JOB-SHOP-SKEDULERING OG TOGSKEDULERING

Christian Schmidt

LYNGBY 2002
EKSAMENSPROJEKT
NR. 34/02

IMM

Trykt af IMM, DTU

Forord

Denne rapport præsenterer resultaterne af mit eksamensprojekt ved Sektion for Operationsanalyse, Institut for Matematisk Modelling (IMM), Danmarks Tekniske Universitet (DTU).

Projektet indgår som den afsluttende opgave i forbindelse med studiet til civilingeniør.

Vejleder på projektet har været professor Jens Clausen.

Lyngby, 3. juni 2002

Christian Schmidt

Abstract

This thesis investigates the job-shop scheduling problem with emphasis on a particular application, namely railway scheduling. Two different objective functions are examined: makespan and total weighted tardiness.

Different solution strategies are applied, including Branch & Bound with two different branching strategies and the Shifting Bottleneck heuristic. Most methods are mentioned in different variants for the two objective functions.

Performance is measured with different test instances from in the literature.

A bounding function for total weighted tardiness is developed and investigated. Also two different bounding functions for makespan are examined. A local search heuristic is used to find tighter bounds, and this improves the performance marginally.

For Branch & Bound a new dominance criterion is suggested for pruning the search tree. With generel job-shop instanses this does not lead to increased performance, but for train scheduling instanses the results look promising.

Keywords: job shop scheduling; single track railway scheduling; total weighted tardiness; makespan; branch and bound; dominance criteria; shifting bottleneck

Indhold

Kapitel 1

Indledning

Job-shop-skedulering er et klassisk problem inden for operationsanalysen. I denne rapport betragtes en konkret anvendelse, planlægning af togafgange på et jernbanenet med begrænset kapacitet.

1.1 Problemformulering

Med udgangspunkt i Elias Oliveira og Barbara M. Smith: "A Hybrid Constraint-Based Method for the Single-Track Railway Scheduling" [17], ønsker vi at undersøge forskellige løsningsmetoder — såvel eksakte som heuristiske — for job-shop-skeduleringsproblemet. Vi ønsker desuden at betragte en anvendelse heraf, nemlig togskedulering.

Særligt ønsker vi at undersøge og evt. forbedre branch-and-bound-metoden omtalt i artiklen.

1.2 Oversigt over rapporten

I kapitel 2 defineres togskeduleringsproblemet, og ovennævnte artikel diskuteres. I kapitel 3 defineres job-shop-skeduleringsproblemet, og de centrale størrelser og begreber introduceres, og i kapitel 4 formuleres togskeduleringsproblemet så som et job-shop-skeduleringsproblem.

Kapitel 5 omtaler forskellige løsningsmetoder: to metoder til branch-and-bound samt en heuristik. I forbindelse hermed introduceres en ny bounding-funktion samt et nyt dominanskriterium til beskæring af søgetræet.

I kapitel 6 præsenteres de opnåede beregningsmæssige resultater. Som test-data benyttes bl.a. kendte instanser fra litteraturen.

Endelig konkluderes i kapitel 7, hvad der er opnået i forbindelse med dette projekt.

I kapitel 3 introduceres en del symboler og variablenavne, der benyttes i resten af rapporten. For overskueligheds skyld findes i bilag A en oversigt over disse.

Det forudsættes, at læseren er bekendt med de grundlæggende begreber i operationsanalyse, herunder særligt branch-and-bound.

Det implementerede programmel samt de benyttede testinstanser kan downloades fra http://christianschmidt.dk/eksproj .

Kapitel 2

Togskedulering

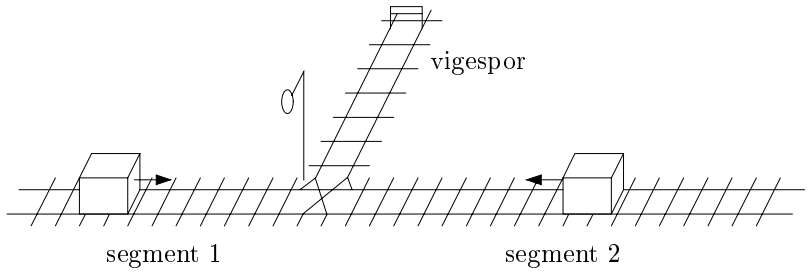
Togskedulering består i sin mest generelle form i at lægge en køreplan for et antal tog, der kører på samme jernbanestrækning. Jernbanestrækningen består udelukkende af enkeltspor, som det eksempelvis kendes fra lokalbaner og andre mindre trafikerede strækninger, dvs. at tog kun kan passere hinanden særlige steder. Disse steder vil i det følgende blive omtalt som vigespor. Ofte vil vigespor ligge i forbindelse med en station.

Ved køreplan forstås en angivelse af, hvor hvert tog skal befinde sig på ethvert givet tidspunkt, herunder ved hvilke vigespor togene skal passere hinanden. Denne planlægning sker på forhånd, som regel længe før første togafgang. Der er således ikke tale om realtidsplanlægning, som det eksempelvis sker i et centralt kontrolcenter for et jernbanenet, hvor man løbende følger driften og løser akutte problemer opstået som følge af forsinkelser.

En særlig disciplin af togskedulering kaldes single-track railway scheduling (STRS). Her antages, at jernbanestrækningen bortset fra vigespor er uforgrenet.

I denne formulering antages det, at strækningen er opdelt i et antal sporsegmenter, der er adskilt af stopsignaler. Hvor to sporsegmenter støder sammen, findes et vigespor. Her kan to modkørende tog passere hinanden, ved at det ene holder ind på vigesporet, mens det andet kører forbi. Antallet og placeringen af signaler og vigespor tages for givet.

For at undgå kollisioner må der kun befinde sig ét tog ad gangen på hvert



Figur 2.1: To modkørende tog ved vigespor.

sporsegment. Ankommer et tog således til et segment, der er optaget, må det holde for rødt og vente, indtil der er fri bane.

