MATEMATISK AFDELING
KØBENHAVNS UNIVERSITET

Bachelorafhandling

Christian Mikkelsen Jensen, 060582

Anders Bjerg Pedersen, 070183

Sports Scheduling

Vejleder: Kent Andersen
Afleveret den 10. november 2006

Indhold

0.1 Indledning

Hvorfor i alverden vælger nogen at bruge et halvt år af deres studier på at finde ud af, hvordan man planlægger en turnering? De fleste af os har deltaget i og måske endda prøvet at lave en turnering, det være sig en volleyballturnering på ens gymnasium eller en bobturnering til et socialt arrangement. Dette virker ikke umiddelbart videre kompliceret, men så snart det f.eks. skal planlægges, hvem der spiller hjemme og ude, stiger kompleksiteten af opgaven betragteligt. Hvis der endvidere skal tages hensyn til bestemte deltagende holds ønsker, mediedækning, tilskuertilfredshed, retfærdighed i spillet, udnyttelse af faciliteter og andre faktorer, bliver et forholdsvis simpelt problem nu til en meget kompleks problemstilling med stor økonomisk betydning. På grund af disse faktorer er matematikken nu kommet til at spille en central rolle inden for moderne planlægning af sportsbegivenheder, hvis teoretiske resultater har vundet indpas gennem de sidste 30 år. Specielt i de sidste 10 år er matematiske metoder blevet anvendt flere og flere gange til at planlægge større turneringer. For bare 10 år siden kunne planlægningen være henlagt til et ældre ægtepar[1], der brugte nogle weekender i deres stue på at konstruere en spilleplan, der typisk var langt fra optimal.

Omkring 2. verdenskrig begyndte man af logistiske årsager at se matematisk på scheduling af forskellige opgaver inden for militæret. Her skal scheduling forstås som den disciplin, der ud fra et givet antal betingelser søger at finde en optimal rækkefølge eller plan for en mængde af opgaver, der skal løses. Klassisk taler man om jobs, der skal løses på maskiner, men man kunne f.eks. også se det som en begrænset mængde ansatte i en virksomhed, der skal løse et bestemt antal opgaver hurtigst og mest effektivt. De opnåede resultater bredte sig nu også til det private, og computerens indtog var her med til kraftigt at forbedre mulighederne for mere præcis og effektiv scheduling, ligesom man hurtigere kunne nå frem til løsninger. Sports scheduling opstod for alvor i slutningen af 70'erne og begyndelsen af 80'erne. I starten var det primært et teoretisk specialtilfælde af scheduling, men disciplinen er nu et etableret forskningsområde.

Med den løbende udvikling af computerkraft og brug af teknikker som heltalsprogrammering og constraint programming indså man potentialet i sports scheduling. Nu kunne klubbernes og andres betingelser for alvor implementeres i løsningen af problemerne, og siden er antallet af artikler, der beskriver matematiske metoder til løsning af forskellige turneringer inden for et hav af sportsgrene, steget markant fra år til år, senest med scheduleringen af den nuværende sæson af den danske SAS-liga. Alt dette har bidraget til, at sports scheduling nu er sit eget område under operationsnalyse.

Opgaven består af 5 afsnit, hvor det første omhandler klassisk scheduling, herunder nogle velkendte problemer. Desuden beskriver vi introducerende dele af kompleksitetsanalysen, der udtaler sig om, hvor "svært" det er at løse et givent problem. Vi kommer også ind på, at man kan afgøre sværhedsgraden af et problem ved at reducere det til et andet problem med allerede kendt kompleksitet.

I afsnit 2 ser vi på disciplinen heltalsprogrammering, hvor vi blandt andet beskriver Branch and Bound, der er et kraftfuldt værktøj til at løse problemer, hvor der indgår heltallige variable. Derudover kommer vi ind på algoritmer, der kan indsnævre problemerne, inden Branch and Bound-algoritmerne anvendes. Her vil vi blandt andet se på

[1] Dette var rent faktisk tilfældet med en større amerikansk basketballturnering for nogle år siden.

gyldige uligheder og Chvátal-Gomorys algoritme.

I afsnit 3 ser vi kort på constraint programming, der er en forholdsvist ny teknik, der på nogle områder er mere effektiv end heltalsprogrammering. Vi vil fokusere på de metoder, der er mest relevante for sports scheduling. Desuden ser vi på en metode kaldet Benders decomposition, der senere i opgaven er grundlaget for scheduleringen af SAS-ligaen.

Afsnit 4 omhandler generel teori inden for sports scheduling. Der indledes med terminologi, definitioner og klassiske problemstillinger, hvorefter vi fremlægger en række teoretiske resultater omhandlende nogle af de klassiske problemstillinger vedrørende konstruktionen af en retfærdig spilleplan, herunder den kanoniske metode. I afsnittet indgår resultater fra grafteorien som et væsentligt redskab. Til sidst i afsnittet inddrages en mængde af betingelser fra teorien, hvor der blandt andet fremlægges resultater om kompleksitet af turneringsplanlægning generelt og feasibility af spilleplaner.

Opgavens sidste afsnit omhandler et konkret eksempel på schedulering af en sportsbegivenhed, nemlig den danske SAS-liga for sæsonen 2006/07, altså den nuværende, der er udført af Rasmus V. Rasmussen. Afsnittet afsluttes med en vurdering af metodens effektivitet sammenlignet med tidligere års resultater, der ikke har anvendt matematiske modeller.

Kendskab til udvalgte emner fra operationsanalyse (lineær programmering) og basal grafteori vil lette læsningen af opgaven. Afsnittene kan i teorien læses uafhængigt af hinanden, med undtagelse af afsnit 5, der forudsætter afsnit 4. Der er dog emner, der henvises til senere i opgaven, som f.eks. et resultat omhandlende kompleksitet.

Vi vil gerne takke Rasmus V. Rasmussen for opklarende svar og ikke mindst vores vejleder Kent Andersen for gode råd og kyndig vejledning gennem hele forløbet.

København, november 2006

Christian Mikkelsen Jensen
Anders Bjerg Pedersen

0.2 Resumé

Dette projekt omhandler emnet sports scheduling, der vedrører planlægning af turneringer. Projektet opbygger indledningsvis en række værktøjer til brug inden for løsning af sports scheduling-problemer, herunder generel scheduleringsteori, kompleksitetsanalyse og reduktion, heltalsprogrammering, constraint programming og Benders decomposition. I sports scheduling-afsnittet arbejder vi primært med minimering af breaks, hvor et iøjnefaldende resultat er de Werras sætning om det minimale antal breaks i en turnering. Projektet afsluttes med at se på en konkret schedulering af den bedste danske fodboldliga og en vurdering af denne schedulerings kvalitet.

0.3 Abstract

This project deals with the subject of sports scheduling, which concerns the planning of tournaments. At first the project introduces several tools needed for solving sports scheduling problems, such as general theory of scheduling, complexity analysis and reduction, integer programming, constraint programming and Benders decomposition. In the section on sports scheduling the primary subject is break minimization, in which a conspicuous result is de Werra's theorem on the minimum number of breaks in a tournament. The project is concluded with a concrete example of scheduling the best Danish soccer league and an assessment of the quality of the scheduling proces.

1 Schedulering

Da sports scheduling udspringer fra generel schedulering, vil vi i dette afsnit kort se på anvendelse af scheduling, idet vi gennemgår to klassiske problemer inden for schedulering, nemlig Job Shop Scheduling og Travelling Salesman-problemet. Med udgangspunkt i disse problemer vil vi bagefter diskutere reduktion og kompleksitet og herefter fremlægge generel teori vedrørende disse to emner.

1.1 Job Shop Scheduling

Vi vælger nu at se på et af de mere klassiske problemer inden for schedulering, nemlig Job Shop Scheduling (JSS). Job Shop Scheduling bygger på ideen om en fabrik, hvor et antal job skal igennem et antal maskiner. Rækkefølgen, hvori de forskellige jobs skal igennem maskinerne er ikke nødvendigvis den samme, ligesom ikke alle jobs nødvendigvis skal igennem alle maskiner (hvis nogen overhovedet). Maskinerne på fabrikken antages kun at kunne udføre ét job af gangen.

Vi lægger ud med nogle definitioner, der er nødvendige for at kunne tale formelt om schedulering samt for at indføre en god notation:

Procestid(p_{ij}) – Procestid for job j på maskine i. Hvis job j ikke skal igennem maskine i, er $p_{ij} = 0$.

Arrival Date (r_j) – Tidspunktet, hvor job j ankommer til systemet.

Due Date (d_j) – Tidspunktet, hvor job j forventes færdigt.

Deadline ($\bar{d}_j$) – Tidspunktet, hvor job j *skal* være færdigt.

Ventetid (w_{ij}) – Hvor lang tid job j venter på at komme til på maskine i (regnet fra tid 0).

Completion Time ($C_j = w_{i_{n_j}j} + p_{i_{n_j}j}$) – Tidspunktet, hvor job j er færdigt i systemet. Her er n_j job j's sidste opgave på maskine i.

Makespan ($C_{max} = \max_{j=1,\ldots,n}(C_j)$) – Tidspunktet, hvor systemet er færdigt med at arbejde (sidste opgave er udført).

Tardiness ($T_j = \max(C_j - d_j, 0)$) – Forsinkelse for job j. Tardiness kan per definition ikke være negativ, derfor sættes den til 0, hvis job j skulle blive færdig før tid.

Målet med Job Shop Scheduling er så f.eks. at minimere tardiness eller makespan. Vi vil her koncentrere os om at minimere tardiness for n jobs på m maskiner og opskrive problemet som et heltalsproblem. ([7], p.20f)

Lad $h_j = o_{i_1 j}, \ldots, o_{i_{n_j} j}$ være den sekvens af opgaver, som job j skal igennem (i den rækkefølge, der nu er fastsat), hvor i_k angiver maskinens nummer, og lad $Q_i = \{q_{i1}, \ldots, q_{im_i}\}$

være mængden af opgaver, der bliver udført på maskine i, hvis rækkefølge selvfølgelig afhænger af, hvordan jobbene scheduleres. Den tid, det tager for en maskine at bearbejde en opgave angives som p_{ij}. Desuden har hvert job en arrival date r_j og en due date d_j.

Vi lader nu w_{i_kj} være starttiden for job j's k'te opgave og ser så på to på hinanden følgende opgaver, o_{i_kj} og $o_{i_{k+1}j}$, for job j. Vi ønsker, at $o_{i_{k+1}j}$ først startes, når opgave o_{i_kj} er færdig, dvs.

$$w_{i_{k+1}j} \geq w_{i_kj} + p_{i_kj}.$$

Desuden ønsker vi, at job j ikke kan startes, før det ankommer til fabrikken:

$$w_{i_1j} \geq r_j.$$

Vi ønsker at minimere den samlede tardiness som summen af tardiness for hvert job (T_j), altså

$$\sum_{j=1}^{n} T_j \geq \sum_{j=1}^{n} \max(w_{i_{n_j}j} + p_{i_{n_j}j} - d_j, 0).$$

Vi ser nu på rækkefølgen på de enkelte maskiner (selve planlægningsfasen), hvor hver maskine kun kan udføre en enkelt opgave af gangen. Lad nu α og β, $\alpha < \beta$, angive numrene på to opgaver i mængden Q_i, dvs. to opgaver der skal udføres på maskine i. Disse kan ikke udføres samtidig, derfor gælder følgende:

$$(w_\beta \geq w_\alpha + p_\alpha) \ \vee \ (w_\alpha \geq w_\beta + p_\beta).$$

Dette ønsker vi omskrevet til ulighedsform, derfor indfører vi den binære variabel $x_{\alpha\beta}$ og M, der er et meget stort tal, og får følgende:

$$\begin{aligned} w_\beta &\geq w_\alpha + p_\alpha - M(1 - x_{\alpha\beta}) \\ w_\alpha &\geq w_\beta + p_\beta - M x_{\alpha\beta} \end{aligned}$$

Dette betyder så, at når $x_{\alpha\beta} = 1$, udføres opgave α før β og omvendt, hvis $x_{\alpha\beta} = 0$.

Vi kan nu opskrive problemet på binær form med 0-1-variable. Da alle indgående tal er heltal, vil vores resultater også være det:

$$\begin{aligned} \min\ Z &= \sum_{j=1}^{n} T_j \\ \text{s.t.} & \\ w_{i_{k+1}j} &\geq w_{i_kj} + p_{i_kj},\ \forall k < n_j,\ j = 1, \ldots, n \\ w_{i_1j} &\geq r_j,\ \forall j \\ \left.\begin{aligned} T_j &\geq w_{i_{n_j}j} + p_{i_{n_j}j} - d_j \\ T_j &\geq 0 \end{aligned}\right\} &,\quad j = 1, \ldots, n \\ \left.\begin{aligned} w_\beta &\geq w_\alpha + p_\alpha - M(1 - x_{\alpha\beta}) \\ w_\alpha &\geq w_\beta + p_\beta - M x_{\alpha\beta} \end{aligned}\right\} &,\ \forall \alpha, \beta \in Q_i,\ \alpha < \beta,\ i = 1, \ldots, m \\ x_{\alpha\beta} &\in \{0, 1\},\ \forall \alpha, \beta. \end{aligned}$$

1.2 Travelling Salesman Problem

Et af de nok mest klassiske problemer inden for operationsanalyse er Travelling Salesman-problemet (TSP). Dette problem går i al sin enkelhed ud på, at en sælger skal besøge et antal byer netop én gang hver på en sådan måde, at han rejser den korteste afstand (eller bruger mindst tid / færrest omkostninger alt efter problemet). For at formalisere denne problemstilling opskriver vi den som et heltalsproblem. ([23], p.7f)

Vi opstiller problemet som en graf $G = (V, E)$, hvor knuderne i V er de n byer, sælgeren skal besøge, og kanterne i E er forbindelserne mellem byerne, så G danner en komplet graf. Lad d_{ij} være afstanden fra by i til by j og lad x_{ij} svarende til kanten mellem by i og by j være den binære variabel, der er 1, hvis vejen (kanten) anvendes og 0 ellers. For $i = j$ sætter vi $d_{ij} = M$, hvor M er et meget stort tal. Da han besøger hver by netop én gang, svarer dette problem til at finde en Hamilton-kreds i G, så den rejste afstand bliver mindst mulig.
Vi skal for det første sikre os, at sælgeren forlader by i præcis én gang og ankommer til by j præcis én gang. Dette gøres ved følgende to constraints:

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \text{og} \quad \sum_{i=1}^{n} x_{ij} = 1, \ i, j = 1, \ldots, n.$$

For at sikre os, at vi rent faktisk får en Hamilton-kreds, indfører vi en delmængde $S \subset V, \ S \neq \emptyset$, så der altid er mindst én forbindelse mellem S og $V \backslash S$ [2]:

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1, \ S \subset V, \ S \neq \emptyset.$$

Sammenfatter vi ovenstående, kan vi skrive det generelle problem op, idet vi ønsker at minimere den rejste afstand:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} \\
\text{s.t.} \quad & \\
& \sum_{j=1}^{n} x_{ij} = 1, \ i = 1, \ldots, n \\
& \sum_{i=1}^{n} x_{ij} = 1, \ j = 1, \ldots, n \\
& \sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1, \ S \subset V, \ S \neq \emptyset \\
& x_{ij} \in \{0, 1\}, \ i, j = 1, \ldots, n, \ i \neq j.
\end{aligned}$$

[2] Vi bemærker, at der altid også vil være et lige antal forbindelser, da sælgeren skal returnere til sin startby

1.3 Reduktion og kompleksitet

Ingen operationsanalytiker ville løse nogen af ovenstående problemer i hånden, han ville vende sig mod en computer. Derfor er det vigtigt at finde ud af, hvor lang tid en computer vil skulle bruge på at løse forskellige problemer. Her kommer to faktorer til at spille ind: størrelsen af inputtet og de beregninger, der skal udføres på dette input. Skal man addere to 'elementer', er der stor forskel på, om disse elementer er f.eks. heltal, kommatal eller matricer.

Mht. til selve beregningerne indfører vi en funktion $\mathcal{O}(g(n))$, hvor $g(n)$ angiver, hvor hurtigt antallet af elementære operationer vokser som funktion af inputtet n. Hvis vi f.eks. lader funktionen $T(n) = c_1 + c_2 n + \frac{c_3 n(n-1)}{2} + c_4 n$, hvor c_i er konstanter afhængige af computeren, være antallet af elementære operationer, der er krævet for at løse et problem (i dette tilfælde boblesortering af n tal), vælges $g(n) = n^2$, da dette er højeste led i $T(n)$. Dvs. boblesortering af n tal er $\mathcal{O}(n^2)$ ([8], **2**.2).

Det bliver her specielt interessant at finde ud af, om $g(n)$ vokser polynomielt (som i eksemplet med boblesortering) eller eksponentielt.

Programmeringsmæssigt er ovenstående formulering af TSP ikke umiddelbart særlig effektiv, når n bliver stor, da antallet af mulige delmængder S vokser eksponentielt[3]. Dette gælder generelt for optimeringsproblemer, derfor er man interesseret i at omskrive problemerne, så der skal anvendes færre computeroperationer for at løse dem.

Den mest benyttede metode til dette er at omskrive fra optimeringproblem til beslutningsproblem (ja/nej-problem). Oftest indlægges der en grænse, B (eksempelvis et omkostningsmaksimum eller maksimal tid til rådighed), som så skal forbedres.

I TSP kan dette f.eks. gøres ved at stille spørgsmålet: "*Findes der* en Hamilton-kreds i G, så den rejste afstand er mindre end eller lig B?"([8], p**2**-4)

Vi skal nu se nærmere på reduktion af problemer til mindre komplicerede problemer: ([23], p83ff)

Definition 1.1.
$\mathcal{NP}$ er den klasse af ja/nej-problemer, for hvilke der for ethvert JA-svar findes et 'kort' polynomielt bevis.
$\mathcal{P}$ er den klasse af ja/nej-problemer i $\mathcal{NP}$, for hvilke der findes en polynomiel algoritme.

Altså har vi, at $\mathcal{P}$ er de 'nemme' problemer fra $\mathcal{NP}$.

Definition 1.2.
Lad $P, Q \in \mathcal{NP}$. Hvis en instans af P kan konverteres til en instans af Q (i størrelsen af instansen af P) inden for polynomiel tid, siger vi, at P er polynomielt reducibelt *til Q, og vi skriver $P \propto Q$.*
Klassen $\mathcal{NPC}$ (klassen af $\mathcal{NP}$-komplette problemer), er den delmængde bestående af problemer $P \in \mathcal{NP}$, sådan at for alle $Q \in \mathcal{NP}$ gælder, at Q er polynomialt reducibelt til P.

Bemærkning 1.3. *Ud fra definitionen har vi, at $(P \propto Q) \not\Rightarrow (Q \propto P)$ generelt, og reducibilitet er transitiv, dvs. $(P \propto Q) \wedge (Q \propto R) \Rightarrow (P \propto R)$.*

[3] Der findes $2^n - 1$ ikke-tomme delmængder af en mængde med n elementer

Vi viser nu en brugbar proposition inden for reduktion ([23], p85):

Proposition 1.4.
Lad $P, Q \in \mathcal{NP}$.

i) Hvis $Q \in \mathcal{P}$ og $P \propto Q$, så er $P \in \mathcal{P}$.

ii) Hvis $P \in \mathcal{NPC}$ og $P \propto Q$, så er $Q \in \mathcal{NPC}$.

Bevis.

i) Da $P \propto Q$, og der findes en polynomial algoritme til Q, kan vi konvertere enhver instans af P til en instans af Q og løse den indenfor polynomial tid. Derfor kan P også løses inden for polymial tid, dvs. $P \in \mathcal{P}$.

ii) Lad $R \in \mathcal{NP}$. Da $P \in \mathcal{NPC}$, er $R \propto P$. Men da der per antagelse gælder, at $P \propto Q$, kan vi anvende Bemærkning 1.3, så der derfor gælder, at $R \propto Q$. Da $R \in \mathcal{NP}$ var valgt vilkårligt, er $Q \in \mathcal{NPC}$ per definition 1.2.

□

Definition 1.5.
Et optimeringsproblem, for hvilket ja/nej-versionen af problemet ligger i $\mathcal{NPC}$, kaldes $\mathcal{NP}$-hårdt.

Som et eksempel på ovenstående viser vi en sætning om reduktion mellem to kendte $\mathcal{NP}$-hårde problemer ([8], **2**.7f). *Partition*-problemet består i, at vi har en mængde $A = \{a_1, a_2, \ldots, a_n\}$ af n heltal. Opgaven går nu ud på at opdele A i to delmængder A_1 og A_2, sådan at

$$\sum_{a_j \in A_1} a_j = \sum_{a_j \in A_2} a_j = \frac{1}{2} \sum_{j=1}^{n} a_j.$$

Deadline Scheduling-problemet er et specialtilfælde af Job Shop Scheduling med kun én maskine og n jobs. Findes der en rækkefølge af jobbene, så hvert job er færdigt inden for intervallet $[r_j; \bar{d}_j]$, og så hvert job færdiggøres, inden det sendes videre?

Sætning 1.6. *Partition-problemet er polynomielt reducibelt til Deadline Scheduling-problemet.*

Bevis.
Lad $A = \{a_1, a_2, \ldots, a_n\}$ være en instans af *Partition*-problemet. Vi vælger $n+1$ jobs i *Deadline Scheduling*-problemet, hvor de første n jobs svarer til elementerne i A, og job $n+1$ er et ekstra job. For de første n jobs sætter vi $p_{1,j} = a_j$, $r_j = 0$ og $\bar{d}_j = 1 + \sum a_j$. For job $n+1$ sætter vi $p_{1,n+1} = 1$, $r_{n+1} = \frac{1}{2} \sum a_j$ og $\bar{d}_{n+1} = 1 + r_{n+1} = 1 + \frac{1}{2} \sum a_j$.
Vi viser sætningen ved at vise, at vi har en gyldig partition af A hvis og kun hvis vi har en feasible schedule til *Deadline Scheduling*-problemet.

($\Rightarrow$) Antag vi har en gyldig partition af A. Lad $I' \subseteq I = \{1, 2, \ldots, n\}$ være en delindexmængde, så $\sum_{i \in I'} a_i = \frac{1}{2} \sum a_j$. Vi udfører jobbene fra I' på maskinen og er færdige efter $\frac{1}{2} \sum a_j$. Vi udfører derefter job $n+1$, som er færdigt efter $1 + \frac{1}{2} \sum a_j$. Til sidst udfører

vi resten af jobbene $(I \backslash I')$ i intervallet $\left[1 + \frac{1}{2}\sum a_j; 1 + \sum a_j\right]$ i tilfældig rækkefølge. Da dette opfylder kravene for *Deadline Scheduling*-problemet, er den ønskede implikation vist.

