:16, 5:16,  
   6:17, 7:22, 9:35, 8:35  
 søgebaseret algoritme, 6

søgestrategi, 23, 24  
 søgetræ, 15  
 traveling salesman-problemet, 5, 9, 20,  
   30, 32, 41  
 TSP, 5, 9, 20, 30, 32, 41  
 udspændende træ, 20  
 underproblem, 15  
 valens, 32  
 verificere, 5, 11  
 vægt, 7  
 åbne delproblemer, 22  
 øvre grænseværdi, 15, 23  
 $\mathcal{U}_{DSP}^1$ , 19, 20, 41  
 $\mathcal{U}_{DSP}^2$ , 41  
 $\mathcal{U}_{KP}^0$ , 31, 32, 38, 41  
 $\mathcal{U}_{KP}^1$ , 17, 18, 27, 32, 40, 41  
 $\mathcal{U}_{KP}^2$ , 40, 41  
 øvre grænseværdifunktion, 23

## Litteratur

- [1] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33:42–54, 2005.
- [2] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Concorde. <http://www.math.princeton.edu/tsp/concorde.html>.
- [3] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding tours in the tsp. Technical Report Report Number 99885, Research Institute for Discrete Mathematics, Universitat Bonn, 1999.
- [4] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. In *In: Rolf G. Karlsson, Andrzej Lingas (Eds.): Algorithm Theory - SWAT '96, 5th Scandinavian Workshop on Algorithm Theory, Reykjavík, Iceland, July*, volume 1097, pages 136–148. Springer, 1996.
- [5] M. Benichou, J.M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mixed-integer programming. *Math Programming*, 1:76–94, 1971.
- [6] J. Clausen. Branch-and-bound algorithms — principles and examples, 1997.
- [7] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.
- [8] E. Erkut. The discrete  $p$ -dispersion problem. *European Journal of Operational Research*, 46:48–60, 1990.
- [9] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming B*, 98:23–47, 2003.
- [10] P. Hansen and I.D. Moon. Dispersing facilities on a network, 1988. Presentation at the TIMS/ORSA Joint National Meeting, Washington, D.C.
- [11] R. Hassin, S. Rubinstein, and A. Tamir. Approximation algorithms for maximum dispersion. *Operations Research Letters*, 21:133–137, 1997.
- [12] M. Jünger and S. Thienel. The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization. *Software: Practice and Experience*, 30:1325–1352, 2000. <http://www.informatik.uni-koeln.de/abacus/>.

- [13] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.
- [14] G. Kortsarz and D. Peleg. On choosing a dense subgraph. In *Proceedings of the 34th Annual IEEE Symposium on Foundation of Computer Science*, pages 692–701, 1993.
- [15] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *The Traveling Salesman Problem*. John Wiley & Sons, Chichester, 1985.
- [16] H.R. Lewis and C.H. Papadimitriou. *Elements of the Theory of Computation*. Pearson Education, New Jersey, 1998.
- [17] J.T. Linderoth and M.W.P. Savelsbergh. A computational study of branch and bound search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11:173–187, 1999.
- [18] R. Lougee-Heimer. The Common Optimization INterface for operations research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development archive*, 47:57–66, 2003. <http://www.coin-or.org/>.
- [19] D. Pisinger. <http://www.diku.dk/~pisinger/codes>.
- [20] D. Pisinger. Upper bounds and exact algorithms for  $p$ -dispersion problems. *Computers and Operations Research*, 33:1380–1398, 2006.
- [21] S.S. Ravi, D.J. Rosenkrantz, and G.K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42:299–310, 1994.
- [22] A. Srivastav and K. Wolf. Finding dense subgraphs with semidefinite programming. In K. Jansen and J. Rolim, editors, *Approximation algorithms for combinatorial optimization*, volume 1444 of *Lecture Notes in Computer Science*, pages 181–191. Springer, 1998.
- [23] Department of Computer Science University of Paderborn. Portable parallel branch-and-bound library. <http://www.uni-paderborn.de/fachbereich/AG/monien/SOFTWARE/PPBB/ppbblib.html>, 1996.
- [24] L.A. Wolsey. *Integer Programming*. J. Wiley, 1998.

- [25] C. Xu and F. Lau. *Load balancing in Parallel Computers: Theory and Practice*. Kluwer Academic Publisher, 1997.