I figur 2.1 ses to tog, der kommer kørende ad hver sit sporsegment. For at passere hinanden må det ene tog køre ind på vigesporet og vente, til det andet er kørt forbi.

2.1 Omtale af litteratur

Som nævnt tager denne rapport udgangspunkt i Elias Oliveira and Barbara M. Smith: "A Hybrid Constraint-Based Method for the Single-Track Railway Scheduling Problem" [17].

Her følger en uddybning og diskussion af artiklen, der bl.a. er baseret på korrespondence med den ene af artiklens forfattere, Elias Oliveira, der også har været så venlig at stille de benyttede testinstanser til rådighed.

2.1.1 Den hybride metode

I artiklen beskrives en metode, der er en kombination af branch-and-bound og lokalsøgning. Beskrivelsen af samspillet mellem de to metoder er ikke helt klar, så her følger en kort gennemgang.

Algoritmen er bygget op om en almindelig branch-and-bound-søgning. Når søgningen når et blad i søgetræet og således har fundet en brugbar løsning, gives denne løsning videre til en lokalsøgningsalgoritme.

Hvis det lykkes lokalsøgningen at finde en bedre løsning, benyttes løsningsværdien heraf som ny øvre grænse. Selve løsningen bruges ikke i den videre branch-and-bound-søgning — dog lagres den selvfølgelig som den hidtil bedste løsning.

Branch-and-bound bruges altså til at komme rundt i løsningsrummet, og lokalsøgningen bruges så til at undersøge nabolaget omkring de brugbare løsninger, branch-and-bound-søgningen finder.

Som nedre grænse for objektfunktionsværdien af den optimale løsning af en given knude i søgetræet bruges tilsyneladende blot objektfunktionsværdien af den aktuelle ikke-brugbare løsning.

2.1.2 Resultater

I artiklens afsnit 2 omtales fire yderligere typer af mulige betingelser til en løsning, herunder at to tog altid skal holde en given afstand, eksempelvis ved transport af farligt gods, og at et sporsegment kan være blokeret i en periode, for eksempel i forbindelse med vedligeholdelse.

Det er dog ikke beskrevet, hvilke metoder der er benyttet for at sikre, at disse betingelser er opfyldt. Der er tilsyneladende heller ikke stillet betingelser af denne art i forbindelse med de rapporterede testkørsler.

Ligeledes anføres, at det ikke er en forudsætning, at der er vigespor mellem alle sporsegmenter — en forudsætning som ellers gøres i denne rapport (dette omtales i afsnit 4.2)[1]. Denne bekvemme forudsætning gør sig dog gældende i de instanser, der er brugt til de rapporterede testkørsler.

[1] Bemærk at det ikke giver et ækvivalent problem blot at slå nabosegmenter, der ikke er adskilt af vigespor, sammen. Jo flere segmenter en given strækning er opdelt i, jo hurtigere kan to tog i samme retning nemlig passere strækningen, idet det bageste tog ikke skal vente på, at det forreste har forladt hele strækningen men kun det tilstødende segment.

Kapitel 3

Introduktion til job-shop-skedulering

Job-shop-skedulering (JSS) er et klassisk problem inden for operationsanalysen, der har været genstand for megen opmærksomhed. Det betragtes som et af de sværerste problemer, og selv tilsyneladende simple instanser viser sig umulige at løse til optimalitet. Heller ikke med heuristiske løsningsmetoder har man formået at nå løsninger af samme kvalitet, som man kender fra andre problemer. Ofte vil eksakte løsningsmetoder endda være mere effektive end heuristiske.

I det følgende defineres JSS, og der introduceres de begreber, der er anvendt i beskrivelsen af løsningsmetoderne i kapitel 5.

3.1 Generel JSS

I sin mest generelle formulering består JSS i at planlægge behandlingen af et antal *jobs* på et antal *maskiner* under hensyntagen til maskinernes kapacitet.

Jobs kan opfattes som projekter, der hver består af en række delopgaver. En maskine udgør en ressource, der skal være til rådighed på tidspunktet for

udførelsen af en given enkeltopgave. Problemet består således i at planlægge delopgaverne under hensyntagen til ressourcernes tilgængelighed.

Mange situationer fra hverdagen kan modelleres på denne måde. Et eksempel er et butikstorv, hvor et antal kunder kommer for at handle. Hver kunde har en række ærinder, der skal nås, eksempelvis først et besøg i bankens hæveautomat og herefter hos grønthandleren og slagteren i vilkårlig rækkefølge. Hver kunde svarer til et job, og hvert ærinde svarer til en delopgave. Ressourcerne udgøres af bankautomaten samt ekspedienterne i forretningerne, der hver kun kan betjene én kunde ad gangen.

I en situation som denne sker planlægningen næsten helt tilfældigt. Som regel vælger kunden at besøge forretningerne i en på forhånd fastlagt rækkefølge og stiller sig blot i kø, hvis ekspedienten er optaget ved ankomsten.

Andre steder sker en overordnet planlægning, uden den dog eksplicit er repræsenteret vha. ovennævnte begreber. Et eksempel herpå er trafiklysene i vejkrydsene i en by, der ofte kontrolleres et centralt sted fra. Her udgør ressourcerne vejbanen midt i et kryds, der kun kan benyttes af biler i et begrænset antal retninger ad gangen. Hvert køretøj har en bestemt destination, og delopgaverne består af de lyskryds, hvert køretøj skal passere på vej dertil.

Modsat i førnævnte eksempel sker der her en decideret bevidst planlægning. Ved hjælp af bl.a. "grønne bølger" langs hovedfærdselsårerne søger man at afvikle trafikken så smidigt som muligt.

3.2 Simpel JSS

JSS findes i mange afarter, hvor problemformulering og løsning er pålagt forskellige begrænsninger. Hovedparten af teorien på området koncentrerer sig om en særlig gruppe varianter kaldet *simpel job-shop-skedulering*. Denne rapport beskæftiger sig udelukkende med varianter heraf, og herefter vil betegnelsen JSS udelukkende benyttes til at omtale simpel JSS.