($\Leftarrow$) Antag nu, at vi har en feasible schedule til *Deadline Scheduling*-problemet. Vi ser først på job $n + 1$, der må skulle udføres i intervallet $\left[\frac{1}{2}\sum a_j; 1 + \frac{1}{2}\sum a_j\right]$, da schedulen er feasible. Vi har nu to lige store disjunkte intervaller, $\left[0; \frac{1}{2}\sum a_j\right]$ og $\left[1 + \frac{1}{2}\sum a_j; 1 + \sum a_j\right]$, hvor jobbene 1 til n scheduleres, og vi har dermed en partition af A. □

2 Heltalsprogrammering

Vi har i forrige afsnit set på et par binære optimeringsproblemer og lidt om disses formuleringer. Vi skal nu udvide dette til det mere generelle og større område *heltalsprogrammering*, hvor vi i stedet for kun at se på binære variable kræver, at nogle (eller alle) variable skal være heltal. Dette område finder blandt andet anvendelse inden for generel schedulering, specielt også sports scheduling.

Vi ser nu på *lineære problemer* på formen $\max\{c(x) : \ Ax \leq b, \ x \geq 0\}$, hvor c er en n-dimensional rækkevektor (objektfunktionens koefficienter), A er en $m \times n$ matrix (constraint-koefficienterne), b er en m-dimensional søjlevektor (vores right-hand-sides), og x er en n-dimensional søjlevektor (vores variable).

Inden for heltalsprogrammering skelner man mellem forskellige typer problemer, alt efter hvordan tallene i x må se ud. Et *mixed heltalsproblem* (MIP) er et problem, hvor *nogle* men ikke alle af variablene skal være heltallige. Et *heltalsproblem* (IP) er et problem, hvor *alle* variable er heltallige, og til sidst er et *binært heltalsproblem* (BIP) et problem, hvor alle variablene er binære, som f.eks. (JSS) og (TSP) fra afsnit 1.

Som man måske kan se af ovenstående problemformulering, ligner heltalsproblemer lineære problemer til forveksling, altså blot et LP-problem med ekstra begrænsninger. Teorien for de to harmonerer også fint, dog gør disse ekstra begrænsninger løsningsmetoderne for heltalsproblemer mere komplicerede, både mht. fremgangsmåde og regnearbejde. Metoderne til løsning af (IP), (MIP) og (BIP) ligner hinanden, og vi vil i det følgende se nærmere på sådanne metoder. ([23], p3f)

2.1 Relaxeringer

Vi indfører først begreberne polyeder og formulering: ([23], p12ff)

Definition 2.1. *Lad $P \subseteq \mathbb{R}^n$ være beskrevet ved et endeligt antal lineære begrænsninger, altså $P = \{x \in \mathbb{R}^n : \ Ax \leq b\}$ med x, A, b som ovenfor. Da kaldes P et polyeder.*

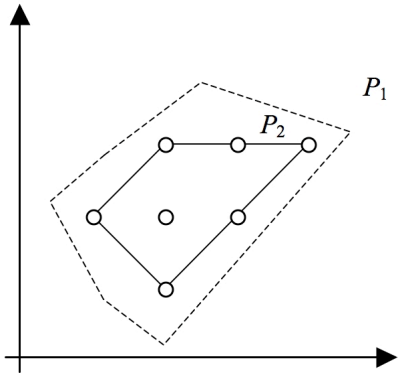
Definition 2.2. *Et polyeder $P \subseteq \mathbb{R}^{n+p}$ kaldes en* formulering *for mængden $X \subseteq \mathbb{Z}^n \times \mathbb{R}^p$, hvis og kun hvis $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$.*

Eksempel 2.3. *Som et eksempel på ovenstående er der i Figur 1 tegnet to formuleringer for mængden $X = \{(1,2),(2,1),(2,2),(2,3),(3,2),(3,3),(4,3)\}$.*

Definition 2.4. *Lad $X \subseteq \mathbb{R}^n$. Det* konvekse hylster*, $conv(X)$, defineres som $conv(X) = \left\{x \in \mathbb{R}^n : \ x = \sum_{i=1}^{t} \lambda_i x^i, \ \sum_{i=1}^{t} \lambda_i = 1, \ \lambda_i \geq 0$ for alle endelige delmængder $\{x^1, \ldots, x^t\} \subseteq X\right\}$.*

Sagt med andre ord, det konvekse hylster er mængden af de punkter, der kan skrives som en konveks kombination af punkter i X. Derudover bemærker vi også, at det konvekse hylster er et polyeder, og at hjørnepunkterne i $conv(X)$ altid ligger i X selv.

Men hvorfor er det nu interessant at se på disse polyedre og konvekse hylstre? Jo, vi har nu teoretisk mulighed for at erstatte vores IP $\max\{c(x) : \ x \in X\}$ med et nyt



Figur 1: To formuleringer for X

lineært problem $\max\{c(x) : x \in conv(X)\}$, som vi har metoder til at løse (f.eks. simplex). I praksis er det dog som regel ekstremt vanskeligt at finde de ofte eksponentielt mange uligheder, der indgår i $conv(X)$, derfor må vi gå ad omveje for at tilnærme vores formulering til $conv(X)$. Hvis man derfor har to formuleringer, P_1, P_2, for $X \subseteq \mathbb{R}^n$, siges P_1 at være bedre end P_2, hvis $P_1 \subset P_2$. Vi ser nu også i Figur 1, at formulering P_2 er bedre end P_1 og endda er lig $conv(X)$.

For at starte et sted klarlægger vi, hvad formålet med at løse vores heltalsproblem $z = \max\{c(x) : \ x \in X \subseteq \mathbb{Z}^n\}$ egentlig er, dvs. hvornår kan vi stille os tilfreds med en løsning til problemet. Vi ønsker altså at bevise, at en given løsning x^* er optimal. Standardmetoden til dette er at finde en nedre grænse $\underline{z} \leq z$ og en øvre grænse $\bar{z} \geq z$, så $\underline{z} = z = \bar{z}$. Ideen er så ved hjælp af algoritmer at finde grænser, der kommer tættere og tættere på z.

Ved maksimeringsproblemer kan der altid findes en nedre grænse ved hjælp af enhver feasible løsning. De øvre grænser kan findes ved et begreb kaldet *relaxering*:

Definition 2.5. *Lad* $z = \max\{c(x) : \ x \in X \subseteq \mathbb{R}^n\}$ (P).
Hvis følgende to betingelser er opfyldt:

i) $X \subseteq T \subseteq \mathbb{R}^n$ *og*

ii) $f(x) \geq c(x)$ *for alle* $x \in X$,

så er $z^R = \max\{f(x) : \ x \in T \subseteq \mathbb{R}^n\}$ (RP) *en* relaxering *af* (P).

Proposition 2.6. *Hvis* (RP) *er en relaxering af* (IP), *så er* $z^R \geq z$.

Bevis. Lad x^* være en optimal løsning til (IP). Så vil $x^* \in X \subseteq T$ og $z = c(x^*) \leq f(x^*)$. Men da $x^* \in T$, er $f(x^*)$ en feasible løsning til (RP) og dermed en nedre grænse for z^R. Derfor har vi, at $z \leq f(x^*) \leq z^R$. □

En typisk relaxering for et heltalsproblem vil være at fjerne selve heltalsbegrænsningen, altså gå fra problemet $z = \max\{c(x) : \ x \in P \cap \mathbb{Z}^n\}$ til det relaxerede problem $z^{LP} = \max\{c(x) : \ x \in P\}$ med $P = \{x \in \mathbb{R}^n : \ Ax \leq b\}$.

Vi skal nu se, hvordan man ved hjælp af relaxeringer kan bevise optimalitet eller infeasibility: ([23], p26)

Proposition 2.7. *Lad* (RP) $z^R = \max\{f(x) : x \in T \subseteq \mathbb{R}^n\}$ *være en relaxering af* (IP) $z = \max\{c(x) : x \in X \subseteq \mathbb{Z}^n\}$ *og lad* x^* *være en optimal løsning til* (RP)*. Da gælder:*

i) Hvis (RP) *er infeasible, så er* (IP) *det også.*

ii) Hvis $x^* \in X$*, og* $f(x^*) = c(x^*)$*, så er* x^* *også optimal for* (IP)*.*

Bevis. Da (RP) er infeasible, er $T = \emptyset$, og dermed er også $X = \emptyset$, hvilket beviser første påstand.
Da $x^* \in X$, er $z \geq c(x^*) = f(x^*) = z^R$. Af Proposition 2.6 har vi, at $z \leq z^R$, derfor er $c(x^*) = z = z^R$, som så er optimal for både (IP) og (RP). □

Ligesom man i lineære problemer kan finde og bruge dualer til løsning af selve problemet, findes disse også i heltalsproblemer: ([23], p28)

Definition 2.8. (IP) $z = \max\{c(x) : x \in X\}$ *og* (D) $w = \min\{\omega(u) : u \in U\}$ *udgør et (svagt) dualt par, hvis* $c(x) \leq \omega(u)$ *for alle* $x \in X$ *og alle* $u \in U$*, og hvis* $z = w$*, udgør de et stærkt dualt par.*

En meget væsentlig egenskab for det duale heltalsproblem er dets evne til at levere en øvre grænse på objektfunktionsværdien, når der findes en feasible solution. Hvis man derfor opskriver det duale til en relaxering af et heltalsproblem og løser dette, får man en øvre grænse til heltalsproblemet. Sagt med andre ord udgør f.eks. problemerne (IP) $z = \max\{c(x) : Ax \leq b, x \in \mathbb{Z}^n_+\}$ og (D) $w = \min\{u(b) : uA \geq c, u \in \mathbb{R}^m_+\}$ et (svagt) dualt par.

Proposition 2.9. *Lad* (IP) *og* (D) *udgøre et svagt dualt par. Da gælder:*

i) Hvis (D) *er ubegrænset, så er* (IP) *infeasible.*

ii) Hvis $x^* \in X$*, og* $u^* \in U$ *og* $c(x^*) = \omega(u^*)$*, så er* x^* *optimal for* (IP) *og* u^* *er optimal for* (D)*.*

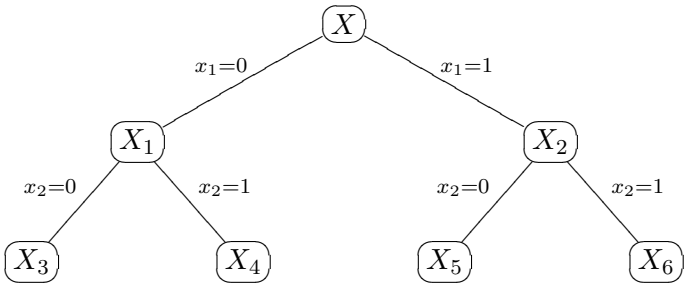
Beviset er næsten analogt til Proposition 2.7:

Bevis. Da (D) er ubegrænset, er er der ingen øvre grænse for (IP), og dermed er (IP) infeasible.
Da $x^* \in X$, og $u^* \in U$, er $w \geq c(x^*) = \omega(u^*)$. Af Definition 2.8 har vi, at $z \leq w$, derfor er $c(x^*) = z = w$, hvilket vil sige, at x^* er optimal for (IP), og u^* er optimal for (D). □

2.2 Branch and Bound

Vi skal nu se på en meget anvendt metode til løsning af heltalsproblemer, nemlig branch-and-bound-metoden. Hovedideen bag metoden er at dele ens problem op i mindre problemer, der er nemmere at løse hver for sig og så samle disse små løsninger til sidst. Fremgangsmåden er først at vælge en variabel, der skal "branches", dvs. deles op i to intervaller, f.eks. en binær variabel i et (BIP), der fixeres til 0 i den ene branch (=nyt

mindre problem) og 1 i den anden. Man forsøger derefter at løse disse to mindre problemer, og hvis man er ude af stand til dette, må man branche en ny variabel. Herunder i Figur 2 ses et eksempel på et binært branch-and-bound-træ.



Figur 2: Eksempel på et binært branch-and-bound-træ.

Vi vil nu gennemgå den algoritme, der bruges til branch-and-bound. Vi starter med selve opdelingen af problemet: ([23], p91ff)

Proposition 2.10. *Lad $X = X_1 \cup \cdots \cup X_K$ være en partition af X og lad $z^k = \max\{c(x) : x \in X_k\}$, $k = 1, \ldots, K$. Så gælder:*

i) $z = \max_k z^k$ *og*

ii) Hvis $\bar{z}^k$ er øvre grænse for z^k, og $\underline{z}^k$ er nedre grænse for z^k, så er $\bar{z} = \max_k \bar{z}^k$ øvre grænse for z, tilsvarende er $\underline{z} = \max_k \underline{z}^k$ nedre grænse for z.

Ovenstående proposition siger blot, at den optimale løsning z kan findes blandt de optimale løsninger til delproblemerne $X_1, \ldots, X_K$, og at øvre/nedre grænser for z er største øvre/nedre grænser for alle delproblemerne.

I branch-and-bound-metoden anvendes såkaldt "pruning" for at eliminere de grene, der af forskellige årsager ikke giver et korrekt eller godt nok resultat. Når man til sidst ikke har flere grene tilbage at branche, har metoden fundet en optimal løsning til problemet (med mindre selvfølgelig problemet er infeasible). Følgende regler kan bruges for at udelukke grene: ([13], p353)

Proposition 2.11. *Grenen i branch-and-bound-træet med mængde X_i kan prunes, hvis én af følgende 3 betingelser er opfyldt:*

1. *Pruning ved infeasibility: $X_i = \emptyset$.*
2. *Pruning ved optimalitet: Lad $x^* \in X_i$ være en optimal løsning til (RP_i), hvor $f(x^*) = c(x^*)$.*
3. *Pruning ved grænser: $\bar{z}_i \leq \underline{z}$.*

Bevis. 1 og 2 følger af Proposition 2.7.
Vi har, at $\bar{z}_i \leq \underline{z}$, og $\bar{z}_i$ er en øvre grænse for X_i. Men da $\bar{z}_i \leq \underline{z}$, er $\bar{z}_i \notin [\underline{z}; \bar{z}]$. □

Det bemærkes, at hvis *2.* er opfyldt, har vi fundet en ny nedre grænse for z. Vi opstiller nu en algoritme, der ved hjælp af branch-and-bound kan løse et $(LP)-$, $(IP)-$, $(MIP)-$ eller $(BIP)-$problem: ([5], p517)

Algoritme 2.12.

Initialisering: *Sæt $\underline{z} = -\infty$. Anvend de tre metoder fra Proposition 2.11 på problemet. Hvis problemet ikke bliver prunet, gå videre til branching. Ellers stop.*

Branching: *Vælg ét blandt de tilbageværende (ikke-prunede) delproblemer. Vælg i dette delproblem den første heltalsvariabel, der i den optimale løsning til LP-relaxeringen af delproblemet ikke antager heltalsværdi som vores branching-variabel. Kald denne variabel x_j og dens optimale relaxerede værdi x_j^*. Lav nu to nye delproblemer med hhv. constraint $x_j \leq \lfloor x_j^* \rfloor$ og $x_j \geq \lfloor x_j^* + 1 \rfloor$ tilføjet.*

Bounding: *Beregn for hvert nyt delproblem den optimale løsning $\bar{z}_i$ til det LP-relaxerede problem.*

Pruning: *Anvend pruning-reglerne fra Proposition 2.11 på hvert nyt delproblem. Hvis Proposition 2.11 (2.) er opfyldt og $x_j^* \geq \underline{z}$, sæt da $\underline{z} = x_j^*$ og gentag Proposition 2.11 (3.) for alle uløste delproblemer.*

Optimalitet: *Stop, hvis der ikke er flere uløste delproblemer. Ellers gå til Branching.*

Eksempel 2.13. *Vi illustrerer branch-and-bound-metoden ved at løse et mindre heltalsproblem. Lad først problemet være givet ved*

$$\begin{aligned} max\ z &= 3x_1 - 3x_2 + 4x_3 - x_4 \quad (IP) \\ s.t. & \\ 2x_1 + 4x_3 &\leq 10 \\ x_1 + x_2 - x_3 &\geq 2 \\ 5x_1 - 4x_2 &\leq 0 \\ -x_1 + 3x_3 - x_4 &\leq 4 \\ x_i \in \mathbb{Z}_+,\ i &= 1,2,3,4. \end{aligned}$$

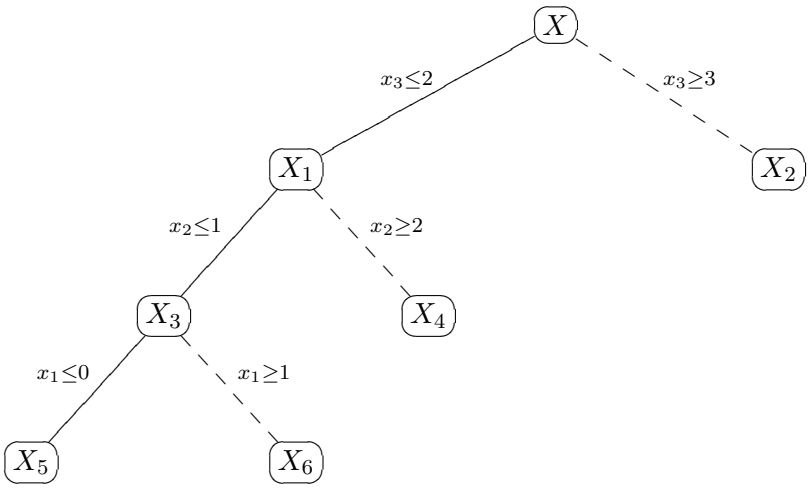
Branch-and-bound-træet til (IP) er vist i Figur 3. Vi relaxerer nu (IP) til dets lineære version (RP), dvs. erstatter $x_i \in \mathbb{Z}_+,\ i = 1,2,3,4$ med $x_i \in \mathbb{R}_+,\ i = 1,2,3,4$, og løser dette, f.eks. vha. simplex. Vi får $x^* = (0, 0, \frac{5}{2}, \frac{7}{2})$ med $z(x^*) = \frac{13}{2}$, som altså ikke er en heltalsløsning. Vi vælger derfor at branche efter den første ikke-heltallige variabel x_3 og får så to nye delproblemer med områder X_1 og X_2 og tilføjet hhv. $x_3 \leq 2$ og $x_3 \geq 3$. Vi løser nu disse to problemers relaxeringer og ser, at delproblemet X_2 er infeasible jfr. Proposition 2.11 (1.), og vi behøver derfor ikke bekymre os mere om denne gren (vi pruner den). For delproblemet X_1 får vi $x_2^* = (1, \frac{5}{4}, 2, 1)$ med $z(x_2^*) = \frac{25}{4}$.

Dette resultat hjælper os ikke videre, derfor er vi nødsaget til at branche problemet igen, denne gang efter x_2, så vi får to nye delproblemer til X_1, nemlig X_3 og X_4 tilføjet hhv. $x_2 \leq 1$ og $x_2 \geq 2$, hvis relaxerede løsninger giver $x_3^* = (\frac{4}{5}, 1, 2, \frac{6}{5})$ med $z(x_3^*) = \frac{31}{5}$ og $x_4^* = (\frac{8}{5}, 2, \frac{17}{10}, 0)$ med $z(x_4^*) = \frac{28}{5}$.

Igen kommer vi ikke til nogen umiddelbare konklusioner, vi vælger derfor at branche X_3 efter variablen x_1 i de to delproblemer X_5 og X_6 tilføjet hhv. $x_1 \leq 0$ (hvilket reelt vil sige, at vi fixer $x_1 = 0$) og $x_1 \geq 1$. X_6 er nu infeasible, dvs. vi igen kan prune denne gren vha. Proposition 2.11 (1.). X_5 har relaxeret løsning $x_5^* = (0, 0, 2, 2)$ med $z(x_5^*) = 6$. Da

denne løsning er heltallig, kan vi vha. Proposition 2.11 (2.) prune grenen X_5 og sætter så $\underline{z} = 6$.

Vi ser nu på tilbageværende uløste grene, og vi kan nu jfr. Proposition 2.11 (3.) prune grenen X_4, da $\frac{28}{5} \leq 6$. Da vi nu ikke har flere uløste grene, får vi, at vores optimale heltalsløsning til (IP) bliver $x_5^* = (0, 0, 2, 2)$ med $z(x_5^*) = 6$.



Figur 3: Branch-and-bound-træ til Eksempel 2.13.

2.3 Preprocessing

Man kan dog, inden man overhovedet går i gang med branch-and-bound-metoden, lave preprocessing, der kan gøre metoden mere effektiv og knap så regnetung. Vi illustrerer med et eksempel:

Eksempel 2.14. *Betragt heltalsproblemet:*

$$\begin{aligned} \max\ z &= 3x_1 - x_2 + 3x_3 \quad (IP) \\ s.t. & \\ x_1 - 2x_3 &\leq 5 \\ 3x_1 - 2x_2 + x_3 &\geq 2 \\ 2x_1 + x_2 - x_3 &\leq 22 \\ 0 \leq x_1 &\leq 12 \\ 0 \leq x_2 &\leq 2 \\ 0 \leq x_3 &\leq 2 \\ x_i \in \mathbb{Z}_+,\ i &= 1, 2, 3. \end{aligned}$$

Skærpe grænser: *Vi vil forsøge at skærpe grænsen for variablen x_1. Vi isolerer x_1 i hver af de tre constraints:*

$$\begin{aligned} x_1 &\leq 5 + 2x_3 \leq 5 + 2 \times 2 = 9 \\ 3x_1 &\geq 2 + 2x_2 - x_3 \geq 2 + 2 \times 0 - 2 = 0 \\ 2x_1 &\leq 22 - x_2 + x_3 \leq 22 - 0 + 2 = 24 \end{aligned}$$

Vi ser nu af ovenstående, at vi kan skærpe grænserne for x_1 *til* $0 \leq x_1 \leq 9$.

Overflødige constraints: *Ved indsættelse af* $x_1 = 9,\ x_2, x_3 = 0$ *i constraints 1 og 2 ses det, at disse ikke kan udelukkes. Dog kan venstresiden i constraint 3 højst blive 20, derfor er denne constraint overflødig (efter de ændrede grænser for* x_1*).*

Fastsætte variable: *Ser vi på* x_3 *i objektfunktionen og constraints, ser vi, at det kan betale sig at fastsætte denne til dens maksimale værdi 2, da dette kun vil være til gavn overalt.*

Vi får nu vores opdaterede problem:

$$\begin{aligned} \max\ z &= 3x_1 - x_2 + 6 \\ s.t.& \\ x_1 &\leq 9 \\ 3x_1 - 2x_2 &\geq 0 \\ 0 \leq x_1 &\leq 9 \\ 0 \leq x_2 &\leq 2 \\ x_i \in \mathbb{Z}_+,\ i &= 1, 2. \end{aligned}$$

Vi bemærker, at problemet nu nærmest er trivielt, idet også den første constraint er overflødig.