Simpel JSS består i at planlægge behandlingen af et antal jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ på et antal maskiner $\mathcal{M} = \{M_1, \ldots, M_m\}$. Hvert job skal behandles på hver maskine i en rækkefølge, der er specifik for det enkelte job. Behandlingen af et job på en maskine betegnes en *operation*. Problemet består i at bestemme en rækkefølge, hvori de enkelte operationer skal behandles på de enkelte maskiner.

Et eksempel er den type problemer, der har givet navne til begreberne i JSS, nemlig en produktion, hvor der skal laves en række produkter. Hvert produkt skal behandles på et antal maskiner i en bestemt rækkefølge, og hver maskine kan kun behandle ét produkt ad gangen.

Hvert job består af en ordnet liste af operationer, der hver er tilknyttet netop én maskine. Hvert job behandles på hver maskine netop én gang. En operation kan først påbegyndes, når den foregående operation på samme job er afsluttet. Hver maskine kan kun behandle ét job ad gangen, og en operation kan ikke afbrydes, når først den er påbegyndt. Det antages, at alle maskiner kan benyttes i hele planlægningshorisonten, dvs. der opereres ikke med tid til vedligeholdelse eller nedbrud.

En løsning består i angivelse af et starttidspunkt for hver operation.

Selvom denne rapport begrænser sig til at betragte dette noget specialiserede problem, vil hovedparten af metoderne beskrevet heri næsten umiddelbart kunne benyttes på mere generelle problemer. Især er det nemt at afvige fra betingelserne om, at hvert job skal bestå af lige mange operationer, samt at hvert job behandles netop én gang på hver maskine.

3.3 Definitioner

Hvert job J_i består af en ordnet liste af operationer $\{o_{i1}, \ldots, o_{im}\}$ samt et givet tidspunkt for den tidligst tilladte afvikling af jobbets første operation $r_i \geq 0$. I mange formuleringer af JSS er r_i begrænset til værdien 0, men i denne fremstilling betragtes altså det lidt mere generelle problem.

Hver operation o_{ij} er tilknyttet en givet maskine $M_p \in \mathcal{M}$. Notationsmæssigt opfattes en maskine som en mængde indeholdende de tilhørende operationer, én fra hvert job.

For hver operation er tillige givet en *behandlingstid* $p_{ij} \geq 0$, dvs. den tid det tager at behandle operationen o_{ij}. Variablen S_{ij} angiver *starttidspunktet* for afvikling af en operation i den aktuelle løsning, og C_{ij} angiver *afslutningstidpunktet*. Da behandlingen af en operation ikke kan afbrydes, afsluttes behandlingen netop p_{ij} tidsenheder efter dens påbegyndelse, dvs. $C_{ij} = S_{ij} + p_{ij}$.

En operation o_{ij} kan først påbegyndes, når den forrige operation på samme

job $o_{i(j-1)}$ er afsluttet, dvs.

$$S_{ij} \geq S_{i(j-1)} + p_{i(j-1)} \quad \text{for } j \geq 2 \tag{3.1}$$

Starttidspunkttet for den første operation på et job o_{i1} er begrænset af jobbets tidligste starttidspunkt r_i, dvs. $S_{i1} \geq r_i$.

Heraf følger en nedre grænse for S_{ij} baseret på jobbets tidligste starttidspunkt samt varigheden af de forudgående operationer:

$$S_{ij} \geq r_i + \sum_{l<j} p_{il} \tag{3.2}$$

Færdiggørelsestidspunktet C_i for et job J_i er, når behandlingen af jobbets sidste operation er afsluttet. Således er

$$C_i = C_{im} = S_{im} + p_{im} \tag{3.3}$$

En operation kan ikke afbrydes, når først den er påbegyndt. En maskine tilknyttet en given operation o_{ij} kan således kun behandle denne ene operation i perioden $[S_{ij}; C_{ij}[$. Det betyder, at to operationer o_{ij}, o_{kl} på samme maskine M_p ikke kan overlappe tidsmæssigt, dvs. at

$$[S_{ij}; C_{ij}[\cap [S_{kl}; C_{kl}[= \emptyset \quad \text{for alle } o_{ij}, o_{kl} \in M_p \tag{3.4}$$

Hvis en løsning ikke overholder denne betingelse, er løsningen ikke-brugbar, og der tales om en *konflikt*, der involverer de overlappende operationer. Dette er den eneste af de her nævnte betingelser, der ikke er overholdt i de ikke-brugbare løsninger, der omtales i denne fremstilling.

Ved visse anvendelser opererer man med en *deadline* d_i for hvert job. Denne deadline skal ikke opfattes som et obligatorisk krav til en brugbar løsning men blot som en parameter, der indgår i objektfunktionen. På baggrund heraf defineres et jobs *tardiness* som

$$T_i = (C_i - d_i)^+ \tag{3.5}$$

altså jobbets forsinkelse — om nogen — i forhold til den givne deadline.

3.4 Objektfunktioner

Afhængigt af det konkrete problem anvendes flere forskellige objektfunktioner til løsning af JSS.

Den i litteraturen hyppigst omtalte er *makespan*. Her ønsker man at minimere tidspunktet for afslutning af det sidste job:

$$O_{makespan} = \max_{J_i \in \mathcal{J}}(C_i) \tag{3.6}$$

En anden udbredt objektfunktion er minimering af *total tardiness* (TT), $\sum T_i$. En variation heraf, *total weighted tardiness* (TWT), giver mulighed for at tildele de enkelte jobs en prioritet $w_i > 0$, der bruges til at vægte jobbets forsinkelse:

$$O_{TWT} = \sum_{J_i \in \mathcal{J}} w_i T_i \tag{3.7}$$

Ved makespan er det kun det sidst afsluttede job, der får betydning for objektfunktionsværdien, mens TWT afhænger af tidspunktet for hvert enkelt jobs færdiggørelse. Makespan vil således typisk bruges ved løsning af opgaver, hvor alle jobs skal være afsluttede, før det samlede produkt har nogen nytteværdi. Dette kunne eksempelvis være, hvis resultatet af de enkelte jobs skal afsendes i samme sending. I modsætning hertil er TWT, der eksempelvis kan benyttes ved en produktion, hvor hvert job repræsenterer en vare, der kan sælges umiddelbart efter dets færdiggørelse.