Vi sammenfatter ovenstående eksempel i følgende proposition: ([23], p105f)

Proposition 2.15.
Lad $X = \left\{x :\ a_0x_0 + \sum_{j=1}^{n} a_jx_j \leq b, l_j \leq x_j \leq u_j,\ j = 0, \ldots, n\right\}$, *hvor* x_0 *er den undersøgte variabel:*

i) <u>*Skærpe grænser:*</u>

$$x_0 \leq \frac{\left(b - \sum_{j:\ a_j>0} a_jl_j - \sum_{j:\ a_j<0} a_ju_j\right)}{a_0},\ a_0 > 0.$$

$$x_0 \geq \frac{\left(b - \sum_{j:\ a_j>0} a_jl_j - \sum_{j:\ a_j<0} a_ju_j\right)}{a_0},\ a_0 < 0.$$

ii) <u>*Overflødige constraints:*</u> $a_0x_0 + \sum_{j=1}^{n} a_jx_j \leq b$ *er overflødig, hvis*

$$\sum_{j:\ a_j>0} a_ju_j + \sum_{j:\ a_j<0} a_jl_j \leq b.$$

iii) <u>*Infeasibility:*</u> $X = \emptyset$, *hvis*

$$\sum_{j:\ a_j>0} a_jl_j + \sum_{j:\ a_j<0} a_ju_j > b.$$

iv) <u>*Fastsættelse af variable:*</u> *For et maksimeringsproblem*

$$z = \max\{c(x) :\ Ax \leq b, l \leq x \leq u\}$$

gælder, at hvis $a_{ij} \geq 0$ *for alle* $i = 1, \ldots, m$ *og* $c_j < 0$, *så er* $x_j = l_j$.
Modsat gælder, at hvis $a_{ij} \leq 0$ *for alle* $i = 1, \ldots, m$ *og* $c_j > 0$, *så er* $x_j = u_j$.

2.4 Gyldige uligheder

Umiddelbart virker branch-and-bound kombineret med preprocessing som en effektiv metode til at løse selv større heltalsproblemer, men ved problemer med f.eks. *rigtig* mange variable bliver metoden alligevel for tung og langsom, selv for moderne computere. Vi vil derfor i det følgende skitsere alternative og i nogle tilfælde mere effektive metoder til at tilnærme mængdens konvekse hylster ved at tilføje flere constraints, inden vi giver os i kast med branch-and-bound, altså forbedre vores preprocessing.

Vi starter med at definere begrebet *gyldige uligheder*: ([23], p114)

Definition 2.16. *Lad $\pi \in \mathbb{R}^n$ og $\pi_0 \in \mathbb{R}$. Uligheden $\pi x \leq \pi_0$ kaldes en gyldig ulighed for $X \subseteq \mathbb{R}^n$, hvis $\pi x \leq \pi_0$ for alle $x \in X$.*

I Eksempel 2.14 har vi set flere eksempler på sådanne uligheder, vi skynder os derfor videre til at bevise en proposition om disse uligheder mht. lineære problemer: ([23], p117)

Proposition 2.17. *Lad $P = \{x : \ Ax \leq b, x \geq 0\}$. $\pi x \leq \pi_0$ er en gyldig ulighed for P, hvis og kun hvis*

- *der findes $u \geq 0$ med $u \in \mathbb{R}^m$, så $uA \geq \pi$ og $ub \leq \pi_0$, eller*
- *der findes $u \geq 0, v \geq 0$ med $u, v \in \mathbb{R}^m$, så $uA - v = \pi$ og $ub \leq \pi_0$.*

Bevis. Udtrykket $\pi x \leq \pi_0$ er ensbetydende med at sige, at $\max\{\pi x : \ x \in P\} \leq \pi_0$. Opskriver vi det duale til dette problem, får vi $\min\{ub : \ uA \geq \pi, \ u \geq 0\} \leq \pi_0$. Alternativt kan man tilføje en vektor $v \geq 0$, så vi får lighedstegn:
$\min\{ub : \ uA - v = \pi, \ u \geq 0, v \geq 0\} \leq \pi_0$. Da det er velkendt, at det duale til det duale giver det primale, er påstanden bevist. □

For os vil det nu være interessant at få et lignende resultat for heltalsproblemer. Vi bemærker hurtigt det nærmest trivielle faktum, at når $X = \{y \in \mathbb{Z} : \ y \leq b\}$, er $y \leq \lfloor b \rfloor$ en gyldig ulighed for X. Dette faktum vil vi nu bruge til at opstille en algoritme, der kan finde *alle* gyldige uligheder for et heltalsproblem. Denne algoritme kaldes *Chvátal-Gomorys algoritme*: ([23], p119)

Algoritme 2.18. *Lad $X = P \cap \mathbb{Z}^n$ med $P = \{x \in \mathbb{R}^n_+ : \ Ax \leq b\}$, hvor A er en $m \times n$ matrix med søjler $\{a_1, a_2, \ldots, a_n\}$ og lad desuden $u \in \mathbb{R}^m_+$. Da gælder:*

i) Uligheden $\sum_{j=1}^n ua_j x_j \leq ub$ er en gyldig ulighed for P.

ii) Uligheden $\sum_{j=1}^n \lfloor ua_j \rfloor x_j \leq ub$ er en gyldig ulighed for P.

iii) Uligheden $\sum_{j=1}^n \lfloor ua_j \rfloor x_j \leq \lfloor ub \rfloor$ er en gyldig ulighed for X.

Bevis.

i) Da $u \geq 0$ og $\sum_{j=1}^n a_j x_j \leq b$, gælder påstanden.

ii) Denne påstand følger af *i)*, idet det bemærkes, at $x \geq 0$.

iii) Denne påstand følger af *ii*), idet det bemærkes, at $x \in \mathbb{Z}$, og dermed er $\sum_{j=1}^{n} \lfloor ua_j \rfloor x_j \in \mathbb{Z}$.

□

Eksempel 2.19. *Se på heltalsproblemet*

$$\begin{aligned} \max\ z &= x_1 + 2x_2 \quad (IP) \\ s.t. & \\ 5x_1 - x_2 &\leq 12 \\ x_1 + 2x_2 &\geq 5 \\ 3x_1 + 2x_2 &\leq 3 \\ x_j \in \mathbb{Z},\ j &= 1, 2. \end{aligned}$$

Lad P være givet ved vores 3 constraints, lad $X = P \cap \mathbb{Z}^2$ og lad $u = (\frac{1}{2}, \frac{3}{2}, 0)$. Vi anvender Chvátal-Gomorys algoritme til at finde en gyldig ulighed for X:

i) $\sum_{j=1}^{2} ua_j x_j \leq ub \Leftrightarrow 4x_1 + \frac{5}{2}x_2 \leq \frac{27}{2}$ *er en gyldig ulighed for P.*

ii) $\sum_{j=1}^{2} \lfloor ua_j \rfloor x_j \leq ub \Leftrightarrow 4x_1 + 2x_2 \leq \frac{27}{2}$ *er en gyldig ulighed for P.*

iii) $\sum_{j=1}^{2} \lfloor ua_j \rfloor x_j \leq \lfloor ub \rfloor \Leftrightarrow 4x_1 + 2x_2 \leq 13$ *er en gyldig ulighed for X.*

Da $x_j \in \mathbb{Z}^2,\ j = 1, 2$, er venstresiden altid et lige tal, altså kan 13 snævres ned til 12. Ved brug af algoritmen endnu engang med $u = \frac{1}{2}$ vil vi få den gyldige ulighed $2x_1 + x_2 \leq 6$.

Vi vil nu vise en vigtig sætning, der illustrerer brugbarheden af Chvátal-Gomorys algoritme: ([23], p120f)

Sætning 2.20. *Ved brug af Chvátal-Gomorys algoritme et endeligt antal gange kan enhver gyldig ulighed for X findes.*

Vi viser sætningen for et (BIP) ved hjælp af 5 trin.

Bevis. Lad $P = \{x \in \mathbb{R}^n : \ Ax \leq b, 0 \leq x \leq 1\} \neq \emptyset$ og $X = P \cap \mathbb{Z}^n$. Antag nu, at $\pi x \leq \pi_0$ med $\pi \in \mathbb{Z}^n, \pi_0 \in \mathbb{Z}$ er en gyldig ulighed for X.

i) *Uligheden $\pi x \leq \pi_0 + t,\ t \in \mathbb{Z}_+$ er en gyldig ulighed for P for passende valgt t.*
Da vi har en øvre grænse z_{LP} for P, da P er begrænset, sætter vi $t = \lceil z_{LP} \rceil - \pi_0$.

ii) *For passende valgt $M \in \mathbb{Z}$ er uligheden*

$$\pi x \leq \pi_0 + M \sum_{j \in N_0} x_j + M \sum_{j \in N_1} (1 - x_j) \tag{1}$$

en gyldig ulighed for P for enhver partition (N_0, N_1) af $N = \{1, 2, \ldots, n\}$.
Vi kan nøjes med at undersøge, om uligheden er gyldig for alle de kritiske punkter x^* i P og deler op i to tilfælde.
Hvis $x^* \in \mathbb{Z}^n$, er per antagelse $\pi x \leq \pi_0$ opfyldt, og så er (1) også opfyldt.
Hvis $x^* \notin \mathbb{Z}^n$, ser vi på $\sum_{j \in N_0} x_j^* + \sum_{j \in N_1} (1 - x_j^*)$. Da denne kun består af

positive led ($0 \leq x^* \leq 1$), findes et $\alpha > 0$, så $\sum_{j\in N_0} x_j^* + \sum_{j\in N_1}(1-x_j^*) \geq \alpha$ for alle partitioner (N_0, N_1) af $N = \{1, 2, \ldots, n\}$. Vælg nu $M \geq \frac{t}{\alpha}$, så fås, at

$$\pi_0 + M\left(\sum_{j\in N_0} x_j^* + \sum_{j\in N_1}(1-x_j^*)\right) \geq \pi_0 + M\alpha \geq \pi_0 + t.$$

iii) *Lad $\pi x \leq \pi_0 + \tau + 1$ med $\tau \in \mathbb{Z}_+$ være en Chvátal-Gomory-ulighed for X. Så gælder, at*

$$\pi x \leq \pi_0 + \tau + \sum_{j\in N_0} x_j + \sum_{j\in N_1}(1-x_j) \tag{2}$$

er en Chvátal-Gomory-ulighed for X.
Vi anvender Chvátal-Gomory-algoritmen på de to uligheder $\pi x \leq \pi_0 + \tau + 1$ og (1) med $u = \left(\frac{M-1}{M}, \frac{1}{M}\right)$ og x_j^* fra ovenstående. Vi starter med venstresiden og laver Trin 1 og 2:

$$\frac{\pi x(M-1) + \pi x}{M} = \pi x.$$

Højresiden bliver (Trin 1):

$$\frac{(\pi_0+\tau+1)(M-1) + \pi_0 + M\left(\sum_{j\in N_0} x_j^* + \sum_{j\in N_1}(1-x_j^*)\right)}{M}$$

$$= \frac{M(\pi_0+\tau+1) - \pi_0 - \tau - 1 + \pi_0 + M\left(\sum_{j\in N_0} x_j^* + \sum_{j\in N_1}(1-x_j^*)\right)}{M}$$

$$= \pi_0 + \tau + 1 + \frac{-\tau-1}{M} + \sum_{j\in N_0} x_j^* + \sum_{j\in N_1}(1-x_j^*)$$

Vi udfører til sidst Trin 3 på højresiden, idet vi bemærker, at $-1 \leq \frac{-\tau-1}{M} \leq 0$ for passende valgt M er det eneste ikke-heltallige led:

$$\pi_0 + \tau + 1 - 1 + \sum_{j\in N_0} x_j^* + \sum_{j\in N_1}(1-x_j^*) = \pi_0 + \tau + \sum_{j\in N_0} x_j^* + \sum_{j\in N_1}(1-x_j^*)$$

Altså er (2) en Chvátal-Gomory-ulighed for X.

iv) *Lad (T_0, T_1) være en partition af mængden $\{1, 2, \ldots, p-1\}$. Hvis*

$$\pi x \leq \pi_0 + \tau + \sum_{j\in T_0\cup\{p\}} x_j + \sum_{j\in T_1}(1-x_j) \tag{3}$$

og

$$\pi x \leq \pi_0 + \tau + \sum_{j\in T_0} x_j + \sum_{j\in T_1\cup\{p\}}(1-x_j) \tag{4}$$

er Chvátal-Gomory-uligheder for X, så er

$$\pi x \leq \pi_0 + \tau + \sum_{j\in T_0} x_j + \sum_{j\in T_1}(1-x_j) \tag{5}$$

også en Chvátal-Gomory-ulighed for X.
Vi anvender Chvátal-Gomory-algoritmen på (3) og (4) med $u = \left(\frac{1}{2}, \frac{1}{2}\right)$. Venstresiden giver trivielt πx efter udført Trin 1. Vi regner derfor på højresiden (Trin 1):

$$\pi_0 + \tau + \frac{1}{2}\left(x_p + \sum_{j\in T_0} x_j + \sum_{j\in T_1}(1-x_j) + \sum_{j\in T_0} x_j + (1-x_p) + \sum_{j\in T_1}(1-x_j)\right)$$

$$= \pi_0 + \tau + \sum_{j\in T_0} x_j + \sum_{j\in T_1}(1-x_j) + \frac{1}{2}$$

Vi udfører til sidst Trin 3, dvs. runder udtrykket ned og får så det ønskede resultat:

$$\pi x \leq \pi_0 + \tau + \sum_{j\in T_0} x_j + \sum_{j\in T_1}(1-x_j).$$

v) *Hvis $\pi x \leq \pi_0 + \tau + 1$ er en Chvátal-Gomory-ulighed for X, så er $\pi x \leq \pi_0 + \tau$ det også.*
Vi anvender iv) successivt for $p = n, n-1, \ldots, 0$ for alle partitioner (T_0, T_1) af $\{1, 2, \ldots, p-1\}$ og finder, at $\pi x \leq \pi_0 + \tau$.

Vi kan nu anvende v), idet vi starter med $\tau = t - 1$ og lader denne aftage mod 0 ($\tau = t-1, \ldots, 0$), hvorved vi får, at $\pi x \leq \pi_0$ er en Chvátal-Gomory-ulighed.
Det er nu blevet vist, at enhver gyldig ulighed for X i 0-1-tilfældet kan findes ved hjælp af Chvátal-Gomorys algoritme.

□

2.5 Cutting Plane-algoritmer

Vi har i det foregående fået opstillet en metode til at finde gyldige uligheder for en mængde X. Vi vil nu anvende disse uligheder i heltalsproblemer til at indsnævre vores formulering af problemet ved at tilføje gyldige uligheder (constraints). Men hvorfor så ikke udelukkende lave preprocessing? Preprocessing går ofte for hurtigt i stå, især ved større problemer, hvor forholdene skal være væsentlig mere gunstige for at kunne f.eks. fixe en variabel.

Det store problem bliver hurtigt antallet af constraints, hvilket gør problemet meget stort, og det tager dermed lang tid at løse. Problemet kan også blive for stort til branch-and-bound-software, og vi er derfor ikke interesserede i at finde *mange* uligheder, men de *bedste*. Til dette formål anvendes såkaldte Cutting Plane-algoritmer, som vi vil gennemgå i det følgende. ([23], p123f)

Lad os antage, at vi har en familie $\mathcal{F}$ af gyldige uligheder $\pi x \leq \pi_0,\ (\pi, \pi_0) \in \mathcal{F}$ for mængden $X = P \cap \mathbb{Z}^n$, og lad problemet (IP) være givet ved $\max\{c(x) :\ x \in X\}$. Vi får brug for følgende definition: ([23], p89)

Definition 2.21. *Et* separationsproblem *er defineret på følgende måde: Hvis $x_t \notin conv(X)$, find da en ulighed gældende for alle punkter i X, der skærer x_t væk fra X.*

Følgende algoritme er (i teorien) i stand til at finde de "brugbare"gyldige uligheder i $\mathcal{F}$ ([23], p124f):

Algoritme 2.22.

Initialisering: *Fastsæt $t = 0$ og $P_0 = P$.*

Trin 1: *Løs det relaxerede problem $\bar{z}^t = \max\{c(x) : \ x \in P_t\}$, og lad x^t være den optimale løsning. Hvis $x^t \in \mathbb{Z}^n$ stop; x_t er en optimal løsning til (IP). Ellers gå til Trin 2.*

Trin 2: *Hvis $x^t \notin \mathbb{Z}^n$, løs da separationsproblemet for x^t og familien af gyldige uligheder $\mathcal{F}$. Hvis der findes en ulighed $(\pi^t, \pi_0^t) \in \mathcal{F}$ med $\pi^t x > \pi_0$, der skærer x^t væk fra X, sæt da $P_{t+1} = P_t \cap \{x : \ \pi^t x \leq \pi_0^t\}$ og sæt $t = t + 1$. Findes en sådan ulighed ikke, stoppes algoritmen.*

Hvis algoritmen stopper uden at have fundet en heltalsløsning til (IP), er formuleringen $P_{t+1} = P_t \cap \{x : \ \pi^i x \leq \pi_0^i, \ i = 1, \ldots, t\}$ en bedre formulering end P.

Gomory fandt en stærkt forbedret algoritme, der er knap så regnetung. Ideen i algoritmen bygger på Chvátal-Gomorys algoritme. Lad problemet (IP) være givet ved $\max\{c(x) : \ Ax = b, \ x \in \mathbb{Z}_+\}$:

Algoritme 2.23.

Trin 1: *Løs det relaxerede problem til (IP) (f.eks. med simplex) og kald den optimale løsning x^*.*
Hvis $x^* \in \mathbb{Z}^m$, da er x^* en optimal løsning til (IP), og algoritmen stoppes.

Trin 2: *Omskriv (IP) på formen*

$$\begin{aligned} max \ z = \bar{a}_{00} \ + \sum_{j \in IB} \bar{a}_{0j} x_j \\ x_{B_u} + \sum_{j \in IB} \bar{a}_{uj} x_j \ = \bar{a}_{u0}, \ u = 1, \ldots, m \quad (*) \\ x \ \in \mathbb{Z}_+ \end{aligned}$$

hvor $\bar{a}_{00}$ er den relaxerede værdi hørende til x^*, IB er mængden af ikke-basisvariable, $-\bar{a}_{0j}$ er reduced costs, altså $\bar{a}_{0j} \geq 0$, og hvor $\bar{a}_{u0} \geq 0$ er vores right-hand-sides.

Trin 3: *Da $x^* \notin \mathbb{Z}^m$, findes mindst én $\bar{a}_{u0} \notin \mathbb{Z}$. Vælges denne række, og udføres Chvátal-Gomorys algoritme på den, fås uligheden*

$$x_{B_u} + \sum_{j \in IB} \lfloor \bar{a}_{uj} \rfloor x_j \leq \lfloor \bar{a}_{u0} \rfloor \Leftrightarrow \sum_{j \in IB} (\bar{a}_{uj} - \lfloor \bar{a}_{uj} \rfloor) x_j \geq \bar{a}_{u0} - \lfloor \bar{a}_{u0} \rfloor.$$

(ved at substituere x_{B_u} og indsætte i (*) fås højresiden), eller skrevet kortere: $\sum_{j \in IB} f_{uj} x_j \geq f_{u0}$ med $f_{uj} = \bar{a}_{uj} - \lfloor \bar{a}_{uj} \rfloor$ og $f_{u0} = \bar{a}_{u0} - \lfloor \bar{a}_{u0} \rfloor$.

Trin 4: *Lav den nye constraint*

$$s = -f_{u0} + \sum_{j \in IB} f_{uj} x_j,$$

med $s \in \mathbb{Z}_+$.

Vi bemærker, at vi i Trin 3 rent faktisk finder en ulighed, der skærer x^* væk fra X, og at Trin 4 er nødvendigt for, at vores constraint passer ind i vores problem, der kun består af ligheder. Vi får derved en ekstra (slack-)variabel.

Algoritmen kan anvendes flere gange i træk, og man vil derved tilnærme sig det konvekse hylster.

Eksempel 2.24. *Se på heltalsproblemet*

$$\begin{aligned} max\ z &= 8x_1 + 7x_2 \quad (IP) \\ s.t. & \\ 7x_1 &\leq 6 \\ x_1 + 6x_2 &\geq 10 \\ 7x_1 + 2x_2 &\leq 15 \\ x_i \in \mathbb{Z}_+,\ i &= 1, 2. \end{aligned}$$

1. *Vi tilføjer slack- og surplusvariable, løser det relaxerede problem og får den optimale løsning* $x^* = \left(\frac{6}{7}, \frac{32}{21}, 0, 0, \frac{125}{21}\right) \notin \mathbb{Z}_+$.

2. *Vi omskriver på den ønskede form:*

$$\begin{array}{rrrrrrr} max\ z = \frac{368}{21} & & & +\frac{41}{42}x_3 & +\frac{7}{6}x_4 & & \\ & x_1 & & +\frac{1}{7}x_3 & & = & \frac{6}{7} \\ & & x_2 & -\frac{1}{42}x_3 & +\frac{1}{6}x_4 & = & \frac{32}{21} \\ & & & -\frac{20}{21}x_3 & -\frac{1}{3}x_4 \quad +x_5 & = & \frac{125}{21} \end{array}$$

$x_i \in \mathbb{Z}_+,\ i = 1, 2 \ldots 5.$

3. *Da* $\frac{125}{21} \notin \mathbb{Z}_+$, *vælger vi at udføre trin 3 i Gomorys algoritme på række 3:*

$$-\frac{20}{21}x_3 - \frac{1}{3}x_4 \geq \frac{125}{21} - 5 = \frac{20}{21}.$$

4. *Dermed er*

$$s = -\frac{20}{21} - \frac{20}{21}x_3 - \frac{1}{3}x_4$$

vores nye constraint.

3 CP og Benders Decomposition

Heltalsprogrammering, som det er blevet gennemgået i foregående afsnit, virker umiddelbart som en farbar vej for at løse praktiske problemer, og det er det i mange tilfælde også. Der kan dog opstå situationer, hvor heltalsprogrammering ikke er effektiv nok til at løse problemerne, typisk hvor LP-relaxeringerne er svage. Blandt andet inden for sports scheduling opstår denne situation. Her må IP suppleres med andre metoder.

Vi vil i dette afsnit først gennemgå *constraint programming* (CP), der i samspil med heltalsprogrammering kan løse nogle af de mere komplicerede problemer. Vi skal også se på *Benders decomposition*, der kan bruges som en metode til at generere uligheder for at styrke LP-relaxeringen. Dette vil vi anvende som redskab i vores eksempel med SAS-ligaen.