Ovennævnte objektfunktioner er — såvel som de fleste andre omtalt i litteraturen — såkaldte *regulære* objektfunktioner. En regulær objektfunktion er en funktion, der søges minimeret, og hvor jobbenes afslutningstidspunkter C_i indgår som de eneste variable, dvs. at objektfunktionen kan udtrykkes $O(C_1, \ldots, C_n)$.

Derudover gælder, at funktionen kun stiger, når mindst én af de indgående afslutningstider stiger, dvs. at for to løsninger med afslutningstiderne hhv. $C_1, \ldots, C_n$ og $C'_1, \ldots, C'_n$, da vil $O(C_1, \ldots, C_n) > O(C'_1, \ldots, C'_n)$ kun i det tilfælde, at der findes mindst ét job J_i, for hvilket det gælder, at $C_i > C'_i$.

3.5 Grafrepræsentation

Job-shop-skeduleringsproblemet ses ofte repræsenteret ved en orienteret, acyklisk graf. Udover at give mulighed for at visualisere en given løsning, er visse begreber nemmere at forklare med udgangspunkt i en graf. I denne rapport benyttes betegnelserne fra selve formuleringen af JSS synonymt med deres ækvivalenter i grafrepræsentationen.

Knuderne i grafen repræsenterer de enkelte operationer, og kanternes retning angiver den indbyrdes rækkefølge to operationer imellem i den aktuelle løsning. En kant $o_{ij} \rightarrow o_{kl}$ angiver således, at o_{ij} udføres før o_{kl}. Den mindste tidsmæssige afstand mellem påbegyndelse af de to operationer er p_{ij}, dvs. $S_{kl} \geq S_{ij} + p_{ij}$. Denne størrelse kaldes i grafsammenhæng kantens *længde* og skrives $|o_{ij} \rightarrow o_{kl}|$.

Starttidspunktet S_{kl} bestemmes ved at betragte de indgående kanter og de tilhørende operationer, dvs. de operationer, der går umiddelbart forud for o_{ij}. Som følge af ovenstående gælder således, at

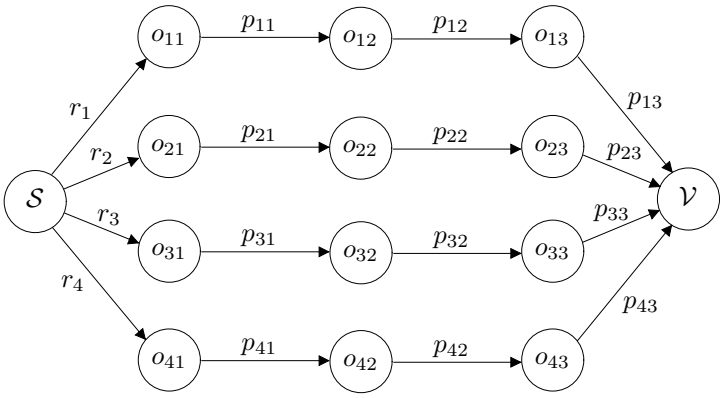
$$S_{kl} \geq \max_{o_{ij} \rightarrow o_{kl}} (S_{ij} + |o_{ij} \rightarrow o_{kl}|) \tag{3.8}$$

For regulære objektfunktioner kan operationer uden ulempe behandles så tidligt som muligt under hensyntagen til de indgående kanter, dvs. så snart alle de forudgående operationer er afsluttede. Derfor vil vi i denne rapport beregne S_{kl} i en brugbar løsning ved at erstatte ulighedstegnet med lighedstegn i ovenstående. I ikke-brugbare løsninger anvendes S_{kl} som en betegnelse for en nedre grænse for starttidspunktet i en brugbar løsning. Hvordan denne grænse beregnes, vil blive omtalt i afsnit 3.7.

Initielt tilføjes kanter, der svarer til den givne rækkefølge af operationer for hvert job, dvs. at for J_i tilføjes kanterne $o_{ij} \rightarrow o_{i(j+1)}$ med længden p_{ij} for $j \in \{1, \ldots, n-1\}$. $S_{i(j+1)}$ opdateres i overensstemmelse med (3.8) og antager hermed minimumsværdien angivet i (3.2). En løsning bestående udelukkende af disse kanter vil blive omtalt som den *initielle løsning*.

Der tilføjes en kildeknude $\mathcal{S}$ med starttidspunktet 0. Starttidspunkterne for hvert enkelt job J_i repræsenteres ved at tilføje kanter $\mathcal{S} \rightarrow o_{i1}$ med længden r_i.

Figur 3.1 viser den initielle løsning for et problem bestående af fire jobs og tre maskiner. Desuden er tilføjet en afløbsknude for beregning af makespan, hvilket er omtalt i næste afsnit.



Figur 3.1: Initiel løsning med afløbsknude for makespan.

Bemærk at det ikke fremgår af grafen, hvilken maskine en given operation hører til.

3.5.1 Repræsentation af objektfunktionsværdi

Ved beregning af makespan introduceres en afløbsknude $\mathcal{V}$, der er forbundet til knuden svarende til den sidste operation på hvert job J_i vha. kanterne $o_{im} \rightarrow \mathcal{V}$ som vist på figur 3.1. Objektfunktionsværdien vil da være lig starttidspunktet for pseudo-operationen repræsenteret ved $\mathcal{V}$.