3.1 Constraint Programming

Som vi har set flere steder, kan heltalsproblemer ofte blive store (mange constraints og variable). Constraint programming (CP) er en metode til at formulere problemer på, så de dels fylder mindre, men primært kan CP bruges til at generere logiske deduktioner, der kan begrænse løsningsmængden. Dette gøres ved at tillade og anvende flere typer af constraints fremfor udelukkende at bruge lineære uligheder, der anvendes i heltalsprogrammering. Af de mere væsentlige kan nævnes ([5], p527ff):

- *Lineære constraints*, såsom $3x_1 - 2x_2 \geq 4$. Disse kender vi f.eks. fra heltalsprogrammering.
- *Adskillende constraints*, som eksempelvis i (JSS), hvor opgave $o_{i_k j}$ skal være afsluttet, før opgave $o_{i_{k+1} j}$ kan starte.
- *Relaterende constraints*, som at der f.eks. skal udføres mindst 3 opgaver på en bestemt maskine i et scheduleringsproblem.
- *Eksplicitte constraints*, som f.eks. at selvom $x, y \in \mathbb{R}$, så skal $(x, y) \in \mathbb{Z}^2$.
- *Unære constraints*, der er constraints på enkelte variable, f.eks. $-2 \leq x \leq 2$.
- *Logiske constraints*, som f.eks. HVIS $x_1 = 2$ OG $x_2 = 3$, SÅ er $x_3 = 7$.

De umiddelbare fordele ved ovenstående typer er det meget større udvalg af logiske og matematiske udsagn, f.eks. de gængse HVIS og OG. Disse er implementeret i næsten alle programmeringssprog, og udsagn kan dermed ofte skrives meget mere kompakt.

I dette afsnit vil vi se på såkaldte *constraint satisfaction problems* (CSP), der kan opstilles generelt på følgende måde ([4], p4f):

Et CSP er et problem af typen $\mathcal{P} = (X, D, C)$, hvor $X = \{x_1, \ldots, x_n\}$ er en endelig mængde af variable, $D = \{D_{x_1}, \ldots, D_{x_n}\}$ er domænerne for de enkelte variable, og C er mængden af constraints, der skal være opfyldt. Dvs. ethvert element $c \in C$ kan skrives som en mængde: $c \subseteq D_{x_1} \times \cdots \times D_{x_n}$. Disse constraints behøver ikke nødvendigvis udtale sig om alle variable på én gang. Problemet løses ved at tildele hver variabel x_i i X en

værdi $s_{x_i} \in D_{x_i}$, $i = 1, \ldots, n$, så alle constraints er opfyldt. Mængden af alle løsninger til $\mathcal{P}$ skrives $sol(\mathcal{P})$.

En af de generelle fremgangsmåder til at løse kombinatoriske problemer, som f.eks. inden for sports scheduling, er følgende ide: opstil først problemet på formen $\mathcal{P} = (X, D, C)$ og find derefter $\mathcal{P}' = (X, D, C')$, hvor $sol(\mathcal{P}') = sol(\mathcal{P})$. Vi udvider nu $\mathcal{P}'$ til to nye problemer, hvor vi har tilføjet en ny constraint c' til det ene og dens negation $\neg c'$ til det andet, så vi får problemerne $\mathcal{P}'_1 = (X, D, C' \cup \{c'\})$ og $\mathcal{P}'_2 = (X, D, C' \cup \{\neg c'\})$. Til disse problemer finder man igen nye constraints c'' og fortsætter således. Dette giver os et binært *søgetræ*, hvor enderne (bladene) enten giver ingen løsninger (et infeasible problem) eller problemer, hvor D kun består af singletons, hvilket vil sige, at vi har fundet en løsning.

En anden metode er *propagation*-teknikker, der indskrænker domænerne i $\mathcal{P}$, så vi opnår et nyt problem $\mathcal{P}' = (X, D', C)$, hvor der for hvert $x \in X$ gælder, at $D'_x \subseteq D_x$, og $sol(\mathcal{P}) = sol(\mathcal{P}')$.

Vi vil ikke gå nærmere i detaljer med constraint programming her, men vil se på tre vigtige og meget brugte constraints inden for sports scheduling, der illustrerer forskellen på constraint og heltalsprogrammering, nemlig sequence, one-factor og all-different.

Først ser vi på sequence, der er på formen

$$\text{sequence}(\texttt{Min}, \texttt{Max}, \texttt{width}, \texttt{vars}, \texttt{values}, \texttt{card}).$$

Variablene i `vars` kan antage værdier fra vektoren `values`, og det i'te element `card[i]` i vektoren `card` angiver så, hvor mange af variablene, der antager værdien `values[i]`. Desuden skal der gælde, at for enhver sammenhængende delsekvens af variable af længde `width` skal mindst `Min` og højst `Max` variable antage værdien `values[i]`. ([16], p6)

Eksempel 3.1. *Vi ser på følgende sequence-constraint:*

$$\text{sequence}(1, 2, 3, [x_1, x_2, \ldots, x_{12}], [0, 1], [6, 6]).$$

Her skal der gælde, at halvdelen (6) af variablene $x_1, x_2, \ldots, x_{12}$ skal antage værdien 0 og resten (6) værdien 1. Desuden skal der gælde, at for enhver delsekvens af længde 3, skal mindst 1 og højst 2 af variablene x_i antage værdien hhv. 0 eller 1.

Inden for kombinatoriske problemer ser man ofte problemerne som grafteoretiske problemer, hvor vores variable x_i er knuder i en graf, og $v_i = j$ er en kant mellem knude i og j. Vi ser nu på one-factor, som er på formen

$$\begin{aligned}\text{one-factor}(x_1, \ldots, x_n) = \quad & \{(v_1, \ldots, v_n) \in D_{x_1} \times \cdots \times D_{x_n} \mid \\ & \forall i, j : \; v_i \neq i \wedge (v_i = j \Leftrightarrow v_j = i).\}\end{aligned}$$

Altså danner vi par af knuder i vores graf. Vi vender tilbage til one-factor i afsnit 4.3, hvor vi vil se nærmere på dens egenskaber inden for sports scheduling.

Den sidste constraint, vi skal se på, skrives

$$\begin{aligned}\text{all-different}(x_1, \ldots, x_n) = \quad & \{(v_1, \ldots, v_n) \in D_{x_1} \times \cdots \times D_{x_n} \mid \\ & \forall i, j, i \neq j : \; v_i \neq v_j\}\end{aligned}$$

og betyder i al sin enkelhed, at variablene $x_1, x_2, \ldots, x_n$ alle skal antage forskellige værdier, altså kan to kanter ikke ende i samme knude. Her skal det bemærkes, at hver variabel har sit eget domæne, den lever på. Hvis man f.eks. opfatter $x_1, x_2, \ldots, x_n$ som opgaver, der skal udføres af n maskiner, der alle skal udføre netop én opgave hver, så vil f.eks. $x_3 = 7$ betyde, at opgave 3 skal løses af maskine 7. ([4], p5)

Eksempel 3.2. *Se igen på eksemplet med (TSP) fra før og constrainten*

$$\sum_j x_{ij} = 1, \quad \textit{og} \quad \sum_i x_{ij} = 1, \ i, j = 1, \ldots, n.$$

Vi har nummereret byerne $1, \ldots, n$*. Lad nu* y_i *være den by, som sælgeren ankommer til efter by nummer* i*. Da han skal ankomme til alle byer netop én gang, kan vi anvende* all-different$(y_1, y_2, \ldots, y_n)$*. Før havde vi* n^2 *variable og* $2n$ *constraints, nu har vi én constraint og* n *variable.*

Vi vil nu vise en enkelt sætning om denne specielle all-different constraint ([7], p196f). I beviset får vi brug for D_k, der er domænet hørende til variablen y_k og mængden $D(J)$, hvor J er en delmængde af variablene, og $D(J)$ er foreningsmængden af domænerne hørende til variablene i J. Vi skal også bruge et skema T, der repræsenterer tildelingen af værdier til variablene. Søjlerne i skemaet er de forskellige variable, og rækkerne er elementerne i mængden $\bigcup_{i=1,\ldots,n} D_i$.

Sætning 3.3. all-different$(y_1, y_2, \ldots, y_n)$ *har en løsning, hvis og kun hvis der for ethvert* $k \in \{1, 2, \ldots, n\}$ *gælder, at størrelsen af domænet hørende til enhver delmængde med* k *variable, er mindst* k*.*

Bevis. ($\Rightarrow$) Antag, at vi har en løsning til all-different$(y_1, y_2, \ldots, y_n)$. Altså vil hvert y_i have en bestemt værdi, forskellig fra y_j, $i \neq j$. Så vil der for enhver delmængde J med k variable gælde, at $|D(J)| \geq k$.

($\Leftarrow$) Antag, at der for ethvert $k \in \{1, 2, \ldots, n\}$ gælder, at størrelsen af domænet hørende til enhver delmængde med k variable er mindst k. Vi viser påstanden ved induktion efter antallet af variable og bemærker straks, at tilfældet $k = 1$ er trivielt. Altså kan induktionen starte. Antag nu, at påstanden gælder for $k-1$ variable. Vi vil vise ved kontraposition, at påstanden så også gælder for k variable.

Vi antager altså, at der ikke eksisterer en løsning til all-different$(y_1, y_2, \ldots, y_k)$. Da der per induktionsantagelsen gælder, at der findes en løsning til all-different$(y_1, y_2, \ldots, y_{k-1})$, gælder der for ethvert $i \in D_k$, at vi ikke kan tildele y_k nogen værdi, altså så $y_k = i$.

Vi opskriver skemaet T' for ethvert $i \in D_k$, der fremkommer fra T ved at slette den k'te søjle og den i'te række. T har ingen løsninger jævnfør kontrapositionsantagelsen, dvs. $|D(J_i)| < |J_i|$ i T. T' kan ikke have en løsning, da vi ved at fjerne række i og søjle k får trukket 1 fra på venstresiden og højst 1 på højresiden, derfor gælder uligheden stadig.

Per induktionsantagelsen findes der i T' en delmængde J_i af $\{y_1, y_2, \ldots, y_{k-1}\}$ med $|D(J_i)| < |J_i|$. Ved igen at gå fra T' til T, mens vi stadig ser på $J_i \subseteq \{y_1, y_2, \ldots, y_{k-1}\}$, vil vi få, at $|D(J_i)| \leq |J_i|$, altså er $i \in D(J_i)$, da vi ellers ingen løsning ville have for $k-1$ variable (som induktionsantagelsen siger, der er).

Vi har, at $|D(J_i)| \leq |J_i|$ gælder for ethvert $i \in D_k$, altså vil den specielt også gælde for $\bigcup_{i \in D_k} J_i$:

$$\left| \bigcup_{i \in D_k} D(J_i) \right| \leq \left| \bigcup_{i \in D_k} J_i \right|.$$

Men da $i \in D(J_i)$ for ethvert $i \in D_k$, får vi, at

$$\left| D_k \cup \bigcup_{i \in D_k} D(J_i) \right| = \left| \bigcup_{i \in D_k} D(J_i) \right| \leq \left| \bigcup_{i \in D_k} J_i \right| < \left| \{y_k\} \cup \bigcup_{i \in D_k} J_i \right|.$$

Dermed gælder der for ethvert $k \in \{1, 2, \ldots, n\}$, at størrelsen af domænet hørende til enhver delmængde med k variable, er mindre end k, hvilket giver det ønskede, og sætningen er bevist. □

Ovenstående sætning er et vigtigt redskab til at reducere domæner for variable i constraint programming. Man finder først alle delmængder J af variable med $|D(J)| \leq |J|$. Hvis $|D(J)| < |J|$, findes ingen tildeling. Ellers kan i udelades fra domænet D_j, hvis og kun hvis $i \in D(J)$ for en delmængde J af variable, hvorom der gælder, at $|D(J)| = |J|$, og $y_j \notin J$.

Vi har nu illustreret et par af forskellene ved constraint programming sammenlignet med heltalsprogrammering. Generelt vil der i problemer skulle udføres 3 trin:

1. at formulere en model for problemet (gerne så kompakt som muligt) og bruger en del forskellige constraints
2. effektivt at finde feasible løsninger, der opfylder disse constraints, og
3. at finde den optimale løsning blandt disse feasible løsninger.

Som vi har vist i ovenstående afsnit, er constraint programming typisk mere effektiv i de første punkter, sammenlignet med heltalsprogrammering, hvorimod den taber til heltalsprogrammering i nummer 3. Det ville derfor være oplagt at anvende constraint programming på 1 og 2 og heltalsprogrammring på 3, men dette er stadig et varmt emne inden for forskningen. Problemerne ligger mestendels i at omskrive mellem de to formuleringstyper, uden at problemet eksploderer i størrelse. Ovenstående problemer lægger grund for meget forskning i dag. ([5], p528ff)

3.2 Benders Decomposition

Når vi senere skal schedulere SAS-ligaen, vil vi anvende dele af en teknik, der kaldes *Logic-based Benders decomposition*. Denne teknik går i hovedtræk ud på at fastsætte visse variable i ens hovedproblem, hvorved der dannes et underproblem. Dette underproblem løses typisk indirekte via dets såkaldte *inference dual*, hvorved der genereres et *Benders cut*, der er med til at udelukke en stor del af de mulige fastsættelser af variable. Dette fortsætter man med, indtil man ikke længere kan finde et bedre cut. Generelt kan man sige, at metoden "lærer af sine egne fejl". Vi vil ikke gå i dybden med al teorien omkring Benders decomposition men i stedet prøve at illustrere ideen og vise en central

sætning, omhandlende hvorfor teknikken virker. ([6], p1ff)

Vi vil først se på det klassiske (lineære) tilfælde for at illustrere metoden i kendte termer. Vi ser på et generelt lineært minimeringsproblem, hvor variablene er delt op i to grupper, $x \in \mathbb{R}^n$ og $y \in D_y$ (vores hovedproblem). Vi fastsætter nu variablene i y til værdierne $\bar{y}$, og opnår dermed et delproblem:

$$\begin{aligned} \min\ z &= cx + f(\bar{y}) \\ \text{s.t.} & \\ Ax &\geq a - g(\bar{y}) \\ x &\geq 0, \end{aligned}$$

hvor $f(\bar{y})$ og $g(\bar{y})$ så bliver konstanter. Det duale til dette problem bliver

$$\begin{aligned} \max\ \beta &= u(a - g(\bar{y})) + f(\bar{y}) \\ \text{s.t.} & \\ uA &\leq c \\ u &\geq 0. \end{aligned}$$

Her lader vi β^* være den optimale værdi af objektfunktionen og bemærker, at da $\bar{y}$ ikke indgår i begrænsningerne, vil løsningen til ovenstående problem ($\bar{u}$) altid være en feasible løsning til det oprindelige problem for enhver fastsættelse $\bar{y}$. Altså vil vores delproblem altid have en feasible løsning, hvis vores hovedproblem har det.

Hvis $\bar{u}$ er en brugbar (endelig) løsning til det duale problem, giver dette os en brugbar løsning $\bar{x}$ til det primale problem, ligesom vi får en grænse på dets objektfunktion, et såkaldt *Benders cut*:

$$z \geq \beta_{\bar{y}}(y) = \bar{u}(a - g(y)) + f(\bar{y}),\ \beta_{\bar{y}}(\bar{y}) = \beta^*,$$

som kan vises at være gyldigt for *alle* y, da $\bar{u}$ er uafhængig af vores valg af y. ([6], p40)

I det generelle *logisk-baserede* Benders decomposition, bliver tingene en smule anderledes. Vi ser på det generelle problem

$$\begin{aligned} \min\ z &= c(x, y) \\ \text{s.t.} & \\ (x, y) &\in S \\ x \in D_x\ &,\ y \in D_y. \end{aligned} \tag{6}$$

Her er S en begrænset mængde, f.eks. repræsenteret ved hjælp af en matrix, og D_x og D_y er domænerne til de to *vektorer* af variable, x og y. Måden, disse variable deles op i, kan f.eks. være en opdeling i binære og heltalsvariable, eller hvad der nu måtte være passende. Vi laver nu vores delproblem ved ligesom i det klassiske tilfælde at fixere den ene vektor med variable y til værdierne $\bar{y}$ og får så delproblemet

$$\begin{aligned} \min\ z &= c(x, \bar{y}) \\ \text{s.t.} & \\ (x, \bar{y}) &\in S \\ x &\in D_x. \end{aligned} \tag{7}$$

Det antages her, at (7) er feasible, men (7) kan godt være unbounded.

Vi vælger nu ikke at løse dette problem direkte, men ser i stedet på dets *inference dual*:

$$\begin{aligned} \max \quad & \beta \\ \text{s.t.} \quad & \\ (x, \bar{y}) &\in S \\ c(x, \bar{y}) &\geq \beta \\ x &\in D_x. \end{aligned} \tag{8}$$

Ligesom i det klassiske tilfælde vil vi forsøge at konstruere et Benders cut. Vi laver cuttet $z \geq \beta_{\bar{y}}(y)$ med $\beta_{\bar{y}}(\bar{y}) = \beta^*$, hvor β^* igen er værdien af objektfunktionen i vores inference dual. Der findes ikke en generel metode til at finde funktionen $\beta_{\bar{y}}(y)$ for alle slags problemer, men kernen i metoden er at finde den, så den kan give os en nedre grænse for objektfunktionen i (6) uanset valget af $\bar{y}$. Dette er altså ikke nogen fast grænse, men en grænse der er afhængig af y, vi skal altså finde et y, så objektfunktionen i (6) minimeres. Løsningen til dette kalder vi $\bar{y}^2$, og ud fra denne værdi ser vi igen på (8) og får genereret et nyt cut. Igen skal vi finde et y, der minimerer objektfunktionen i (6). Problemet med at minimere objektfunktionen opstiller vi som et selvstændigt problem, hvor vi løbende tilføjer de cuts, der er blevet genereret:

$$\begin{aligned} \min \quad & z \\ \text{s.t.} \quad & \\ z &\geq \beta_{\bar{y}^k}(y) \quad , \quad k = 1, \ldots, n \\ y &\in D_y, \end{aligned} \tag{9}$$

hvor $\bar{y}^1, \ldots, \bar{y}^n$ er de fastsatte værdier af y fra iteration $1, \ldots, n$. Så længe (8) kan producere et nyt cut, der forbedrer værdien af vores master problem, fortsætter proceduren. ([6], p37f)

Ovenstående kan nu samles i en algoritme: ([6], p38f)

Algoritme 3.4.

Vælg *$\bar{y} \in D_y$ fra (6) og sæt $\bar{z} = -\infty$ og $k = 0$.*

Mens *(8) har en feasible løsning, der opfylder, at $\beta > \bar{z}$:*

> *Dan ud fra β en funktion $\beta_{\bar{y}}(y)$ med $\beta_{\bar{y}}(\bar{y}) = \beta$ og $z \geq \beta_{\bar{y}}(y)$.*
> *Lad $k = k + 1$, lad $y^k = \bar{y}$ og tilføj Benders cuttet $z \geq \beta_{y^k}(y)$ til (9).*
>
> **Hvis** *(9) er infeasible, STOP. Da er (6) også infeasible.*
>
> **Ellers** *lad $\bar{y}$ være optimal løsning til (9) med optimal værdi $\bar{z}$.*

(6) har optimal værdi $\bar{z}$.

Vi vil nu bevise de tilfælde, hvor algoritmen stopper:

Sætning 3.5. *Antag i hver iteration af Benders algoritme, at den begrænsende funktion $\beta_{\bar{y}}(y)$ opfylder følgende:*
Enhver feasible løsning (x, y) til (6) opfylder, at $c(x, y) \geq \beta_{\bar{y}}(y)$.
Så gælder:

i) Hvis algoritmen stopper med $(\bar{z}, \bar{y})$ som endelig løsning til (9), så har (6) optimal løsning $(\bar{x}, \bar{y})$ med $c(\bar{x}, \bar{y}) = \bar{z}$.

ii) Hvis algoritmen stopper med et infeasible masterproblem, så er (6) infeasible.

iii) Hvis algoritmen stopper med et delproblem, hvor det duale (8) er infeasible, da er (6) ubegrænset.

Bevis.

i) Vi lader β^* være den sidst opnåede optimale løsning til (8). Da (7) og (8) har samme optimale løsning, vil β^* også være optimal for (7) med en tilhørende optimal løsning $\bar{x}$. Enhver feasible løsning til (6) med værdien $\bar{z}$ er optimal i (6), da $\bar{z}$ opfylder $c(x, y) \geq \beta_{\bar{y}}(y)$. Da vi ikke kunne finde noget bedre y til vores masterproblem, har vi en optimal løsning til (6).

ii) Da der for alle Benders cuts gælder, at enhver feasible løsning (x, y) til (6) opfylder, at $c(x, y) \geq \beta_{\bar{y}}(y)$, vil et infeasible masterproblem (9) medføre, at også hovedproblemet (6) bliver infeasible.

iii) Da (8) er infeasible, er der ikke nogen nedre grænse for (7), og dermed har vi fundet et tilfælde, hvor der heller ikke er nogen nedre grænse for (6), og den er dermed ubegrænset.

□

I stedet for at lave et masterproblem kan man i visse tilfælde tilføje sine cuts direkte i hovedproblemet. Dette vil være tilfældet i vores senere gennemgang af SAS-ligaen. Men inden vi kommer så langt, vil vi i næste sektion gennemgå noget generel teori inden for sports scheduling.

4 Sports Scheduling

Vi har i de foregående afsnit fået opbygget en håndfuld nyttige værktøjer: kompleksitetsanalyse, heltalsprogrammering, constraint programming og Benders decomposition. Alle sammen værktøjer, som vi vil få brug for i dette og det kommende afsnit om sports scheduling og schedulering af SAS-ligaen.

Inden for sport har man (nærmest siden sportens oprindelse) haft brug for at kunne lave en retfærdig måde at planlægge turneringer på. Når man tidligere skulle planlægge en større turnering, skete det typisk ved, at folk fra det respektive forbund satte sig sammen på et kontor og ved håndkraft fiflede sig frem til en brugbar plan for turneringen. Men nu, hvor turneringerne er vokset i størrelse og opmærksomhed, er kravene til spilleplanen blevet drastisk forøgede, blandt andet på grund af kommercialiseringen af sporten, og der er nu brug for hjælp fra computere og matematikere til at planlægge de større turneringer.

Dette gjorde, at *sports scheduling* for ca. 30 år siden opstod som sit eget felt inden for operationsanalyse, og i de senere år er der løbende blevet publiceret mere og mere indenfor feltet. Som eksempler på matematisk skemalagte ligaer kan nævnes de tyske, østrigske, hollandske og italienske fodboldrækker, en stor del amerikanske ligaer inden for et væld af sportsgrene og, sidst men ikke mindst, vores egen hjemlige SAS-liga. ([17], p1ff)

4.1 Turneringer generelt

Vi vil gennem resten af opgaven fokusere på en bestemt type turneringer: *round robin-turneringer*. En round robin-turnering (RRT) er det, man på godt dansk ville kalde en "alle-mod-alle-turnering", altså hvor hvert deltagende hold skal spille mod alle de andre hold. Antallet af gange, holdene skal møde hinanden, kan variere, og man betegner så turneringerne double RRT (DRRT), hvor alle hold skal møde hinanden to gange og så videre med triple og kvadriple RRT.

En RRT er generelt delt op i *runder* (eng.: slots), hvor holdene møder hinanden, eventuelt ikke samme dag eller sted. Hvis der er et lige antal deltagende hold, n, spiller alle hold i hver af de $n-1$ runder. Ved et ulige antal hold, vil der være et oversiddende hold i hver af de n runder. Man siger så, at holdet har en *bye* i den pågældende runde. Hvis der i en turnering spilles i flere runder end de minimum n hhv. $n-1$, kalder man turneringen for en *relaxed* turnering. Vi vil dog fremover fokusere på de *kompakte* turneringer, hvor ovenstående ikke er tilfældet.

Hvem holdene skal møde i de respektive runder kan opstilles i en såkaldt *timetable* (TT), hvor rækkerne svarer til holdene, og søjlerne svarer til de enkelte runder. ([17], p2f)

Eksempel 4.1. *Herunder ses et eksempel på timetables for turneringer med hhv. 4 og 5 deltagende hold (B svarer til, at det pågældende hold har en bye):*

Hold \ Runde	*1*	*2*	*3*
Hold 1	*2*	*3*	*4*
Hold 2	*1*	*4*	*3*
Hold 3	*4*	*1*	*2*
Hold 4	*3*	*2*	*1*

(a)

Hold \ Runde	*1*	*2*	*3*	*4*	*5*
Hold 1	*2*	*5*	*3*	*4*	*B*
Hold 2	*1*	*3*	*B*	*5*	*4*
Hold 3	*4*	*2*	*1*	*B*	*5*
Hold 4	*3*	*B*	*5*	*1*	*2*
Hold 5	*B*	*1*	*4*	*2*	*3*

(b)

I de fleste turneringer spilles der på et af holdenes hjemmebane. Dette hold har så en *hjemmekamp* (H), og modstanderholdet har en *udekamp* (A). I en DRRT spiller hvert hold en gang hjemme og en gang ude mod hvert andet hold. Ligesom man kan konstruere en timetable, kan man også konstruere et såkaldt *home away pattern* (hjemme-/udebanemønster) for hvert hold, der angiver, om holdet spiller ude eller hjemme i de forskellige runder. Mængden af home away patterns for alle hold i en turnering kaldes så et *home away pattern set* (HAP) eller bare pattern set. ([17], p3)

Eksempel 4.2. *Vi konstruerer nu mulige HAP for de to TT fra Eksempel 4.1:*

Hold \ Runde	*1*	*2*	*3*
Hold 1	*H*	*A*	*A*
Hold 2	*A*	*H*	*H*
Hold 3	*H*	*H*	*A*
Hold 4	*A*	*A*	*H*

(a)

Hold \ Runde	*1*	*2*	*3*	*4*	*5*
Hold 1	*H*	*A*	*H*	*A*	*B*
Hold 2	*A*	*H*	*B*	*A*	*H*
Hold 3	*H*	*A*	*A*	*B*	*H*
Hold 4	*A*	*B*	*H*	*H*	*A*
Hold 5	*B*	*H*	*A*	*H*	*A*

(b)

Man erstatter ofte H og A med 0 og 1 for bedre at kunne regne på udtrykket matematisk. Når man skal lave HAP for en turnering, vil man selvfølgelig gerne have, at hvert hold så vidt muligt får lige mange hjemme- og udekampe (af hensyn til fans, billetindtægter og lignende). I en single RRT (SRRT) med et lige antal hold vil holdene ikke få lige mange ude- og hjemmebanekampe. Et HAP kan godt være infeasible, f.eks. hvis for mange kampe er tvunget til at blive spillet i for få runder:

Eksempel 4.3. *Nedenstående HAP er infeasible, da hold 2,3 og 4 (eller 1,5 og 6) ikke kan møde hinanden i de sidste 3 runder, og er derfor nødt til at spille deres 3 indbyrdes kampe i runde 1 og 2, hvilket selvfølgelig er umuligt.*

Hold \ Runde	*1*	*2*	*3*	*4*	*5*
Hold 1	*H*	*A*	*A*	*H*	*A*
Hold 2	*A*	*H*	*H*	*A*	*H*
Hold 3	*H*	*A*	*H*	*A*	*H*
Hold 4	*A*	*H*	*H*	*A*	*H*
Hold 5	*H*	*A*	*A*	*H*	*A*
Hold 6	*A*	*H*	*A*	*H*	*A*

Generelt gælder der følgende to nødvendige betingelser for, at et HAP er feasible: ([9], p81)

1. I hver runde skal der spilles lige mange hjemme- og udebanekampe.

2. Alle holdenes hjemme-/udebanemønstre er forskellige.

Disse er selvfølgelig ikke tilstrækkelige, da f.eks. Eksempel 4.3 opfylder betingelserne men er infeasible. To mønstre siges at være *komplementære*, hvis der i hver runde r gælder, at begge mønstre har forskellig type kamp. Et eksempel på dette kan ses i Eksempel 4.2 (a), hvor hold 1 og 2 (3 og 4) har komplementære mønstre.

En anden væsentlig detalje er antallet af såkaldte *breaks*, der opstår, når et hold har to ude- eller hjemmekampe i træk. Eksempelvis har hold 3 i Eksempel 4.2 (b) et break i runde 2-3. For at gøre turneringen mest retfærdig prøver man at tilstræbe, at alle hold har det samme antal breaks. Minimering af breaks er en af de største udfordringer i sports scheduling og er det emne, der vil blive lagt mest vægt på fremover i opgaven.[4]

En *schedule* (spilleplan) for en turnering fremkommer nu ved at kombinere dens TT med dens HAP. I en DRRT kan man med fordel "spejle"en SRRT, så alle hold får lige mange ude- og hjemmebanekampe. En sådan turnering kaldes en *mirrored DRRT*, og dens spilleplan fremkommer ved at gentage timetablen for den enkelte SRRT og i sidste halvdel derefter at ombytte HAP, dvs. A bliver til H og omvendt. ([17], p3f)

Eksempel 4.4. *Vi laver nu spilleplanen for en mirrored DRRT (med TT og HAP fra Eksempel 4.1 og 4.2). I skemaet betyder + hjemmebanekamp, - udebanekamp og understregning break:*

Hold \ Runde	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Hold 1	*+2*	*-5*	*+3*	*-4*	*B*	*-2*	*+5*	*-3*	*+4*	*B*
Hold 2	*-1*	*+3*	*B*	*-5*	*+4*	*+1*	*-3*	*B*	*+5*	*-4*
Hold 3	*+4*	*-2*	*-1*	*B*	*+5*	*-4*	*+2*	*+1*	*B*	*-5*
Hold 4	*-3*	*B*	*+5*	*+1*	*-2*	*+3*	*B*	*-5*	*-1*	*+2*
Hold 5	*B*	*+1*	*-4*	*+2*	*-3*	*B*	*-1*	*+4*	*-2*	*+3*

Når man schedulerer i praksis, vil det ofte være en fordel at vente med at fastsætte holdene før til sidst; man anvender derfor begrebet *pladsholdere* til at repræsentere holdene undervejs i processen.

4.2 Teoretisk minimering af breaks

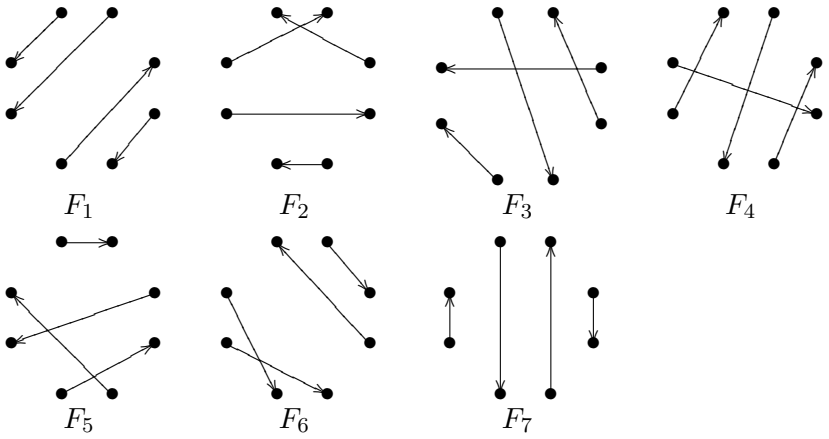
Vi tager nu fat på et af de mere interessante aspekter af sports scheduling, nemlig minimering af breaks. Dette er en væsentlig faktor i konstruktionen af enhver spilleplan, da klubberne f.eks. vil tilfredsstille deres fans bedre med løbende hjemmebanekampe gennem sæsonen og få en stabil indtjening til klubben på billetindtægter. Derudover er der det rent sportslige aspekt i, at hold generelt er stærkere hjemme end ude. Alt dette fører til, at en turnering med mindst muligt antal breaks vil være at foretrække.

I dette afsnit vil vi se på de mere teoretiske resultater for antallet af breaks og en konstruktionsmetode til at lave en turnering med et minimalt antal breaks.

[4] Et andet meget undersøgt emne er det såkaldte *Travelling Tournament Problem* (TTP), hvor man (primært) i amerikanske ligaer har store rejseomkostninger, og det derfor ofte kan være en fordel med to ude- eller hjemmebanekampe i træk. Her er det primære ikke længere at minimere breaks, men faktisk at tilstræbe enkelte breaks (se f.eks. i [1]). Sådanne overvejelser er ikke relevante for f.eks. SAS-ligaen, da holdene rejser hjem efter hver kamp.

Vi ser nu på en kompakt SRRT med et lige antal hold, n (resultaterne kan nemt udvides til et ulige antal hold ved at tilføje et "dummyhold"). Vi minder om, at en *komplet graf*, K_n, er en graf, hvor alle knuder er indbyrdes forbundet med en kant. Vi lader nu knuderne være holdene i turneringen og kanterne være kampene mellem de forbundne knuder (hold). Her udelukker vi altså kanter der går fra en knude til sig selv. En *1-faktor* F_i af en komplet graf $K_n = (V, E)$ er en delgraf af K_n, hvor alle knuder har valens 1. Hvis foreningen af $n-1$ indbyrdes disjunkte 1-faktorer $F = (F_1, \ldots, F_{n-1})$ giver K_n, kaldes F så en *1-faktorisering* af K_n. En sådan 1-faktorisering svarer til en opdeling af kampene i de $n-1$ runder. Orienterer vi nu kanterne i hver 1-faktor, svarer det til, at vi vælger et HAP for turneringen. Vi taler da om en *orienteret 1-faktorisering* $\vec{F} = (\vec{F}_1, \ldots, \vec{F}_{n-1})$ (svarende til en $n-1$-farvning af K_n), der så vil repræsentere spilleplanen for en SRRT. ([17], p6)

Eksempel 4.5. *Nedenfor ses en orienteret 1-faktorisering af K_8 og den dertil hørende spilleplan for en SRRT med $n = 8$. Knuderne navngives startende ved knude 1 som værende den venstre i øverste række og derefter nummereret efter urets retning (negativ omløbsretning):*



Hold \ Runde	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Hold 1	*-8*	*+3*	*-5*	*+7*	*-2*	*+4*	*-6*
Hold 2	*-7*	*+8*	*+4*	*-6*	*+1*	*-3*	*+5*
Hold 3	*+6*	*-1*	*-8*	*+5*	*-7*	*+2*	*-4*
Hold 4	*-5*	*+7*	*-2*	*+8*	*+6*	*-1*	*+3*
Hold 5	*+4*	*-6*	*+1*	*-3*	*-8*	*+7*	*-2*
Hold 6	*-3*	*+5*	*-7*	*+2*	*-4*	*+8*	*+1*
Hold 7	*+2*	*-4*	*+6*	*-1*	*+3*	*-5*	*-8*
Hold 8	*+1*	*-2*	*+3*	*-4*	*+5*	*-6*	*+7*

Det bemærkes, at der er 6 breaks i spilleplanen, hvor alle hold pånær 1 og 8 har netop ét break (understreget).

Der gælder følgende interessante resultat om antallet af breaks: ([22], p382f)

Proposition 4.6. *I en SRRT med n (lige) antal hold er der mindst $n-2$ breaks.*

Bevis. Antag, at der er færre end $n-2$ breaks i en spilleplan, dvs. der er mindst 3 hold, der ikke har breaks. Der findes generelt kun to slags mønstre uden breaks: $H, A, H, \ldots, H$ og $A, H, A, \ldots, A$, altså må mindst to af disse hold have hjemme-/udebanemønstre, der er ens. Disse to hold kan ikke møde hinanden i turneringen, derfor er der ingen kant mellem dem i K_n, der derfor ikke kan være komplet, og den ønskede modstrid er opnået. □

Til at konstruere spilleplaner, kan man bruge følgende definition: ([17], p7)

Definition 4.7. *Den kanoniske 1-faktorisering opfylder følgende (for n lige):*
For $i = 1, \ldots, n-1$ *gælder, at*

$$F_i = \{(n, i)\} \cup \left\{(i+k, i-k) | k = 1, \ldots, \frac{n}{2} - 1\right\},$$

hvor $i+k$ *og* $i-k$ *er tallene fra* $1, \ldots, n-1 \pmod{n-1}$.

Spilleplanen fra Eksempel 4.5 er konstruereret under brug af den kanoniske 1-faktorisering (hvis man ser bort fra orienteringen).

Vi ønsker at vise, at man kan lave en spilleplan med $n-2$ breaks ud fra den kanoniske 1-faktorisering. Til hvert hold p i turneringen, der svarer til vores kanoniske 1-faktorisering, tildeler vi en kampplan $K(p)$, altså rækkefølgen af de kampe, holdet skal spille. For eksempel er hold 3's kampplan i Eksempel 4.5: $K(3) = ((3,6),(1,3),(8,3),(5,3),(7,3),(2,3),(3,4))$ (hvis man i eksemplet ser bort fra orienteringen).
Generelt er $K(n) = ((n,1),(n,2),\ldots,(n,n-1))$.

For $p = 1$ har vi, at kampplanen starter med $(n,1)$ (som vi kalder Del I), derefter kommer der $\frac{n}{2} - 1$ kampe, hvor $i - k = 1$, $i = 2, 3, \ldots, \frac{n}{2}$ og $k = 1, 2, \ldots, \frac{n}{2} - 1$ (Del II). Del III består af $\frac{n}{2} - 1$ kanter, hvor $i + k = 1 \pmod{n-1}$ med $i = \frac{n}{2} + 1, \frac{n}{2} + 2, \ldots, n-1$ og $k = \frac{n}{2} - 1, \frac{n}{2} - 2, \ldots, 2$.

For $2 \leq p \leq n-1$ er vi nødt til at dele op i to tilfælde:

$2 \leq p \leq \frac{n}{2}$: I denne kampplan består Del I af $p-1$ kanter, hvor $i + k = p$, Del II består af (n,p), Del III består af $\frac{n}{2} - 1$ kanter, hvor $i - k = p$, og til sidst Del IV, der består af $\frac{n}{2} - p$ kanter, hvor $i + k = p \pmod{n-1}$. Det bemærkes, at der ikke er nogen Del IV for $p = \frac{n}{2}$, da $\frac{n}{2} - p = 0$.

$\frac{n}{2} + 1 \leq p \leq n-1$: I denne kampplan består Del I af $p - \frac{n}{2}$ kanter med $i - k = p \pmod{n-1}$, Del II består af $\frac{n}{2} - 1$ kanter med $i + k = p$, Del III består af (n,p), og Del IV består endelig af $n - 1 - p$ kanter, hvor $i - k = p$. Vi bemærker igen, at for $p = n-1$ indeholder Del IV ingen kanter. ([22], p383f)

Disse overgange mellem de forskellige dele af kampplanerne kan overskueliggøres i følgende skema:

$$K(1) = \left[\begin{array}{c|c|c} (n,1) & (i+k, i-k=1) & (i+k=1, i-k) \\ & i=2,\ldots,\frac{n}{2} & i=\frac{n}{2}+1,\ldots,n+1 \end{array}\right]$$

For $2 \leq p \leq \frac{n}{2}$ får vi følgende kampplan:

$$K(p) = \left[\begin{array}{c|c|c|c} (i+k=p, i-k) & (n,p) & (i+k, i-k=p) & (i+k=p, i-k) \\ i=1,\ldots,p-1 & & i=p+1,\ldots,\frac{n}{2}+p-1 & i=\frac{n}{2}+p,\ldots,n-1 \end{array}\right]$$

For $\frac{n}{2}+1 \leq p \leq n-1$ får vi følgende kampplan:

$$K(p) = \left[\begin{array}{c|c|c|c} (i+k, i-k=p) & (i+k=p, i-k) & (n,p) & (i+k, i-k=p) \\ i=1,\ldots,p-\frac{n}{2} & i=p-\frac{n}{2}+1,\ldots,p-1 & & i=p+1,\ldots,n-1 \end{array}\right]$$

Skema 1

Sætning 4.8. *For en turnering med n (lige) hold findes der et HAP med netop $n-2$ breaks.*

Bevis. Vi lader en orienteret kanonisk 1-faktorisering svare til vores turnering. Orienteringen foregår som følger:

(a) Kanten (n,i) orienteres fra i til n, hvis i er ulige og fra n til i, hvis i er lige.

(b) Kanten $(i+k, i-k)$ orienteres fra $i+k$ til $i-k$, hvis k er ulige og fra $i-k$ til $i+k$, hvis k er lige.

Vi bemærker først, at hold n jfr. (a) ingen breaks har. Desuden bemærker vi ved at se på $K(p)$ og (b), at der ikke kan forekomme nogen breaks *indenfor* de enkelte dele. Dette følger let ved at se, at når i vokser i udtrykket $i-k=p$, er k nødt til at falde, altså skifter k mellem at være lige og ulige. Tilsvarende for $i+k=p$. Altså kan der kun opstå breaks ved en overgang fra en del til en anden.

Vi ser nu på overgangene imellem de forskellige dele for ethvert p, $1 \leq p \leq n-1$. Der er 3 typer af dele (hvis vi ser bort fra orienteringen): $(i+k, i-k=p)$, $(i+k=p, i-k)$ og (n,p). Vi ser først på overgange mellem de to første. Da i vokser, er det sidste (største) k, for hvilket $i-k=p$, $k=\frac{n}{2}-1$. Altså er vores sidste i givet ved $i=\frac{n}{2}+p-1$, og dermed er vores første i, hvor $i+k=p \pmod{n-1}$, givet ved $i=\frac{n}{2}+p$. Da $\frac{n}{2}+p+k=p \pmod{n-1}$, får vi, at $k=\frac{n}{2}-1$. Da vi har samme k, medfører (b), at de vil have modsat orientering. Specielt kan hold 1 ikke have noget break mellem Del II og Del III. Men da den sidste (og eneste) kant i Del I for hold 1 er kanten $(1,n)$ (jfr. (a)), mens den første kant i Del II er kanten $(3,1)$ (jfr. (b)), kan der heller ikke opstå et break her. Altså har hold 1 og hold n ingen breaks.

Vi mangler nu at se på overgangene fra type $(i+k=p, i-k)$ til (n,p) og type (n,p) til $(i+k, i-k=p)$. Typen $(i+k=p, i-k)$ slutter med kanten $(p, p-2)$, da i vokser, er det sidste (mindste) k, for hvilket $i+k=p$, $k=1$. Tilsvarende ses det, at $k=1$ for typen $(i+k, i-k=p)$, der dermed starter med kanten $(p+2, p)$. Hvis p er ulige, er typen (n,p) orienteret som (p,n), og modsat hvis p er lige. Dermed vil der være et break *enten* mellem $(p, p-2)$ og (p,n) *eller* mellem (n,p) og $(p+2, p)$. Altså har hvert af de $n-2$ hold, hvor $2 \leq p \leq n-1$, netop ét break, og turneringen har samlet $n-2$ breaks. □

Grunden til, at vi fik netop 6 breaks i vores spilleplan fra Eksempel 4.5 var, at vi havde anvendt den kanoniske 1-faktorisering med orientering til at lave den pågældende spilleplan.

Bemærkning 4.9. *Vi bemærker, at der i en turnering med $n-2$ breaks (n lige), i hver runde vil være enten ingen eller netop 2 breaks. Hvis det antages, at der kun er ét break i en bestemt runde, vil der være et forskelligt antal H'er og A'er i den pågældende runde. Altså er et andet hold også nødt til at have et break, da turneringen ellers ville være infeasible. Grunden til, at der ikke kan være over 2 breaks i en runde, er, at to af holdene med breaks ikke vil have mulighed for at møde hinanden i turneringen, da hvert hold højst har ét break.*

For turneringer med et ulige antal hold, følger det næsten direkte af ovenstående sætning, at vi ikke får nogen breaks:

Korollar 4.10. *For en turnering med n (ulige) hold findes der et HAP uden breaks.*

Bevis. Vi starter med en turnering med $n+1$ hold og lader hold $n+1$ være et dummyhold. De steder, hvor hold i møder dette dummyhold, sætter vi en bye. Da holdenes breaks opstod netop ved overgangen til vores dummyhold (jfr. beviset for Sætning 4.8), hvor der nu er en bye, vil der ikke længere være nogen breaks. □

Lignende resultater kan også opstilles for mirrored DRRT. ([22], p387f) Vi indfører følgende notation: lad som sædvanlig K_n være den komplette orienterede graf med n (lige) knuder og lad nu G_n være K_n men med orienteringer begge veje på alle kanter, og så $\mathcal{H}_1 = (\vec{H}_1, \ldots, \vec{H}_{n-1})$ og $\mathcal{H}_2 = (\vec{H}_n, \ldots, \vec{H}_{2n-2})$ er orienterede 1-faktoriseringer af K_n. Dette svarer til, at $\mathcal{H}_1$ er første halvdel af spilleplanen, og $\mathcal{H}_2$ er anden halvdel.

Proposition 4.11. *En mirrored DRRT med n (lige) antal hold har mindst $3n-6$ breaks.*

Bevis. Da $\mathcal{H}_1$ og $\mathcal{H}_2$ er orienterede 1-faktoriseringer af K_n, vil de begge have mindst $n-2$ breaks. Hvert hold i $\mathcal{H}_1$ og $\mathcal{H}_2$, pånær to, vil have mindst ét break. Da der er et ulige antal runder, vil holdene med ét break få yderligere et break ved overgangen fra $\mathcal{H}_1$ til $\mathcal{H}_2$, altså ekstra $n-2$ breaks. I alt får vores mirrored DRRT dermed $3n-6$ breaks. □

Det er dog muligt at lave en ikke-mirrored DRRT med færre end $3n-6$ breaks, f.eks. ved at "spejle" *både* timetable *og* HAP fra den første halvdel. Dette vil give $2n-4$ breaks, dog vil holdene skulle møde hinanden to gange i træk ved overgangen fra første til anden halvdel.