Ved brug af objektfunktionen TWT introduceres en afløbsknude $\mathcal{V}_i$ for hvert job J_i, der er forbundet til hvert jobs sidste operation med kanterne $o_{im} \rightarrow \mathcal{V}_i$. Derudover repræsenteres jobbenes deadline ved at tilføje en kant $\mathcal{S} \rightarrow \mathcal{V}_i$ fra kildeknuden til hver afløbsknude med længden d_i. Knuden $\mathcal{V}_i$ har således netop to indgående kanter, og jvf. (3.8) bliver starttidspunktet for den tilsvarende pseudo-operation

$$\begin{aligned} S_{\mathcal{V}_i} &= \max_{o_{ij} \rightarrow \mathcal{V}_i} (S_{ij} + |o_{ij} \rightarrow \mathcal{V}_i|) \\ &= \max(S_{im} + p_{im}, 0 + d_i) \\ &= \max(C_i, d_i) \\ &= \max(C_i - d_i, 0) + d_i \\ &= T_i + d_i \end{aligned}$$

Sidste lighedstegn følger af definitionen af tardiness (3.5). Tardiness T_i af et enkelt job er således tidspunktet for afvikling af pseudo-operationen $\mathcal{V}_i$ fratrukket jobbets deadline d_i. Det giver følgende udtryk for den samlede objektfunktionsværdi:

$$O_{TWT} = \sum_i w_i(S_{\mathcal{V}_i} - d_i) = \sum_i w_i S_{\mathcal{V}_i} - \sum_i w_i d_i$$

Omskrivningen til højre for lighedstegnet er gjort for at understrege, at det kun er første sumtegn, der skal minimeres, idet d_i er en konstant.

3.5.2 Repræsentation af løsning

En brugbar løsning er repræsenteret ved en acyklisk graf, hvor der for hver maskine findes en vej, der går gennem alle knuder tilknyttet maskinen. Dette sikrer opfyldelse af betingelse (3.4), nemlig at der ikke findes overlappende operationer på den aktuelle maskine. Denne repræsentation er dog ikke unik. Hvis der eksempelvis i en given løsning findes to kanter $o_{gh} \rightarrow o_{ij}$ og $o_{ij} \rightarrow o_{kl}$, da vil kanten $o_{gh} \rightarrow o_{kl}$ kunne tilføjes eller fjernes grafen uden at ændre på løsningen.

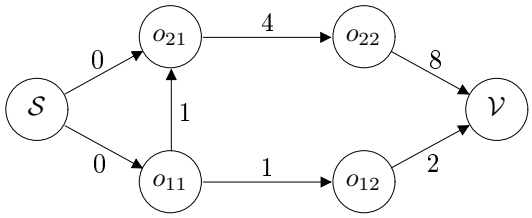
Idet operationer som før nævnt udføres så tidligt som muligt under hensyntagen til (3.8), vil en brugbar løsning repræsenteret ved kanterne i grafen være ækvivalent med en løsning repræsenteret ved angivelse af starttidspunkterne for hver enkelt operation.

En tredje ækvivalent måde at repræsentere en brugbar løsning er således ved at angive en rækkefølge af operationerne på hver enkelt maskine.

En given løsning kan godt være brugbar, uden ovennævnte krav om en vej gennem alle knuder tilknyttet hver maskine er opfyldt. Dog kan der til en sådan løsning umiddelbart tilføjes kanter, således at kravet opfyldes, uden løsningen ændres.

Figur 3.2 viser et eksempel på dette. Her er $o_{12}, o_{22} \in M_p$, $S_{12} = 1$ og $C_{12} = 3$, mens $S_{22} = 5$. Løsningen er brugbar, men der er ingen vej gennem operationerne på maskinen M_p. Da o_{12} er afsluttet, før o_{22} påbegyndes — $C_{12} < S_{22}$ — vil der dog umiddelbart kunne indsættes en kant $o_{12} \rightarrow o_{22}$, uden selve løsningen påvirkes, hvorved kravet om en vej er opfyldt.

Notationsmæssigt er en løsning E en mængde bestående af kanterne i løsningen, bortset fra kanterne $o_{ij} \rightarrow o_{i(j+1)}$ mellem operationer på samme



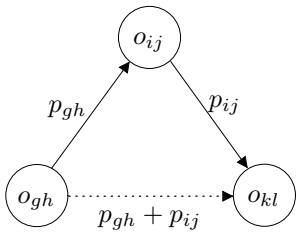
Figur 3.2: En gyldig løsning, selvom der ikke er en vej $o_{12} \rightsquigarrow o_{22}$.

job, idet sidstnævnte jo indgår i alle løsninger. Den initielle løsning betegnes således $\emptyset$.

3.5.3 Kantlængder

Længden af en kant $|o_{ij} \rightarrow o_{kl}|$ repræsenterer som nævnt, at o_{kl} tidligst kan afvikles p_{ij} tidsenheder efter påbegyndelse af o_{ij}.

Dette betyder, at har man tre operationer o_{gh}, o_{ij} og o_{kl}, der er forbundet med kanterne $o_{gh} \rightarrow o_{ij}$ og $o_{ij} \rightarrow o_{kl}$, da vil der implicit eksistere en kant $o_{gh} \rightarrow o_{kl}$, der betyder, at o_{kl} tidligst kan afvikles $p_{gh}+p_{ij}$ tidsenheder efter påbegyndelse af o_{gh}. En sådan implicit kant kaldes en *delayed precedence constraint* (DPC). Dette er illustreret i figur 3.3.



Figur 3.3: DPC mellem o_{gh} og o_{kl}

Denne "omvendte" trekantsulighed kan generaliseres til, at det korteste tidsinterval mellem to operationer o_{ij}, o_{kl} er givet ved den længste vej mellem disse — hvis altså der findes en vej. Længden af en vej er defineret som

summen af de indgående kanters længde. Den længste vej mellem knuderne o_{ij} og o_{kl} betegnes $L(o_{ij}, o_{kl})$.

Der findes således en DPC mellem operationerne o_{ij} og o_{kl}, netop hvis der findes en anden vej $o_{ij} \rightsquigarrow o_{kl}$ mellem disse end en direkte kant $o_{ij} \rightarrow o_{kl}$. Længden af denne DPC er da $L(o_{ij}, o_{kl})$.

I en brugbar løsning kan en operations starttidspunkt udtrykkes som det kortest mulige interval mellem kildeknuden, der afvikles til tiden 0, og den aktuelle operation, dvs. $S_{ij} = L(\mathcal{S}, o_{ij})$.

På baggrund heraf samt repræsentationerne af objektfunktionerne nævnt i afsnit 3.5.1 kan vi også udtrykke objektfunktionsværdien vha. længder. Makespan er som nævnt afviklingstidspunkt for pseudo-operationen $\mathcal{V}$, og dette er givet ved den længste vej $L(\mathcal{S}, \mathcal{V})$. Ligeledes er TWT givet ved

$$\sum_i w_i(L(\mathcal{S}, \mathcal{V}_i) - d_i)$$

dvs. det vægtede sum af den periode, hvormed afviklingstidspunktet for pseudo-operationen $\mathcal{V}_i$ overstiger jobbets deadline d_i. Leddene i summen er alle ikke-negative, idet tidligste afviklingstidspunkt for $\mathcal{V}_i$ som nævnt i afsnit 3.5.1 er d_i pga. kanten $\mathcal{S} \rightarrow \mathcal{V}_i$.

3.5.4 Den disjunktive grafrepræsentation

I litteraturen benyttes i de fleste tilfælde en lidt anden model end den her nævnte. I den findes der to typer kanter, nemlig konjuktive og disjunktive.

De konjuktive er den type, der er omtalt i denne rapport. De disjunktive optræder i par af to i hver sin retning mellem to operationer, hhv. $o_{ij} \rightharpoonup o_{kl}$ og $o_{kl} \rightharpoonup o_{ij}$. Disse dobbeltkanter angiver, at der i en brugbar løsning skal tilføjes en konjuktiv kant mellem de to operationer.

I den initielle løsning er der således disjunktive kanter mellem hvert par af operationer på samme maskine. Herved sikres, at der ikke er tidsmæssigt overlap mellem operationer på samme maskine.

I denne fremstilling er dette krav i stedet formuleret således, at der skal findes en vej gennem alle operationer tilknyttet en given maskine. Herved understreges, at der ikke behøver at indsættes konjuktive kanter mellem

operationer, hvor der i forvejen eksisterer en DPC, og heller ikke nødvendigvis i situationer som den afbildet i figur 3.2. Det skønnes, at denne model er mere velegnet til illustration af metoden kronologisk konfliktløsning, som vil blive omtalt i afsnit 5.1.1.

Grafrepræsentationen er dog kun en abstraktion over problemet med det formål at illustrere visse begreber. Valg af model har således ikke indflydelse på selve løsningsmetoderne.

3.6 Generel løsningsmetode

Ved branch-and-bound-løsning — samt flere andre løsningsmetoder — tages som regel udgangspunkt i den initielle løsning, hvortil der udelukkende tilføjes kanter, indtil en brugbar løsning nås. Der fjernes således ikke kanter.

Da det kun er operationer på samme maskine, der kan konflikte, er der udover kanterne i den initielle løsning normalt ikke behov for kanter mellem operationer på forskellige maskiner. Denne slags kanter har ikke været benyttet i dette projekt. Når der i denne rapport omtales indsættelse af en kant mellem to operationer, gælder det derfor implicit, at der — med undtagelse af pseudo-operationerne $\mathcal{S}$, $\mathcal{V}$ og $\mathcal{V}_i$ — er tale om to operationer på samme maskine.

Der må ikke indsættes kanter, så der dannes en kreds i grafen. Dette vil nemlig føre til en modstrid, nemlig at en operation skal være afsluttet, før den kan påbegyndes.

Når der i denne rapport nævnes øvre og nedre grænser for forskellige variable, så menes der — medmindre andet eksplicit er nævnt — den øvre grænse for variablens værdi i den aktuelle løsning samt de løsninger, der måtte nås ved at tilføje yderligere kanter til den aktuelle løsning.

Da løsningsmetoderne som oftest arbejder med ikke-brugbare løsninger, vil betegnelsen "løsning" i denne rapport dække over såvel brugbare som ikke-brugbare løsninger, men oftest det sidste. Undertiden benyttes også betegnelserne "delløsning" og "ikke-brugbar løsning" for i en given sammenhæng at understrege, at der ikke er tale om brugbare løsninger.

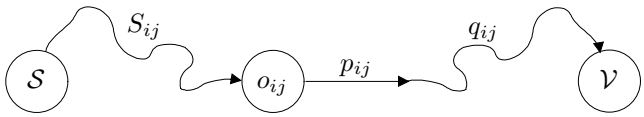
3.7 Hoved og hale

S_{ij} betegnes undertiden operationens *hoved*. Hovedet er en nedre grænse for den tid, der går forud for operationens behandling, og således også for starttidspunktet.

Symmetrisk udgør operationens *hale* — vagt formuleret — en nedre grænse for den tid, der følger operationens afslutning; den præcise definition afhænger af objektfunktionen. Halen kan bruges til at bestemme det seneste tidspunkt, operationen kan afsluttes, uden den aktuelle objektfunktionsværdi stiger, dvs. en øvre grænse for C_{ij}. Denne grænse kaldes operationens deadline d_{ij}. Analogt med deadlines for jobs skal denne deadline altså ikke opfattes som et ultimativt krav til en brugbar løsning.

Ved makespan er halen q_{ij} det korteste interval mellem operationens afslutning og afslutningen af den sidste operation i problemet. Værdien $S_{ij} + p_{ij} + q_{ij}$ udgør således en nedre grænse for problemets makespan. Dette er vist i figur 3.4. Operationens deadline er således

$$d_{ij} = O_{makespan} - q_{ij} \tag{3.9}$$

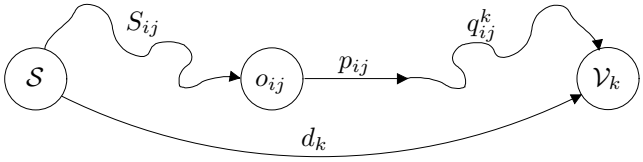


Figur 3.4: En operations hoved, behandlingstid og hale ved makespan.