Problemet med Proposition 4.11 er, at der er mulighed for at ramme to på hinanden følgende breaks (hvilket sker, når der ligger et break lige op til overgangen mellem de to halvdele), hvilket ikke er ønskværdigt. Følgende sætning tager sig af dette tilfælde: ([22], p388ff)

Sætning 4.12. *Der findes en mirrored DRRT med n (lige) antal hold og netop $3n-6$ breaks, og hvis $n \neq 4$, er der ikke to på hinanden følgende breaks.*

For $n = 2$ er der selvfølgelig ingen breaks. For $n = 4$ vil der være mindst 2 breaks. Hvis et hold har 2 breaks, må disse nødvendigvis være sammenhængende. Antager vi derfor, at hvert hold har højst 1 break, vil dette ligge i runde 2 eller 3. Ligger det i runde 2, vil der komme tilsvarende breaks i runde 4 og 5 (som dermed er sammenhængende), og ligger det i runde 3, vil der komme tilsvarende breaks i runde 4 og 6 (og dermed er der breaks i både runde 3 og 4). Sætningen gælder altså ikke for $n = 4$.

Bevis. Vi vil konstruere en metode til at lave en spilleplan med $3n - 6$ breaks og uden sammenhængende breaks ved at omforme den orienterede kanoniske 1-faktorisering. Vi genbruger (b) fra Sætning 4.8 og laver (a) om til (a'):

(a') Kanten (n, i) orienteres fra i til n, hvis $i \leq n - 5$ er ulige, eller hvis $i = n - 2$ og fra n til i ellers.

Ved denne ændring af orienteringsprocessen bemærker vi, at orienteringen for kanterne $a_{n-3,0}, a_{n-2,0}$ og $a_{n-1,0}$ (de sidste 3 runder for hold n) vendes om. De første $n - 4$ hold forbliver altså uændrede, og deres breaks vil stadig forekomme i runderne $3 \leq i \leq n - 3$. Venstresiden ses ved, at kun hold 1 og 2 har en typeovergang i runde 2. Hold 1 har ingen breaks overhovedet, og hold 2 har en typeovergang fra $(2, n - 1)$ til $(n, 2)$ i runde 2, altså ingen breaks i runde 2. Højresiden følger af, at der i de sidste 2 runder kun er mulighed for breaks ved hold n, $n - 1$, $n - 2$ og $n - 3$.

Hold n har nu et break i runde $n - 3$ (hvor første orienteringsvending foregår). Vi konstruerer nu et skema (Skema 2), der illustrerer ændringerne efter orienteringsvendingerne. (α) er lavet ud fra den kanoniske 1-faktorisering samt med (a) og (b) fra Sætning 4.8, og (β) er lavet ud fra (a') og (b):

$\vdots$	$(n, n-4)$	$(n-3, n-5)$
F_{n-3}	$(n-3, n)$	$(n-2, n-4)$
F_{n-2}	$(n, n-2)$	$(n-1, n-3)$
F_{n-1}	$(n-1, n)$	$(1, n-2)$

(α)

$\vdots$	$(n, n-4)$	$(n-3, n-5)$
F'_{n-3}	$(n, n-3)$	$(n-2, n-4)$
F'_{n-2}	$(n-2, n)$	$(n-1, n-3)$
F'_{n-1}	$(n, n-1)$	$(1, n-2)$

(β)

Skema 2

Vi ser i skemaet, at hold $n - 1$ ikke længere har noget break. Vi ser også, at hold $n - 3$'s break har flyttet sig fra runde $n - 3$ til runde $n - 2$, ligesom hold $n - 2$ har fået flytte sit break fra runde $n - 1$ til $n - 2$. Antallet af breaks i $\mathcal{H}_1$ er derfor stadig $n - 2$. Da der nu ikke længere er nogen breaks i runde $n - 1$, er det ønskede vist, og vi har, at vores mirrored DRRT har $3n - 6$ breaks, hvoraf ingen er sammenhængende. □

Eksempel 4.13. *Vi illustrerer Sætning 4.12 ved at anvende (a') på Eksempel 4.5 (ændringer med fed skrift). Det ses af skemaet, at der ingen breaks er i runde 7, derfor vil der ikke komme nogen sammenhængende breaks, hvis turneringen spejles til en mirrored*

DRRT:

Hold \ Runde	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Hold 1	*-8*	*+3*	*-5*	*+7*	*-2*	*+4*	*-6*
Hold 2	*-7*	*+8*	*+4*	*-6*	*+1*	*-3*	*+5*
Hold 3	*+6*	*-1*	*-8*	*+5*	*-7*	*+2*	*-4*
Hold 4	*-5*	*+7*	*-2*	***-8***	*+6*	*-1*	*+3*
Hold 5	*+4*	*-6*	*+1*	*-3*	***+8***	*+7*	*-2*
Hold 6	*-3*	*+5*	*-7*	*+2*	*-4*	***-8***	*+1*
Hold 7	*+2*	*-4*	*+6*	*-1*	*+3*	*-5*	***+8***
Hold 8	*+1*	*-2*	*+3*	***+4***	***-5***	***+6***	***-7***

4.3 Constrained Minimum Break-problemet

Fremover i opgaven vil vi kun se på turneringer med et lige antal hold.

Resultaterne fra foregående afsnit er knap så anvendelige i praksis, da der i de fleste tilfælde vil være et hav af ekstra betingelser (constraints), der skal være opfyldt for forskellige turneringer. Det er ligeledes svært at opstille generelle resultater, da disse betingelser varierer fra turnering til turnering. I dette afsnit vil vi kort se på de forskellige constraints og kompleksiteten af at schedulere turneringer.

De forskellige constraints kan groft set opdeles i følgende grupper: ([17], p4f)

Placeringsbetingelser: Constraints, der sikrer, at holdet spille ude eller hjemme i en bestemt runde. Dette kan typisk opstå, hvis en bestemt "bane" ikke er tilgængelig i en bestemt runde.

Top- og bundholdsbetingelser: Constraints, der tager hensyn til stærke hhv. svage hold i turneringen. Eksempelvis at alle hold gerne vil have en hjemmekamp mod et af topholdene i hver halvdel af turneringen.

Break constraints: Constraints, der sikrer, at der ikke er breaks i bestemte runder, typisk i runde 2 og i sidste runde.

Kampbegrænsninger: Constraints, der sikrer, at bestemte kampe kommer til at ligge i bestemte runder. F.eks. vil fjernsynskanaler ofte gerne sikre sig, at to tophold mødes på et bestemt tidspunkt.

Fællesbanebetingelser: Constraints, der sikrer, at hvis to hold har samme hjemmebane, spiller de ikke begge på hjemmebane i samme runde. Sådanne constraints er knap så væsentlige for holdene i den nuværende SAS-liga, da alle hold har hver deres hjemmebane.

Geografiske betingelser: Her sikres det, at ikke alt for mange kampe foregår inden for kort tid i samme geografiske region. Eksempelvis lå Århus Fremad og AGF begge i Superligaen for noget tid siden (i dag ligger ingen af dem der...).

Pattern-betingelser: Constraints, der sikrer, at nogle bestemte egenskaber ved HAP er opfyldt, f.eks. at der ikke er to på hinanden følgende breaks.

Separationsbetingelser: Her angives det f.eks., hvor mange runder der skal gå, før to hold må møde hinanden igen. Eksempelvis i en mirrored turnering vil der være $n-2$ runder, før holdene mødes igen.

Holdbetingelser: Constraints, der f.eks. sikrer, at nogle bestemte hold har flere hjemmebanekampe end andre grundet deres placering i de foregående sæsoner.

Disse constraints prioriteres normalt efter, hvor vigtige de er for en given turnering. Der skelnes typisk mellem "bløde" og "hårde" constraints, hvor hårde constraints *skal* være opfyldt og så mange som muligt af de bløde skal være opfyldt. Hvis ikke de hårde constraints er opfyldt, anses HAP for værende infeasible.

Et sports scheduling-problem kan generelt deles op i trin, hvor rækkefølgen, de skal udføres i, varierer efter problemet: ([17], p9f)

Trin 1: Generér patterns.
Trin 2: Find et HAP for pladsholdere.
Trin 3: Find en TT for pladsholdere.
Trin 4: Tildel hold til pladsholdere.

Op gennem tiden har der været forskellige tiltag til at løse ovenstående trin. Schreuder ([19]) var en af de første, der løste praktiske problemer mht. sports scheduling ved hjælp af teorien. Han skemalagde 1989/1990-sæsonen af den hollandske fodboldrække som en mirrored DRRT i 2 faser: først genererede han patterns og lavede en spilleplan ved hjælp af den kanoniske 1-faktorisering (trin 1, 2 og 3). I fase 2 tildelte han hold til pladsholderne (trin 4).

For 1997/1998-sæsonen skemalagde Nemhauser og Trick ([12]) en amerikansk basketballturnering for Atlantic Coast Conference med 9 hold ved at følge trinene slavisk og anvende heltalsprogrammering som hjælp til trin 2 og 3. Efter denne artikel begyndte flere at se på mulighederne for at bruge constraint programmering i stedet for heltalsprogrammering, heriblandt Régin ([18]). Generelt diskuteres det, hvorvidt man med fordel kan udføre trin 3 og 4 før 1 og 2 eller omvendt. Régin er på den første vogn. Han ser på et generelt problem, der i dag går under navnet *Break Minimization-problemet*:

Definition 4.14. *Givet en TT, da består Break Minimization-problemet i at finde et pattern set, som minimerer antallet af breaks.*

Diskussionen om, hvilke trin, der skal udføres først, fortsætter, da Trick argumenterer for, at rækkefølgen afhænger af, hvilke constraints der er de væsentligste. Trick ([21]) kombinerer derefter IP og CP til løsning af de forskellige trin.

Elf m.fl. ([2]) løser Break Minimization-problemet ved at omforme det til et grafteoretisk problem. I stedet for at minimere antallet af breaks direkte kan man i stedet maksimere størrelsen af et såkaldt *cut* i en ikke-orienteret graf, der svarer til vores TT ved at lave snedige vægtninger på kanterne. Til sidst i deres artikel opstilles en række formodninger angående kompleksiteten af at finde brugbare HAP for givne TT, der senere løses af Miyashiro/Matsui og Post/Woeginger. Vi gengiver her nogle af resultaterne: ([11], p5ff)

Sætning 4.15. *Følgende problem P kan løses i polynomiel tid:*

For en givet TT med n hold, find et tilhørende HAP med højst $n-2$ breaks og returner infesible, hvis et sådant HAP ikke eksisterer.

Vi ønsker at bevise sætningen ved at reducere problemet til n instanser af det såkaldte 2SAT-problem:

Definition 4.16. *k-SAT-problemet*
Givet n 0-1-variable (Boolske variable) $x_1, x_2, \ldots, x_n$ og m klausuler $C_1, C_2, \ldots, C_m$, alle bestående af op til k enten Boolske variable x_i eller deres negation $\neg x_i$.

Find en 0-1-tildeling af variablene $x_1, x_2, \ldots, x_n$, så alle klausuler er opfyldt, hvis en sådan findes; ellers angiv at ingen findes.

Det interessante ved 2SAT-problemet er, at det er løseligt inden for polynomiel tid. Kan vi reducere problemet i sætningen til et 2SAT-problem, må dette også være løseligt inden for polynomiel tid, jfr. Proposition 1.4, *i*).

Bevis. Lad $N = \{1, 2, \ldots, n\}$ være mængden af deltagende hold og $R = \{1, 2, \ldots, n-1\}$ være mængden af runder. Se på problemet P_k:
Givet et TT for en turnering med n hold, find da (om muligt) et U, der tilfredsstiller følgende betingelser:

1. U er et HAP for TT
2. Spilleplanen har $n-2$ breaks
3. Det k'te holds spilleplan er $(H, A, H, A, \ldots, H)$

Ifølge Sætning 4.8 er problemet P feasible, hvis og kun hvis et af problemerne P_k, $k = 1, \ldots, n$ er feasible. Vi indfører notationen r_{ij}, $i \neq j$ for den runde, hvor hold i skal møde hold j, dvs. $r_{ij} = r_{ji}$. Vi opretter nu et midlertidigt HAP, $U^k = (u^k_{i,r})$, hvor $u^k_{i,r}$ angiver, om hold i spiller ude eller hjemme i runde r.
Vi bemærker nu, at ifølge betingelse 3 fra P_k er hold k's hjemme-udebanemønster fastlagt. Dermed er hold $i \in N\backslash\{k\}$ også fastlagt i runderne r_{ki}.
Vi vælger U^k så den opfylder følgende betingelser:

1. Den k'te række (hold) i U^k er $(H, A, H, A, \ldots, H)$
2. Der gælder for alle $i \in N\backslash\{k\}$, at hvis r_{ki} er lige, opfylder den i'te række i U^k, at
$$\left(u^k_{i,r_{ki}}, u^k_{i,r_{ki}+1}, \ldots, u^k_{i,n-1}, u^k_{i,1}, \ldots, u^k_{i,r_{ki}-1},\right) = (H, A, H, A, \ldots, H).$$
3. Der gælder for alle $i \in N\backslash\{k\}$, at hvis r_{ki} er ulige, opfylder den i'te række i U^k, at
$$\left(u^k_{i,r_{ki}}, u^k_{i,r_{ki}+1}, \ldots, u^k_{i,n-1}, u^k_{i,1}, \ldots, u^k_{i,r_{ki}-1},\right) = (A, H, A, H \ldots, A).$$

Vi indfører de Boolske variable $x_{i,r},\ (i,r) \in N\backslash\{k\}\times R$ og antager nu, at P_k har en løsning U^*. U^* angiver vi som en sandt/falsk-tildeling (T/F) $x^*_{i,r} \in \{T,F\},\ (i,r) \in N\backslash\{k\} \times R$ på følgende måde:

$$x^*_{i,r} = \begin{cases} T,\ u^*_{i,r} = u^k_{i,r} \\ F,\ u^*_{i,r} \neq u^k_{i,r} \end{cases}$$

Med denne definition vil sandt/falsk-tildeling opfylde følgende:

1. For alle hold $i \in N\backslash\{k\}$ gælder (jfr. betingelse 3 i P_k og betingelse 1 i U^k), at $x^*_{i,r_{ki}}$ er sand.

2. For alle par $i,j \in N\backslash\{k\}$ med $i \neq j$ gælder, da U^* er en løsning til vores TT, at

$$\left[u^k_{i,r_{ij}} \neq u^k_{j,r_{ij}} \Rightarrow x^*_{i,r_{ij}} = x^*_{j,r_{ij}}\right] \quad \text{og} \quad \left[u^k_{i,r_{ij}} = u^k_{j,r_{ij}} \Rightarrow x^*_{i,r_{ij}} \neq x^*_{j,r_{ij}}\right]$$

   I det andet tilfælde bemærker vi, at det ikke er en mulig løsning, hvis to hold skal møde hinanden i samme runde, hvor de begge enten spiller hjemme eller ude.

3. Vi ved, at hvert hold $i \in N\backslash\{k\}$ har højst ét break i deres spilleplaner. Derfor vil variablene

$$\left(x^*_{i,r_{ki}}, x^*_{i,r_{ki}+1}, \ldots, x^*_{i,n-1}, x^*_{i,1}, \ldots, x^*_{i,r_{ki}-1}\right)$$

   antage én af følgende værdirækkefølger: $(T,T,\ldots,T,T)$, $(T,T,\ldots,T,F)$, $(T,T,\ldots,F,F)$, $(T,F,\ldots,F,F)$.

Vi bemærker, at betingelse 3 ovenfor er det samme som, at klausulerne

$$\left(x^*_{i,r_{ki}} \vee \neg x^*_{i,r_{ki}+1}\right), \left(x^*_{i,r_{ki}+1} \vee \neg x^*_{i,r_{ki}+2}\right), \ldots, \left(x^*_{i,n-1} \vee \neg x^*_{i,1}\right),$$
$$\left(x^*_{i,1} \vee \neg x^*_{i,2}\right), \ldots, \left(x^*_{i,r_{ki}-2} \vee \neg x^*_{i,r_{ki}-1}\right)$$

alle er sande.

Vi er nu i stand til at omformulere P_k til et 2SAT_k-problem:

Givet et TT for en turnering med n hold, find da (om muligt) et sandt/falsk-tildeling $x_{i,r} \in \{T,F\},\ (i,r) \in N\backslash\{k\} \times R$, der tilfredsstiller følgende betingelser:

(a) $x_{i,r_{ki}} = T$, $\quad i \in N\backslash\{k\}$

(b1) $x_{i,r_{ij}} = x_{j,r_{ij}}$, $\quad i,j \in N\backslash\{k\},\ u^k_{i,r_{ij}} \neq u^k_{j,r_{ij}}$

(b2) $x_{i,r_{ij}} \neq x_{j,r_{ij}}$, $\quad i,j \in N\backslash\{k\},\ u^k_{i,r_{ij}} = u^k_{j,r_{ij}}$

(c) $(x^*_{i,r_{ki}} \vee \neg x^*_{i,r_{ki}+1}) = (x^*_{i,r_{ki}+1} \vee \neg x^*_{i,r_{ki}+2})$
$= \ldots = (x^*_{i,n-1} \vee \neg x^*_{i,1}) = (x^*_{i,1} \vee \neg x^*_{i,2})$
$= \ldots = (x^*_{i,r_{ki}-2} \vee \neg x^*_{i,r_{ki}-1}) = T$ $\quad i \in N\backslash\{k\}$

For en sandt/falsk-tildeling $x'_{i,r}$, der opfylder ovenstående 4 betingelser, kan vi lave et HAP, $U' = (t'_{i,r})$, der opfylder, at det k'te hold i U' har kampplan $(H,A,H,A,\ldots,A)$ og for alle hold $i \in N\backslash\{k\}$ gælder, at

$$t'_{i,r} = \begin{cases} t^k_{i,r}, & x'_{i,r} = T \\ A, & x'_{i,r} = F \text{ og } t^k_{i,r} = H \\ H, & x'_{i,r} = F \text{ og } t^k_{i,r} = A \end{cases}$$

Betingelserne (a), (b1) og (b2) sikrer, at vores HAP, U', er en løsning til vores givne TT, og betingelse (c) sørger for, at hvert hold har højst ét break. Dette ses ved, at i klausulen (x, y) er x sand, når der ikke er breaks; ved ét break bliver y sand, og ved flere breaks findes der én klausul (x_i, y_i), hvor begge er falske.

Derfor bliver turneringens samlede antal breaks lig $n - 2$, da hold k ikke har nogen breaks, og det hold, der møder hold k i første runde har heller ingen breaks. Gennem beviset har vi ligeledes vist, at vi ud fra en løsning til P_k kan konstruere en tildeling, der opfylder alle betingelserne i 2SAT_k-problemet.

Da hver af 2SAT_k-problemerne er specialtilfælde af 2SAT-problemet (som er løsbart inden for polynomiel tid), er også problemet P løseligt inden for polynomiel tid, hvilket beviser påstanden i sætningen. □

Der er efterhånden også vist andre interessante resultater omkring kompleksiteten af sports scheduling. Bl.a. har Matsui/Miyashiro i [10] også vist resultater om kompleksitet ved nogle specialtilfælde, hvor bl.a. alle hold har højst ét break. Post/Woeginger ser på såkaldte *partielle TT* for SRRT, hvor man ser på et færre antal runder, hvor holdene møder hinanden højst én gang og forsøger at minimere breaks i sådanne TT: ([14], p172)

Sætning 4.17. *Minimering af breaks i partielle TT med n (lige) hold og 3 runder er $\mathcal{NP}$-hårdt.*

Korollar 4.18. *Minimering af breaks i partielle TT med n (lige) hold og et fastsat antal runder $r \geq 4$ er $\mathcal{NP}$-hårdt.*

4.4 HAP feasibility-problemet

I dette afsnit vil vi nu at se på det såkaldte *HAP feasibility problem*: ([9], p79)

Definition 4.19. *Givet et HAP, da består HAP feasibility-problemet i at afgøre, om det er muligt at finde en tilhørende TT.*

Vi vi nu vende tilbage til constraint programming og se mere specifikt på dens anvendelse inden for sports scheduling. Man siger, at et CSP (se afsnit 3.1) er *kantkonsistent* mht. en constraint c omhandlende variablene $x_1, \ldots, x_k$, $k \leq n$, hvis der for ethvert $i \in \{1, \ldots, k\}$ og hvert $v \in D_{x_i}$ findes et element $(v_{x_1}, \ldots, v_{x_{i-1}}, v, v_{x_{i+1}}, \ldots, v_{x_k}) \in c$. Et CSP generelt kaldes så kantkonsistent, hvis det er kantkonsistent mht. til alle dets constraints.[5] ([4], p4f)

I formuleringen af et sports scheduling-problem er de to constraints all-different og one-factor med til at sikre, at hhv. hvert hold møder alle andre hold i deres spilleplan netop én gang, og at holdene parres to og to i runderne, altså hvis hold i skal spille mod hold j, så skal hold j spille mod hold i. Man har fundet ud af, at all-different i sig selv bør medføre kantkonsistens. For at sikre disse betingelser man foreslået 3 propagationsteknikker, hvor o_{ir} angiver hold i's modstander i runde r: ([17], p13f)

1. kantkonsistent propagation mht. følgende constraints:

$$o_{ir} \neq i \quad i = 1, \ldots, n$$
$$o_{o_{ir}r} = i \quad i = 1, \ldots, n$$

[5] Kantkonsistent propagation laver så et CSP om til et kantkonsistent CSP.