Ved TWT defineres en hale for hvert job. Således er halen q_{ij}^k det korteste interval mellem operationens afslutning og afslutningen af J_k. Dette interval er kun defineret, hvis der i den aktuelle løsning findes en vej fra o_{ij} til den sidste operation på J_k og således også afløbsknuden $\mathcal{V}_k$. I modsat fald antager q_{ij}^k værdien $-\infty$.

En nedre grænse for et jobs afslutning C_k er derfor $S_{ij} + p_{ij} + q_{ij}^k$. Heraf følger, at en øvre grænse for operationens afslutning, uden at T_k påvirkes er

$$d_{ij}^k = \max(d_k, C_k) - q_{ij}^k \tag{3.10}$$



Figur 3.5: En operations hoved, behandlingstid og hale mht. jobbet J_k ved TWT.

Dette er illustreret i figur 3.5. Især bemærkes, at hvis ikke der findes en vej $o_{ij} \rightsquigarrow \mathcal{V}_k$, da vil værdien af d_{ij}^k være ∞, dvs. at det ikke vil påvirke afslutningstidspunktet for J_k, uanset hvor sent operationen udføres.

Hvis T_k stiger blot for ét job, vil objektfunktionsværdien ændres. En øvre grænse for C_{ij}, uden objektfunktionsværdien påvirkes, er således den mindste af de øvre grænser mht. hvert af de enkelte jobs:

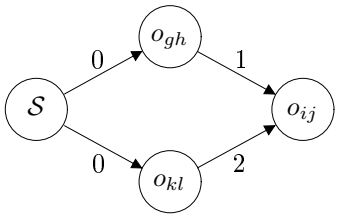
$$d_{ij} = \min_{J_k \in \mathcal{J}}(d_{ij}^k) = \min_{J_k \in \mathcal{J}}(\max(d_k, C_k) - q_{ij}^k) \tag{3.11}$$

3.7.1 Beregning af hoved og hale

Hoved og hale i en given delløsning er som sagt nedre grænser for tiden før hhv. efter operationens behandling i alle brugbare løsninger, der kan nås ved at tilføje kanter til den aktuelle løsning. De kan beregnes på forskellige måder, afhængigt af hvilke værdier der i forvejen beregnes i en given løsningsalgoritme, samt hvor meget tid man ønsker at afsætte til beregningen.

Én måde at beregne værdierne på er vha. længste veje i grafen. Udtrykt i længste veje er $S_{ij} = L(\mathcal{S}, o_{ij})$, den længste vej mellem kildeknude og o_{ij}. $L(o_{ij}, \mathcal{V})$ udgør det mindste interval mellem påbegyndelse af o_{ij} og afslutningen af den sidste operation, dvs. halen $q_{ij} = L(o_{ij}, \mathcal{V}) - p_{ij}$.

Dette er dog ikke nødvendigvis den tætteste grænse, der kan findes for en given delløsning. Figur 3.6 viser et eksempel, hvor man kan finde en bedre værdi af S_{ij} ved at betragte andet end de kanter, der allerede er indsat. Her tilhører o_{gh} og o_{kl} samme maskine M_p. Et estimat for S_{ij} vha. længste vej er $L(\mathcal{S}, o_{ij}) = 2$. Da o_{gh} og o_{kl} er tilknyttet samme maskine, kan de ikke udføres samtidig, dvs. at for at den aktuelle delløsning bliver brugbar, skal



Figur 3.6: $L(\mathcal{S}, o_{ij})$ er ikke den bedste værdi for S_{ij}.

der indsættes en kant enten $o_{gh} \rightarrow o_{kl}$ eller $o_{kl} \rightarrow o_{gh}$. Uanset hvilken vej, kanten vender, vil den seneste af de to operationer tidligst kunne afsluttes til tidspunktet 3, og først her vil o_{ij} kunne påbegyndes. En bedre værdi for S_{ij} i dette tilfælde er således 3.

3.8 Kompleksitet

JSS er stærkt $\mathcal{NP}$-hårdt udover helt simple tilfælde, hvor antallet af maskiner og jobs er 1 eller 2, og hvor der stilles særlige krav til længden af operationer og vægten af jobs [10]. Dette gælder med såvel makespan som TWT som objektfunktion.

Selv specialtilfældet bestående af kun én maskine er i mange tilfælde $\mathcal{NP}$-hårdt [14]. I afsnit 5.3 vil vi dog omtale en relakseret udgave af dette problem, der kan løses effektivt. Løsningen af det relakserede problem benyttes som nedre grænse for objektfunktionsværdien af det oprindelige problem.

3.9 Matematisk formulering

Her følger en formulering af JSS som et mixed integer problem med henblik på løsning vha. CPLEX eller tilsvarende.

Vi ønsker at bestemme rækkefølgen af operationerne på hver enkelt maskine. Derfor introduceres for hver par af operationer $o_{ij}, o_{kl}, i < k$, på samme maskine M_p variablen $x^p_{ik} \in \{0, 1\}$, der har samme betydning som en kant i grafen mellem o_{ij} og o_{kl}. Betingelsen $i < k$ sikrer, at der kun benyttes én variabel pr. par af operationer.

$\forall\, M_p \in \mathcal{M}, o_{ij}, o_{kl} \in M_p, i < k$:

$$x_{ik}^p = \begin{cases} 0 & \text{hvis } o_{ij} \text{ afvikles før } o_{kl} \\ 1 & \text{hvis } o_{kl} \text{ afvikles før } o_{ij} \end{cases} \tag{3.12}$$

$x_{ik}^p = 0$ svarer således til kanten $o_{ij} \rightarrow o_{kl}$, og $x_{ik}^p = 1$ svarer til den modsatte retning, $o_{kl} \rightarrow o_{ij}$.

Bemærk at det ikke er alle potentielle kanter i grafen, der repræsenteres, idet vi her kun betragter operationer på samme maskine, hvilket er tilstrækkeligt.

Beslutningsvariablene er tidspunktet for afvikling af hver enkelt operation S_{ij}, eller ækvivalent ordningsvariablerne x_{ik}^p.