2. kantkonsistent propagation mht. følgende constraints:

$$\begin{aligned} &o_{ir} \neq i \quad i = 1, \ldots, n, \ r = 1, \ldots, n-1 \\ &o_{o_{ir}r} = i \quad i = 1, \ldots, n, \ r = 1, \ldots, n-1 \\ &\text{all-different}(o_{1r}, \ldots, o_{nr}) \quad r = 1, \ldots, n-1 \end{aligned}$$

3. kantkonsistent propagation mht. følgende constraints:

$$\text{one-factor}(o_{1r}, \ldots, o_{nr}) \quad r = 1, \ldots, n-1.$$

I [4], p8ff vises det eksperimentelt, at den første metode ikke er synderligt effektiv, hvorimod tilføjelsen af all-different i toeren markant forbedrer kørselstiden. Hvis der ikke er givet et HAP, er treeren den mest effektive metode, hvorimod hvis der er givet et HAP (som f.eks. i tilfældet med den danske SAS-liga), er toeren mest effektiv. Dette har Trick vist i [20], hvor han betragter en SRRT, hvor D_i er mængden af mulige modstandere for hold i i en bestemt runde. D_i kaldes så *bipartit*, hvis mængden af alle hold i turneringen kan deles op i to lige store grupper X og Y med hver $\frac{n}{2}$ hold, så der gælder, at hvert hold i X kun skal spille mod hold fra Y og modsat. Hvis hjemme-udebanemønsteret for en runde i en turnering er givet på forhånd, opstår der naturligt en opdeling (bipartition) mellem hjemme- og udehold som ovenfor: ([20], p5)

Sætning 4.20. *Hvis D_i, $i = 1, \ldots, n$ er bipartit, medfører kantkonsistens for de tre constraints*

$$\begin{aligned} &o_{ir} \neq i \quad i = 1, \ldots, n \\ &o_{o_{ir}r} = i \quad i = 1, \ldots, n \\ &\text{all-different}(o_{1r}, \ldots, o_{nr}) \end{aligned}$$

kantkonsistens for

$$\text{one-factor}(o_{1r}, \ldots, o_{nr}).$$

Altså bør man bruge metode 2 i de tilfælde, hvor der er tale om et HAP feasibility-problem, da den er kantkonsistent og hurtigere. Miyashiro/Iwasaki/Matsui har i [9] set på, hvordan man sikrer, at ens HAP er feasible. Vi vender nu tilbage til Eksempel 4.3 og indfører lidt notation: lad som før $N = \{1, 2, \ldots, n\}$ være mængden af hold, $R = \{1, 2, \ldots, n-1\}$ mængden af runder, og lad $\tilde{N} \subseteq N$ være en delmængde af holdene, hvor $m \in \tilde{N}$ er de tilsvarende hjemme-/udebanemønstre. Lad desuden $\sum_{m \in \tilde{N}} h_{mr}$ og $\sum_{m \in \tilde{N}} (1 - h_{mr})$ være hhv. antal hjemme- og udekampe i runde r for mønstrene hørende til holdene i $\tilde{N}$. Vi indfører nu følgende størrelse: ([9], p82)

$$\alpha(\tilde{N}) = \left(\sum_{r \in R} \min \left\{ \sum_{m \in \tilde{N}} h_{mr}, \sum_{m \in \tilde{N}} (1 - h_{mr}) \right\} \right) - \frac{|\tilde{N}| \left(|\tilde{N}| - 1 \right)}{2}.$$

Proposition 4.21. *For ethvert feasible HAP gælder der for enhver delmængde $\tilde{N}$ af N, at $\alpha(\tilde{N}) \geq 0$.*

Bevis. Vi ser, at $\frac{|\tilde{N}|(|\tilde{N}|-1)}{2}$ er antallet af kampe, der skal spilles i alt af holdene i $\tilde{N}$, for at de alle kan møde hinanden. Antag, at der findes $\tilde{N} \subseteq N$, så $\alpha(\tilde{N}) < 0$. Så kan alle holdene i $\tilde{N}$ ikke møde hinanden, og vi har opnået en modstrid. □

I eksempel 4.3 vil der for $\tilde{N} = \{2, 3, 4\}$ gælde, at $\alpha(\tilde{N}) = 2 - 3 = -1$. Altså ser vi igen, at vores HAP ikke feasible.

Vi kan nu omformulere de to betingelser fra side 32 om nødvendige betingelser, for at et HAP er feasible, til:

1. $\alpha(\tilde{N}) = 0$
2. For alle delmængder $\tilde{N}$ af N med $|\tilde{N}| = 2$ gælder, at $\alpha(\tilde{N}) \geq 0$.

I [9] (p96f) vises det, at en turnering med et minimalt antal breaks, der opfylder Proposition 4.21, kan findes inden for polynomiel tid. Desuden undersøges det eksperimentelt, om implikationen også gælder den anden vej, altså om der for et HAP, der opfylder, at $\alpha(\tilde{N}) \geq 0$, også gælder, at dette HAP er feasible. I eksperimentet blev der fokuseret på HAP med et minimalt antal breaks. Hvorvidt det pågældende HAP var feasible, blev undersøgt vha. heltalsprogrammering. Konklusionen på eksperimentet blev, at dette var sandt for turneringer med op til 26 deltagende hold. Grunden til, at tallet ikke blev større, skyldtes manglende computerkraft:

Korollar 4.22. *For turneringer med $n \leq 26$ hold gælder der, at et HAP med et minimalt antal breaks er feasible, hvis og kun hvis der for alle $\tilde{N} \subseteq N$ gælder, at $\alpha(\tilde{N}) \geq 0$.*

5 SAS-ligaen – et konkret eksempel

Vi vil nu langt om længe se nærmere på en konkret schedulering af en turnering, nemlig den bedste danske fodboldrække SAS-ligaen, hvis 2006/07-sæson er blevet scheduleret af Rasmussen i [15]. Det lidt atypiske ved SAS-ligaen er, at der er tale om en *triple* RRT, altså hvor alle hold møder hinanden 3 gange. Der er 12 deltagende hold, hvilket giver os 33 runder med 6 kampe i hver. Vi vælger at opdele turneringen i to dele, $\mathcal{P} = \{1, 2\}$, hvor første del er de første 11 runder, og anden del er de resterende 22; dvs. vi opdeler i en SRRT efterfulgt af en DRRT. Tilsvarende kalder vi igen mængden af alle runder for R, og $R^1 = \{1, 2, \ldots, 11\}$ og $R^2 = \{12, 13, \ldots, 33\}$ lader vi være runderne i de to dele.

I SAS-ligaen findes der (ligesom i de fleste andre ligaer) et stort antal constraints, der skal eller helst skal være opfyldt i den endelige spilleplan. Disse constraints stiller krav til både HAP og TT for turneringen, og der forefindes constraints af alle typer, bortset fra, at ingen nuværende SAS-ligahold deler stadion. De forskellige constraints vægtes efter, hvor vigtige de er. Såkaldte hårde constraints *må* ikke brydes, og de bløde constraints *kan* brydes, men medfører en "straf-værdi, der ønskes minimeret i den endelige spilleplan. Vi opremser nu hurtigt de forskellige constraints for SAS-ligaen: ([15], p3f)

Placeringsbetingelser: Constraints, der sikrer, at et hold spiller ude eller hjemme i en bestemt runde. Dette kan enten være en blød eller hård constraint afhængig af grunden hertil.

Top- og bundholdsbetingelser: I SAS-ligaen regnes to hold for "tophold" (Brøndby og FCK). Da publikumsindtægterne fra et stadion er bedre, når et tophold kommer på besøg, tilstræbes det, at et af topholdene besøger hvert af de øvrige hold i del 1 (runde 1-11). Disse er bløde constraints. Mængden af ikke-tophold kalder vi N^{To}.

Breakbetingelser: Her sikres det for SAS-ligaens vedkommende, at der hverken er breaks i runde 2 eller i sidste runde (33). Førstnævnte er en blød constraint, sidstnævnte er hård. Desuden ønskes det så vidt muligt, at holdene alternerer mellem hjemme- og udebanekampe. Disse constraints er selvfølgelig bløde.

Kampbegrænsninger: I disse bløde constraints prøver man at sikre, at en kamp mellem to hold (i, j) skal spilles i en bestemt periode.

Geografiske betingelser: For to (eller flere) hold, der kommer fra samme geografiske område sikres det med disse bløde constraints, at der er hold, der spiller hhv. ude og hjemme i hver runde.

Pattern-betingelser: I SAS-ligaen tillades to eller flere på hinanden følgende breaks ikke (hårde constraints).

Separationsbetingelser: Her angives det f.eks., hvor mange runder ($k = 4$ for SAS-ligaens vedkommende) der skal gå, før to hold må møde hinanden igen. Dette er en hård constraint.

Holdbetingelser: De 6 bedste hold fra forrige sæson tildeles hver en ekstra hjemmebane-kamp i del 1, altså har disse hold i alt 17 hjemmebanekampe og 16 udebanekampe samlet set i turneringen. Dette er hårde constraints. Desuden tilstræbes det, at

hvis et hold i i de foregående sæsoner har spillet mange gange ude mod et andet hold j i del 1, lader man i den nye sæson hold i få hjemmebane mod hold j. Disse er bløde constraints.

Selve fremgangsmåden til at finde en brugbar spilleplan, der minimerer størrelsen af de brudte constraints, bygger på metoden fra [16], hvor der bruges logic-based Benders decompostition til at opdele i hoved- og underproblemer, ligesom der undervejs kan tilføjes specielle Benders cuts. Fremgangsmåden kan opdeles i fire trin, hvis detaljer vi vender tilbage til senere:

Trin 1 begynder med at generere hjemme-/udebanemønstre med et minimalt antal breaks, der sikrer, at antallet af hjemme- og udebanekampe for hvert hold højst er 6 i del 1, at der ikke er to på hinanden følgende breaks, og at der ikke nogen breaks i sidste runde. Først genereres mønstre med 0 breaks, derefter fortsættes med 1 break osv. Disse mønstre samles løbende i en mængde, og kan der ikke genereres flere, stopper algoritmen. Trin 1 udføres vha. constraint programming.

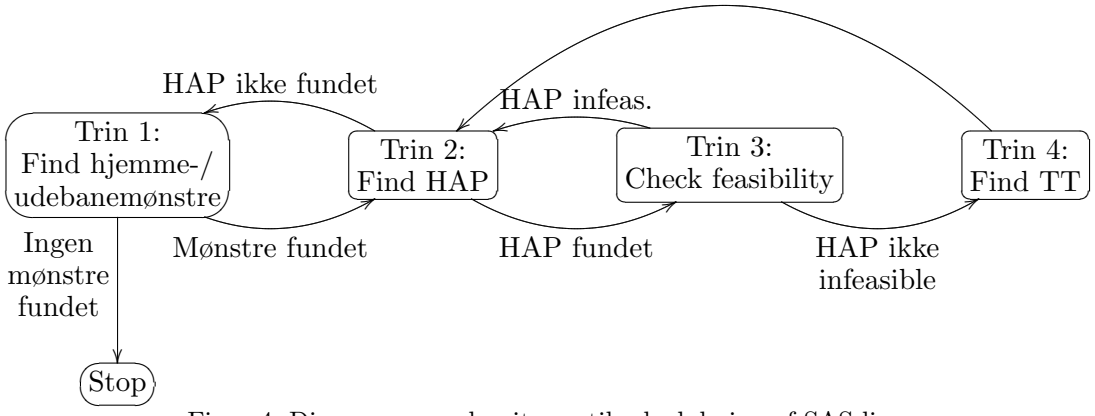
Trin 2 forsøger nu at generere HAP ud fra de fundne mønstre, samtidig med, at disse opfylder lidt flere af de givne constraints. Dette løses vha. heltalsprogrammering og svarer til vores hovedproblem mht. Benders decomposition. Her indføres et antal bløde constraints, bl.a. geografiske, placerings- og breakbetingelser. Trin 2 udføres vha. heltalsprogrammering.

Trin 3 undersøger, om de fundne HAP er infeasible ud fra kriterierne fra afsnit 4.4. Hvis HAP er infeasible, genereres et Benders cut, der tilføjes til hovedproblemet, og algoritmen vender tilbage til trin 2. Hvis det fundne HAP ikke er konstateret infeasible, fortsættes der til trin 4. I trin 3 anvendes både heltalsprogrammering *og* constraint programmering.

Trin 4 anvender heltalsprogrammering til at finde et TT, der passer til vores HAP genereret i Trin 2. Dette er vores underproblem mht. Benders decomposition-teknikken. Hvis et feasible TT findes, tilføjes et optimalitetscut til hovedproblemet, ellers tilføjes et regulært Benders cut til hovedproblemet. I begge tilfælde returneres der altså til trin 2.

Algoritmen fortsætter, indtil vores hovedproblem er infeasible, og der ikke kan dannes flere hjemme-/udebanemønstre. Vi har nu enten ingen løsning eller en optimal løsning til problemet.

Det centrale i algoritmen er styrken af de tilføjede Benders cuts. Cuts, der kun udtaler sig om et enkelt tilfælde, er ikke nær så stærke som cuts, der udtaler sig generelt om mange tilfælde. Hvis det f.eks. findes, at to bestemte hjemme-/udebanemønstre ikke kan optræde i samme HAP, kan et cut sortere et stort antal HAP væk fra løsningsmængden. Fremgangsmåden er illustreret i Figur 4. ([15], p5f)



Figur 4: Diagram over algoritmen til schedulering af SAS-ligaen

5.1 Generér mønstre

I Trin 1 genererer vi hjemme-udebanemønstre for del 1 og 2 hver for sig, idet dette reducerer antallet af mulige mønstre og giver større fleksibilitet mht. feasibility af de enkelte dele. I dette trin af processen tages der kun højde for hårde constraints (derfor er der heller ingen objektfunktion, der skal minimeres). Lad P^1 og P^2 være de genererede mønstre for de to dele, hvor vi lader B^1 og B^2 være øvre grænser for antallet af breaks. Hver gang vi anvender Trin 1, stiger disse to variable med 1, indtil de når den af DBU fastsatte grænse på 3 (altså kan hvert hold højst have 7 breaks i hele turneringen). Indførelsen af variablen h_r, der er 1, hvis mønsteret har en hjemmekamp i runde r og 0 ellers gør, at vi kan løse opgaven for del 1 med CP[6], hvortil vi anvender følgende formulering: ([15], p7)

$$\sum_{r=1}^{11} h_r \leq 6 \tag{10}$$

$$\sum_{r=1}^{11} h_r \geq 5 \tag{11}$$

$$\sum_{r=\hat{r}}^{\hat{r}+2} h_r \leq 2 \qquad \forall \hat{r} \in \{1,2,\ldots,9\} \tag{12}$$

$$\sum_{r=\hat{r}}^{\hat{r}+2} h_r \geq 1 \qquad \forall \hat{r} \in \{1,2,\ldots,9\} \tag{13}$$

$$\left(\sum_{r=1}^{10} h_r = h_{r+1}\right) = B \tag{14}$$

$$h_r \in \{0,1\} \qquad \forall r \in \{1,2,\ldots,11\}$$

(10) og (11) sikrer os, at hvert hold har 5 eller 6 hjemmebanekampe i del 1. (12) og (13) sikrer, at der i en periode på 3 runder er maksimalt 2 hjemmekampe (udekampe) og dermed ikke er to på hinanden følgende breaks (altså vore pattern-betingelse). Til sidst

[6] Vi anvender her CP, da CP er mere effektiv end IP til at generere feasible løsninger, jfr. afsnit 3.1

sikrer (14), at vi kun ser på mønstre med B breaks. B sættes, første gang problemet løses, til 0, og når vi vender tilbage, sættes B til 1,2,3, med $B^1 = 3$ i vores tilfælde.

For del 2's vedkommende ser modellen ud som følger:

$$\text{sequence}(1, 2, 3, [h_{12}, h_{13}, \ldots, h_{33}], [0, 1], [11, 11]) \tag{15}$$

$$\sum_{r=12}^{32} (h_r = h_{r+1}) = B^2 \tag{16}$$

$$h_{32} \neq h_{33} \tag{17}$$

$$h_r \in \{0, 1\} \qquad \forall r \in \{12, 13, \ldots, 33\}$$

(15) sikrer, at der i en sekvens på 3 på hinanden følgende runder (blandt runderne 12 til 33) mindst skal være 1 kamp hjemme (ude) og højst 2 kampe hjemme (ude). Altså kan der ikke opstå nogen på hinanden følgende breaks. Desuden skal der være 11 hjemmebanekampe og 11 udebanekampe i hvert mønster. (16) svarer til (14), blot for del 2. (17) sikrer, at der ikke kommer et break i sidste runde (den hårde af vores breakbetingelser). Grunden, til at vi ikke anvender constraint programming til del 1, er, at de 6 bedst placerede hold fra forrige sæson har en ekstra hjemmebanekamp. Dermed bliver sequence uanvendelig.

5.2 Generér HAP

Vi skal nu i gang med at finde HAP ud fra vores mængde af feasible mønstre, som vi har fundet i Trin 1 ovenfor. Vi skal på forhånd advare om, at der nu kommer en del notation. Lad P_i^p være mængden af de genererede mønstre, der opfylder de hårde placeringsbetingelser for hold i i del p. Lad nu $m \in P^p$ være et mønster, og lad x_{im}^p, $i \in N$, $m \in P^p$, $p \in \mathcal{P}$ være 1, hvis hold i bruger mønster m i del p og 0 ellers. Objektfunktionen i vores hovedproblem bliver nu:

$$\min \sum_{p \in \mathcal{P}} \sum_{i \in N} \sum_{m \in P_i^p} c_{im}^p x_{im}^p + \sum_{i \in N} c^{Br} \psi_i^{Br} + \left(\sum_{l \in C^{Ge}} \sum_{r \in R} c^{Ge} \psi_{lr}^{Ge} \right) + v$$

Vi vil nu forklare de forskellige led. c^{Br} er straffen, der tillægges, hver gang der opstår et break. c_m^{Br} er altså breakstraffen for at bruge mønster m. Tilsvarende er c_{im}^{Pl} straffen for at bruge mønster m til hold i, hvor et antal placeringsbetingelser er brudte, og c_m^{Be} er straffen for et break i runde 2. Disse tre potentielle straffe samles som $c_{im}^p = c_m^{Br} + c_{im}^{Pl} + c_m^{Be}$, $i \in N,\ m \in \{P^1, P^2\}$, $p \in \mathcal{P}$. Leddet

$$\sum_{p \in \mathcal{P}} \sum_{i \in N} \sum_{m \in P_i^p} c_{im}^p x_{im}^p$$

giver altså straf i objektfunktionen, hvis der er breaks i anden runde, breaks i nogle af de to dele eller brudte placeringsbetingelser. Det skal her bemærkes, at når vi udvælger mønstre, der skal "parres", er det på forhånd fastlagt, om (og dermed også hvor mange) af disse bløde constraints, der er brudt. For hvert mønster har c_{im}^p altså en bestemt værdi, der så kommer i spil, hvis mønsteret bruges (altså hvis $x_{im}^p = 1$).

Leddet

$$\sum_{i \in N} c^{Br} \psi_i^{Br}$$

giver en straf, hvis der er et break i første runde i del 2, dvs. variablen ψ_i^{Br}, $i \in N$ er 1, hvis hold i har et break i runde 1 i del 2 og 0 ellers. Vi lader nu $h_{mr},\ r \in R,\ m \in \{P^1, P^2\}$ være 1, hvis mønster m har en hjemmekamp i runde r og 0 ellers. Dette har vi to constraints, der sikrer; den ene, hvis der er tale om to udebanekampe i træk, og den anden to hjemmebanekampe. F.eks. er den for udebanekampene givet ved

$$\sum_{m \in P_i^1} h_{m,11} x_{im}^1 + \left(\sum_{m \in P_i^2} h_{m,12} x_{im}^2 \right) + \psi_i^{Br} \geq 1.$$

Desuden indfører vi variablen ψ_{lr}^{Ge}, der giver en straf i objektfunktionen, hvis de valgte mønstre gør, at bløde geografiske betingelser brydes. Hvis dette sker, kommer straffen c^{Ge} i spil. Her er l en geografisk constraint fra mængden $C^{Ge} = C_H^{Ge} \cup C_A^{Ge}$ af alle geografiske constraints (krav om hhv. hjemme- og udebanekampe), der sikrer, at mindst ét hold fra et område N_l^{Ge} (som har flere deltagende hold i turneringen) enten spiller hjemme eller ude i hver runde r. ψ_{lr}^{Ge} er 1, hvis den geografiske constraint l er brudt og 0 ellers. Altså giver leddet

$$\sum_{l \in C^{Ge}} \sum_{r \in R} c^{Ge} \psi_{lr}^{Ge}$$

en straf, hvis en geografisk betingelse brydes.

I objektfunktionen skal der gives en geografisk straf, hvis alle hold i en geografisk region, N_l^{Ge}, spiller ude (hjemme). Hvis $\sum_{i \in N_l^{Ge}} \sum_{m \in P_i^p} h_{mr} x_{im}^p = 0$, spiller holdene fra N_l^{Ge} ude i runde r, og ψ_{lr}^{Ge} skal være 1. Dette giver os betingelserne:

$$\left(\sum_{i \in N_l^{Ge}} \sum_{m \in P_i^p} (1 - h_{mr}) x_{im}^p \right) + \psi_{lr}^{Ge} \geq 1.$$

Til sidst indfører vi variablen v, som skal bruges til vores optimalitetscuts. Denne variabel vender vi tilbage til senere, men den bidrager med nogle straffe, hvis bløde constraints bliver brudt under tildeling af TT i trin 4.

Til at sikre, at der ikke opstår nogle på hinanden følgende breaks mellem del 1 og del 2, har vi fire constraints. De to første sikrer, at der er hhv. højst 2 og mindst 1 hjemmebanekamp i runderne 10-11-12, og de to sidste sikrer det samme for runderne 11-12-13. F.eks. sikrer følgende constraint, at der er højst 2 hjemmebanekampe i runderne 10-11-12:

$$\sum_{m \in P_i^1} (h_{m,10} + h_{m,11}) x_{im}^1 + \sum_{m \in P_i^2} h_{m,12} x_{im}^2 \leq 2.$$

Vi sætter også LB og UB til hhv. nedre og øvre grænse for vores objektfunktion. LB initialiseres til det mindste antal breaks, der kan forekomme jfr. Sætning 4.8, altså i vores tilfælde sættes $LB = 30$. UB kunne i teorien sættes til antal *mulige* breaks plus summen af alle bløde constraints overtrådt, men i praksis sættes den bare til et meget stort tal. IP-modellen for at finde HAP bliver nu: ([15], p7ff)

$$\min \sum_{p\in\mathcal{P}}\sum_{i\in N}\sum_{m\in P_i^p} c_{im}^p x_{im}^p + \sum_{i\in N} c^{Br}\psi_i^{Br} + \left(\sum_{l\in C^{Ge}}\sum_{r\in R} c^{Ge}\psi_{lr}^{Ge}\right) + v \tag{18}$$

$$\text{s.t.} \sum_{p\in\mathcal{P}}\sum_{i\in N}\sum_{m\in P_i^p} c_{im}^p x_{im}^p + \sum_{i\in N} c^{Br}\psi_i^{Br} + \left(\sum_{l\in C^{Ge}}\sum_{r\in R} c^{Ge}\psi_{lr}^{Ge}\right) + v \geq LB \tag{19}$$

$$\sum_{p\in\mathcal{P}}\sum_{i\in N}\sum_{m\in P_i^p} c_{im}^p x_{im}^p + \sum_{i\in N} c^{Br}\psi_i^{Br} + \left(\sum_{l\in C^{Ge}}\sum_{r\in R} c^{Ge}\psi_{lr}^{Ge}\right) + v \leq UB \tag{20}$$

$$\sum_{m\in P_i^p} x_{im}^p = 1, \qquad \forall i \in N,\ p \in \mathcal{P} \tag{21}$$

$$\sum_{i\in N} x_{im}^p \leq 1, \qquad \forall m \in P^p,\ p \in \mathcal{P} \tag{22}$$

$$\sum_{i\in N}\sum_{m\in P_i^p} h_{mr}x_{im}^p = 6, \qquad \forall r \in R^p, \forall p \in \mathcal{P} \tag{23}$$

$$\left(\sum_{i\in N_l^{Ge}}\sum_{m\in P_i^p} h_{mr}x_{im}^p\right) + \psi_{lr}^{Ge} \geq 1, \qquad \forall l \in C_H^{Ge},\ \forall r \in R^p,\ \forall p \in \mathcal{P} \tag{24}$$

$$\left(\sum_{i\in N_l^{Ge}}\sum_{m\in P_i^p} (1-h_{mr})x_{im}^p\right) + \psi_{lr}^{Ge} \geq 1, \qquad \forall l \in C_A^{Ge},\ \forall r \in R^p,\ \forall p \in \mathcal{P} \tag{25}$$

$$\sum_{m\in P_i^1} h_{m,11}x_{im}^1 + \left(\sum_{m\in P_i^2} h_{m,12}x_{im}^2\right) + \psi_i^{Br} \geq 1 \qquad \forall i \in N \tag{26}$$

$$\sum_{m\in P_i^1} (1-h_{m,11})x_{im}^1 + \left(\sum_{m\in P_i^2} (1-h_{m,12})x_{im}^2\right) + \psi_i^{Br} \geq 1 \qquad \forall i \in N \tag{27}$$

$$\sum_{m\in P_i^1} (h_{m,10}+h_{m,11})x_{im}^1 + \sum_{m\in P_i^2} h_{m,12}x_{im}^2 \leq 2, \qquad \forall i \in N \tag{28}$$

$$\sum_{m\in P_i^1} (h_{m,10}+h_{m,11})x_{im}^1 + \sum_{m\in P_i^2} h_{m,12}x_{im}^2 \geq 1, \qquad \forall i \in N \tag{29}$$

$$\sum_{m\in P_i^1} h_{m,11}x_{im}^1 + \sum_{m\in P_i^2} (h_{m,12}+h_{m,13})x_{im}^2 \leq 2, \qquad \forall i \in N \tag{30}$$

$$\sum_{m\in P_i^1} h_{m,11}x_{im}^1 + \sum_{m\in P_i^2} (h_{m,12}+h_{m,13})x_{im}^2 \geq 1, \qquad \forall i \in N \tag{31}$$

$$x_{im}^p \in \{0,1\}, \qquad \forall i \in N,\ \forall m \in P^p,\ \forall p \in \mathcal{P}$$

$$\psi_i^{Br} \in \{0,1\}, \qquad \forall i \in N$$

$$\psi_{lr}^{Ge} \in \{0,1\}, \qquad \forall l \in C^{Ge},\ \forall r \in R$$

$$v \in \mathbb{R}_+$$

(18) er vores objektfunktion i hovedproblemet, hvor de forskellige straffe kommer i spil. I (19) og (20) tildeler vi vores objektfunktion en nedre og øvre grænse. (21) sikrer, at hvert hold tildeles netop ét mønster i del 1 og 2, og (22) sørger for, at hvert mønster tildeles højst ét hold. (23) holder styr på, at halvdelen af holdene spiller hjemme i hver runde. Begrænsningerne (24) og (25) tildeler en straf, hvis ikke de geografiske constraints for de forskelige områder er overholdt (dvs. at ét hold spiller hhv. hjemme og ude i samme runde). (26) og (27) tildeler en straf, hvis der er et break i overgangen mellem del 1 og del 2. (28),(29),(30) og (31) sikrer, at der ikke kommer nogen på hinanden følgende breaks mellem del 1 og del 2. (28) og (29) ser på de sidste to runder i del 1 og den første i del 2, hvor de sikrer, at hvert hold har netop 1 eller 2 hjemmebanekampe. Tilsvarende ser (30) og (31) på den sidste runde i del 1 og de to første i del 2.

Hvis ovenstående model er infeasible, skal vi ifølge algoritmen gå tilbage til Trin 1 og generere flere mønstre. Findes en løsning, benævner vi den $(\bar{x}, \bar{\psi}^{Br}, \bar{\psi}^{Ge})$ og lader $\bar{P}^p$ være de mønstre, vi har valgt at bruge i del p.

5.3 Undersøg feasibility

Vi skal nu i Trin 3 undersøge, om det fundne HAP er feasible eller ej. Dette gøres ved først at undersøge feasibility af de to dele hver for sig og til sidst feasibility af de to dele kombineret.

Til kontrollen af del 1 kan vi anvende Proposition 4.21. I stedet for at se på alle delmængder af mønstre vælger vi at opskrive et problem (UBM), der finder den delmængde $\tilde{N}$, der har den mindste venstreside i følgende udtryk:

$$\sum_{r\in R^1} \min\left\{\sum_{m\in\tilde{N}} h_{mr}, \sum_{m\in\tilde{N}} (1-h_{mr})\right\} \geq \frac{|\tilde{N}|\left(|\tilde{N}|-1\right)}{2}.$$

Hertil indfører vi variablene $\alpha_m \in \{0,1\}$, der angiver, om mønster m ligger i den delmængde $\tilde{N}$ af mønstre i vores fundne HAP, vi ser på (1 hvis ja, 0 hvis nej). Desuden er $\delta_r \in \{0,1\}$ lig med 1, hvis vi ser på hjemmebanekampe og 0 ellers. Til sidst tæller $\beta_r \in \mathbb{R}_+$ antallet af kampe hjemme (ude) i hver runde s, afhængig af værdien af δ_r:

$$\min \sum_{r\in R^1} \beta_r \tag{32}$$

$$\text{s.t.} \sum_{m\in\bar{P}^1} \alpha_m = |\tilde{N}| \tag{33}$$

$$\beta_r - \sum_{m\in\bar{P}^1} h_{mr}\alpha_m + |\tilde{N}|(1-\delta_r) \geq 0, \qquad r \in R^1 \tag{34}$$

$$\beta_r - \sum_{m\in\bar{P}^1} (1-h_{mr})\alpha_m + |\tilde{N}|\delta_r \geq 0, \qquad r \in R^1 \tag{35}$$

$$\alpha_m, \delta_r \in \{0,1\}, \qquad m \in \bar{P}^1,\ r \in R^1$$

$$\beta_r \in \mathbb{R}_+, \qquad r \in R^1.$$

(32) svarer til $\sum_{r\in R^1} \min\left\{\sum_{m\in\tilde{N}} h_{mr}, \sum_{m\in\tilde{N}}(1-h_{mr})\right\}$. I (33) sikres det, at vi får netop det antal mønstre, der svarer til $|\tilde{N}| \in \{2,3,\ldots,12\}$. I (34) og (35) begrænses

β_r. Hvis der er færrest hjemmebanekampe, vil (34) blive dominerende, da β_r så skal være større end $\sum_{m\in\bar{P}^1} h_{mr}\alpha_m$ og tilsvarende vil (35) blive dominerende for færrest udebanekampe. Vi bemærker, at β_r altid vil ende med at være et heltal, da vi minimerer en funktion over binære variable. Grunden til, at β_r er reel, er, at dette gør problemet regnemæssigt hurtigere at løse. ([16], p12f)

I [16] opstilles følgende nødvendige betingelse for, at et HAP er feasible:

Definition 5.1. ***Mønstervariationsbetingelsen:***
Givet et HAP $\bar{P}^p$ og en delmængde $\tilde{N}$ af størrelse $|\tilde{N}|$, da er $\bar{P}^p$ feasible, hvis og kun hvis den optimale løsning til (UBM) er større end eller lig $\frac{|\tilde{N}|(|\tilde{N}|-1)}{2}$.

Er ovenstående betingelse ikke opfyldt i Trin 3, tilføjer vi Benders-cuttet

$$\sum_{i\in N}\sum_{m\in\hat{P}^1} x^1_{im} \leq |\tilde{N}| - 1,\ \hat{P}^1 = \{m \in \bar{P}^1 \mid \alpha_m = 1\}.$$

Altså udelukkes elementerne i delmængden $\hat{P}^1$ fra alle at indgå sammen i kommende HAP.

For at undersøge del 2 anvender vi mønstervariationsbetingelsen samt en anden betingelse, vi kommer tilbage til. Vores cut fra tidligere, $\sum_{i\in N}\sum_{m\in\hat{P}^2} x^2_{im} \leq |\tilde{N}| - 1$, kan genbruges, dog kan vi forstærke det yderligere. I del 2 skal holdene møde hinanden 2 gange, vi kan altså anvende betingelsen på begge halvdele af del 2. Hvis vi har en situation, hvor holdene ikke kan mødes i første haldel af del 2, kan mængden af mønstre $\hat{P}^2$ udvides til $\hat{P}^2_U$ med de mønstre, der har første halvdel af del 2 identisk med et af holdene i $\hat{P}^2$ (og tilsvarende for sidste halvdel). Altså tilføjer vi cuttet

$$\sum_{i\in N}\sum_{m\in\hat{P}^2_U} x^2_{im} \leq |\tilde{N}| - 1,$$

hvis en af halvdelene i del 2 viser sig at være infeasible.

Den anden betingelse, vi skal se på, indfører vi for at få overholdt vores separationsbetingelse. Vi introducerer til dette formål $\sigma_{m_1m_2} \in R$, som angiver den runde, hvor det hold, der anvender mønster m_1 spiller hjemme mod det hold, der anvender mønster m_2. Dette giver os CP-problemet (GAM):

$$(h_{m_1r} = 0) \vee (h_{m_2r} = 1) \Rightarrow (\sigma_{m_1m_2} \neq r), \quad \forall m_1, m_2 \in \hat{P}^2,\ m_1 \neq m_2,\ \forall r \in R^2 \quad (36)$$
$$\textit{all-different}(\textit{all}(m_2 \in \hat{P}^2 \backslash m_1)\sigma_{m_1m_2}, \textit{all}(m_2 \in \hat{P}^2 \backslash m_1)\sigma_{m_2m_1}), \quad \forall m_1 \in \hat{P}^2 \quad (37)$$
$$(\sigma_{m_1m_2} - \sigma_{m_2m_1} \leq -4) \vee (\sigma_{m_1m_2} - \sigma_{m_2m_1} > 4), \quad \forall m_1, m_2 \in \hat{P}^2,\ m_1 < m_2 \quad (38)$$
$$(\sigma_{m_1m_2} \leq 22) \Leftrightarrow (\sigma_{m_2m_1} \geq 23), \quad \forall m_1, m_2 \in \hat{P}^2,\ m_1 < m_2 \quad (39)$$
$$\sigma_{m_1m_2} \in R^2 \quad \forall m_1, m_2 \in \hat{P}^2,\ m_1 \neq m_2.$$

(36) sikrer, at hvis det hold, der har mønster nummer m_1, spiller ude i runde r, eller det hold, der har mønster nummer m_2, spiller hjemme i runde r, kan de to hold ikke mødes hos det hold, der har mønster nummer m_1, i runde r. Altså at et hold ikke kan spille en hjemmebanekamp i en runde, hvor det pågældende hold har udebane. I (37)

fordeles kampene i samme mønster, så hjemme- og udekampe mod alle de andre hold spilles i forskellige runder. F.eks. kan m_1 ikke møde m_2 på hjemmebane i samme runde, som m_2 møder m_1 på hjemmebane. (38) sikrer, at separationsbetingelsen bliver opfyldt. Her lader vi $k = 4$ være det antal runder, der skal være mellem to holds indbyrdes kampe. (39) sørger for, at to hold møder hinanden én gang i første halvdel og én gang i anden halvdel af del 2. Grunden til mønsternumrene $m_1 < m_2$ i (38) og (39) er, at vi ellers kunne få samme betingelse til at fremgå to gange.

Vi kan nu formelt definere den betingelse, der gør, at vi overhovedet kan anvende ovenstående:

Definition 5.2. ***Den multiple mønsterseparationsbetingelse:***
Givet et HAP $\bar{P}^p$ og en delmængde $\tilde{N}$ af $\bar{P}^p$, da er $\bar{P}^p$ feasible, kun hvis (GAM) har en feasible løsning.

Hvis vi kan finde en delmængde $\tilde{N}$, hvor ovenstående betingelse ikke er opfyldt, tilføjer vi følgende Benders cut til vores hovedproblem: ([15], p11f)

$$\sum_{i \in N} \sum_{m \in \hat{P}^2} x^2_{im} \leq |\tilde{N}| - 1.$$

Til slut i Trin 3 ser vi på feasibility af de to dele sat sammen. Først ser vi på, om alle par af to mønstre overholder separationsbetingelserne ved overgangen fra del 1 til del 2. Hvis dette ikke er tilfældet, hvor i_1 har mønster m^1_1 i del 1 og mønster m^2_1 i del 2, og tilsvarende har hold i_2 mønstrene m^1_2 og m^2_2, tilføjer vi så cuttet

$$x^1_{i_1 m^1_1} + x^2_{i_1 m^2_1} + x^1_{i_2 m^1_2} + x^2_{i_2 m^2_2} \leq 3$$

til vores hovedproblem. Vi vil ikke komme nærmere ind på resten af feasibility-undersøgelsen for de samlede dele, men blot nævne, at vi igen anvender Definition 5.2 på det samlede HAP. Her anvendes igen CP, og der tilføjes et cut, hvis dette HAP viser sig at være infeasible. Hvis det viser dig, at vores HAP ikke konstateres infeasible, fortsætter vi til Trin 4, ellers går vi tilbage til Trin 2, så snart vores cut er tilføjet. ([15], p12f)

5.4 Tildel timetable

I Trin 4 tildeler vi så endelig en TT til vores HAP. Til dette anvendes en IP-model, som vi ikke vil komme nærmere ind på her. I denne model indgår de resterende bløde betingelser, igen som "straffe" i vores objektfunktion, der skal minimeres, bl.a. tager vi her højde for følgende betingelser: topholdsbetingelser, kampbegrænsninger og holdbetingelser.

Hvis modellen viser sig at være infeasible, må mindst ét af de fundne mønstre skulle skiftes ud. Vi tilføjer derfor cuttet

$$\sum_{i \in N} (x^1_{i m^1_i} + x^2_{i m^2_i}) \leq 23$$

til vores hovedproblem og går tilbage til Trin 2. Ellers har vi fundet en feasible spilleplan for turneringen, og vores opgave er nu at prøve at finde en bedre. Dette gøres så ved at

opdatere vores UB lig værdien af objektfunktionen fra vores hovedproblem. Derudover tilføjer vi følgende optimalitetscut til vores hovedproblem:

$$v \geq v^{TT} \left(\sum_{i \in N} (x^1_{im^1_i} + x^2_{im^2_i}) - 23 \right),$$

hvor v^{TT} er objektfunktionsværdien af det TT, der lige er genereret. Denne constraint vil altså være overflødig, hvis vi ikke vælger nøjagtig samme HAP igen. Hvis vi vælger samme HAP igen, vil der gælde, at $v \geq v^{TT}$, og straffene fra Trin 4 kommer dermed til at indgå i vores hovedproblem. ([15], p13ff)

5.5 Metodens effektivitet

Udover at schedulere sæsonen 2006/07 har Rasmussen testet ovenstående metode for sæsonen 2005/06 og sammenlignet med tilfældigt genererede instanser. I sæsonen 2005/06 blev k fastsat til 0, da alle holdbetingelser ellers ikke ville kunne være opfyldt. Rasmussen har scheduleret selvsamme sæson for k løbende fra 0 til 4. I visse tilfælde kan det være en hastighedsmæssig fordel også at indføre en constraint, som sikrer, at alle mønstre er parvist komplementære. Generelt har algoritmen svært ved at bevise optimalitet inden for tidsgrænsen på 1800 sekunder[7], men det kan diskuteres, hvorvidt man overhovedet kan tale om optimalitet, da det er et vægtningsspørgsmål, hvor store de forskellige "straffe" for at bryde constraints skal være. Desuden kan det være en fordel for et forbund at have nogle forskellige spilleplaner at vælge imellem.

I nedenstående skema ses resultaterne af de forskellige tests. $ki,\ i = 0, 1, \ldots, 4$ angiver k-værdien af testen, og +komp angiver, at komplementær-constrainten er tilføjet. Tallene i skemaet angiver, hvor mange af de pågældende bløde constraints, der er brudt. Af de udførte tests er kun $k1$+komp vist optimal inden for tidsgrænsen.

Test	Plac. (29)	Kamp (1)	Top (10)	Hold (1)	Beg. (10)	Geo. (3)	Break (372)	I alt (426)
Spilleplan 05/06	3	0	2	0	2	8	46	61
$k0$	0	0	1	1	2	1	36	41
$k0$+komp	0	1	1	0	2	0	38	42
$k1$	0	0	1	1	2	1	36	41
$k1$+komp	0	0	1	0	2	0	38	41
$k2$	0	1	1	0	2	1	36	41
$k2$+komp	0	0	1	1	2	0	38	42
k	0	0	1	0	2	1	38	42
$k3$+komp	0	0	1	1	2	0	38	42
$k4$	0	0	1	0	2	1	38	42
$k4$+komp	0	1	1	0	2	0	38	42

Skema 3

Metoden viser sig altså at være meget brugbar, idet den kan reducere antallet af brudte constraints ganske betragteligt (over 30%), selvom k forøges. Altsammen inden

[7] Tidsgrænsen er fastsat af Rasmussen i forbindelse med testen.

for kort tid. For SAS-ligaens vedkommende ser vi i skemaet, at der altid er brudte topholdsbetingelser og breakbetingelser vedr. breaks i anden spillerunde (Beg.). Grunden til det første er, at begge tophold hver har 5 udebanekampe i del 1, og derfor vil der være mindst én runde i del 1, hvor ingen af dem spiller ude mod et hold fra N^{To}. Grunden til det andet er, at ét af holdene har ønsket at få lov til at spille to udebanekampe i runde 1 og 2. ([15], p15ff)

6 Konklusion

Vi har i dette projekt vist, at heltalsprogrammering, constraint programming og Benders decomposition kan bruges sammen i praksis inden for sports scheduling. Desuden har vi vist, at den generelle matematiske teori inden for emnet bruges til f.eks. fastsættelse af nedre grænser for breaks og til at træffe feasibilityafgørelser. Herudover kan vi konkludere, at matematiske modeller er langt mere effektive i forhold til tidligere års "manuelle" forsøg på schedulering af den bedste danske fodboldliga, SAS-ligaen.

Sports scheduling er et glimrende eksempel på problemløsningsmetoder, hvor man med fordel kan kombinere heltalsprogrammering og constraint programming, og brugbarheden af denne anvendelse vil efter al sandsynlighed føre til yderligere forskning inden for området. Sports scheduling som helhed vil ligeledes blive uundværlig ved større turneringer, i takt med at turneringerne kommercialiseres og får flere constraints, der umuliggør en "manuel" løsning af problemet.

Alt i alt må vi konkludere, at sports scheduling i mange år fremover vil være et stadig mere anvendt redskab, der vil afføde megen ny forskning.

Litteratur

[1] K. Easton, G. Nemhauser, M. Trick: *The Travelling Tournament Problem - Description and Benchmarks*, artikel, (2001).

[2] M. Elf, M. Jünger, G. Rinaldi: *Minimizing Breaks by Maximizing Cuts*, i *Operations Research Letters 31*, p333-349, (2003).

[3] S. Fredens: *Deterministiske Sekvensmodeller*, forelæsningsnoter, IFOR, Århus Universitet (1981).

[4] M. Henz, T. Müller, S. Thiel: *Global Constraints for Round Robin Tournament Scheduling*, artikel (2003).

[5] F.S. Hillier, G.J. Lieberman: *Introduction to Operations Research, Eighth Edition*, MacGraw-Hill (2005).

[6] J.N. Hooker, G. Ottoson: *Logic-Based Benders Decomposition*, i *Mathematical Programming 96*, p33-60 (2003).

[7] J.N. Hooker: *Logic-Based Methods for Optimization*, John Wiley & Sons, Inc. (2000).

[8] J.Y-T. Leung: *Handbook of Scheduling*, Chapman & Hall/CRC (2004).

[9] R. Miyashiro, H. Iwasaki, T. Matsui: *Characterizing Feasible Pattern Sets with a Minimum Number of Breaks*, i *Lecture Notes in Computer Science vol. 2740*, p78-99 Springer Berlin / Heidelberg (2003).

[10] R. Miyashiro, T. Matsui: *A Polynomial-time algorithm to find an equitable home-away aasignment*, i *Operations Research Letters 33*, p235-241 (2005).

[11] R. Miyashiro, T. Matsui: *Round-Robin Tournaments with s Small Number of Breaks*, URL `http://citeseer.ist.psu.edu/miyashiro03roundrobin.html` (2003).

[12] G.L. Nemhauser, M.A. Trick: *Scheduling a major college basketball conference*, i *Operations Research 46(1)*, p1-8 (1998).

[13] G.L. Nemhauser, L.A. Wolsey: *Integer and Combinatorial Optimization*, Wiley (1988).

[14] G. Post, G. J. Woeginger: *Sports tournaments, home-away assignments, and the break minimization problem*, i *Discrete Optimization vol. 3,2*, p165-173 (2006).

[15] R.V. Rasmussen: *Scheduling a triple round robin tournament for the best Danish soccer league*, Working Paper no. 2006/1, Department of Operations Research, Århus Universitet (2006).
URL `http://www.imf.au.dk/publs?id=596`

[16] R.V. Rasmussen, M.A. Trick: *A Benders approach for the constrained minimum break problem* (2006).
URL `http://mat.gsia.cmu.edu/trick/benders.pdf`

[17] R.V. Rasmussen, M.A. Trick: *Round Robin Scheduling - a Survey*, artikel (2006).

[18] J.-C. Régin: *Minimization of the Number of Breaks in Sports Scheduling Problems using Constraint Programming*, i *DIMACS Series in Discrete Mathematics and Theoretical Computer Science 57*, p115-130, (2001).

[19] J.A.M. Schreuder: *Combinatorial aspects of construction of competition Dutch Professional Football Leagues*, i *Discrete Applied Mathematics 35*, p301-312 (1992).

[20] M.A. Trick: *Integer and Constraint Programming Approaches for Round Robin Tournament Scheduling*, i *Lecture Notes in Computer Science vol. 2740*, p63-77 Springer Berlin / Heidelberg (2003).

[21] M.A. Trick: *A schedule-then-Break Approach to Sports Timetabling*, i *Lecture Notes in Computer Science vol. 2079*, p242-252, Springer Berlin / Heidelberg (2001).

[22] D. de Werra: *Scheduling in Sports*, i *Annals of Discrete Mathemathics 11* redigeret af P. Hansen, North-Holland (1981).

[23] L. A. Wolsey: *Integer Programming*, John Wiley & Sons, Inc. (1998).