3.9.1 Betingelser

Tidligste tidspunkt for påbegyndelse af et job samt rækkefølgen af operationerne på jobbet repræsenteres ved følgende betingelser:

$$\begin{array}{lcll} S_{i1} & \geq & r_i & \forall\, J_i \in \mathcal{J} \\ S_{ij} & \geq & S_{i(j-1)} + p_{i(j-1)} & \forall\, J_i, 2 \leq j \leq m \end{array} \tag{3.13}$$

Her repræsenteres altså kanterne i grafen mellem operationer på samme job. r_i og p_{ij} er konstanter, der følger af den aktuelle problemdefinition.

En operation ikke kan påbegyndes, før dens forgænger i henhold til x_{ik}^p er afsluttet. Det betyder, at afhængigt af værdien af x_{ik}^p skal netop én af følgende betingelser gælde:

$\forall\, M_p \in \mathcal{M}, o_{ij}, o_{kl} \in M_p, i < k$:

$$\begin{array}{lcll} S_{kl} & \geq & S_{ij} + p_{ij} & \text{hvis } x_{ik}^p = 0 \\ S_{ij} & \geq & S_{kl} + p_{kl} & \text{hvis } x_{ik}^p = 1 \end{array} \tag{3.14}$$

Da CPLEX ikke kan håndtere denne slags enten-eller-begrænsninger, lineariseres de vha. følgende omskrivning:

$$\begin{array}{lcl} S_{kl} & \geq & S_{ij} + p_{ij} - h\, x_{ik}^p \\ S_{ij} & \geq & S_{kl} + p_{kl} - h\, (1 - x_{ik}^p) \end{array} \tag{3.15}$$

hvor h er en stor konstant. Hvis $x_{ik}^p = 0$, vil sidste led i nederste ulighed blive meget numerisk stort, og uligheden vil således altid være sand. Tilsvarende gælder for øverste ulighed for $x_{ik}^p = 1$.

h kan eksempelvis være

$$h = \max_i(r_i) + \sum_{i,j} p_{ij}$$

hvilket svarer til den maksimale værdi af $S_{kl}+p_{kl}$ i en løsning, hvor jobbene udføres ét ad gangen. I en sådan løsning vil der aldrig vil kunne opstå en konflikt, så løsningen vil altid være brugbar. Pr. definitionen på regulære objektfunktioner vil det ikke føre til en bedre løsning at lade noget job afslutte senere, og som nævnt i afsnit 3.5 afvikles alle operationer så tidligt som muligt. Derfor er $S_{kl} + p_{kl} \leq h$ i alle optimale løsninger.

3.9.2 Objektfunktioner

Ved beregning af makespan introduceres variablen $O_{makespan}$, der udgør det seneste afslutningstidspunkt for et job:

$$O_{makespan} \geq C_i = S_{im} + p_{im} \quad \text{for alle } J_i \in \mathcal{J}$$

Ved beregning af TWT introducerer vi variablene T_i, der angiver tardiness for hvert job:

$$O_{TWT} = \sum_i w_i T_i$$

Vægtene w_i er konstanter. T_i er begrænset af flg. to størrelser:

$$\begin{array}{rcll} T_i & \geq & 0 & \\ T_i & \geq & C_i - d_i = S_{im} + p_{im} - d_i & \text{for alle } J_i \in \mathcal{J} \end{array} \tag{3.16}$$

Da O_{TWT} skal minimeres og $w_i > 0$, da bliver T_i netop den største af de to højresider og angiver således tardiness iflg. definitionen (3.5).

Afhængigt af den benyttede objektfunktion består problemet nu i at minimere $O_{makespan}$ eller O_{TWT} under hensyntagen til betingelserne nævnt i afsnit 3.9.1.

Kapitel 4

JSS-formulering af STRS

STRS er i litteraturen søgt løst vha. en række specialiserede algoritmer. Vi vil her formulere STRS som et JSS-problem, hvilket tillader os at løse det vha. den lange række af metoder, der findes til løsning af JSS.

I denne formulering svarer en maskine til et sporsegment, og et job svarer til et tog. I denne forbindelse defineres et tog som én tur mellem rutens endestationer, dvs. at det ikke tages i betragtning, at det samme materiel senere benyttes til kørsel af en anden tur. Et togs passage af et givet sporsegment udgør en operation. Operationens behandlingstid p_{ij} er den tid, toget tager om at passere sporsegmentet.

Betingelsen (3.4), om at to operationer på samme maskine ikke kan overlappe, sikrer, at to tog ikke kan kollidere, idet de herved ikke tillades at befinde sig på et givet sporsegment på samme tid.

Ifølge (3.1) kan en operation først påbegyndes, når dens forgænger på samme job er afsluttet. Denne betingelse sikrer dels, at sporsegmenterne passeres i den rigtige rækkefølge, idet det i sagens natur ikke vil give mening at fortsætte fra et givet sporsegment til andet end det følgende segment på strækningen[1]. Derudover sikres, at køreplanen respekterer den tid, et tog

[1] Principielt kunne toget også køre tilbage til det foregående segment. Dette vil dog aldrig kunne betale sig sammenlignet med blot at vente med at køre frem.

er om at passere et sporsegment. Først når hele segmentet er tilbagelagt, kan toget fortsætte til følgende segment.

I afsnit 3.2 er nævnt, at det er relativt uproblematisk at fravige betingelserne i simpel JSS om, at hvert job skal bestå af lige mange operationer, og at hvert job skal behandles på hver maskine netop én gang. Fjernes disse betingelser, vil det i forbindelse med togskedulering betyde, at man kan repræsentere et forgrenet jernbanenet i stedet for blot én lang strækning, samt at togene ikke behøver at følge den samme rute. Som nævnt vil denne rapport dog kun betragte den specialiserede problemstilling.

En oplagt objektfunktion for togskedulering er TWT, idet man ønsker at minimere forsinkelserne for alle tog, modsat for eksempel makespan, hvor man ønsker at minimere ankomsttidspunktet for det sidste tog uden hensyntagen til de øvriges ankomsttidspunkt.

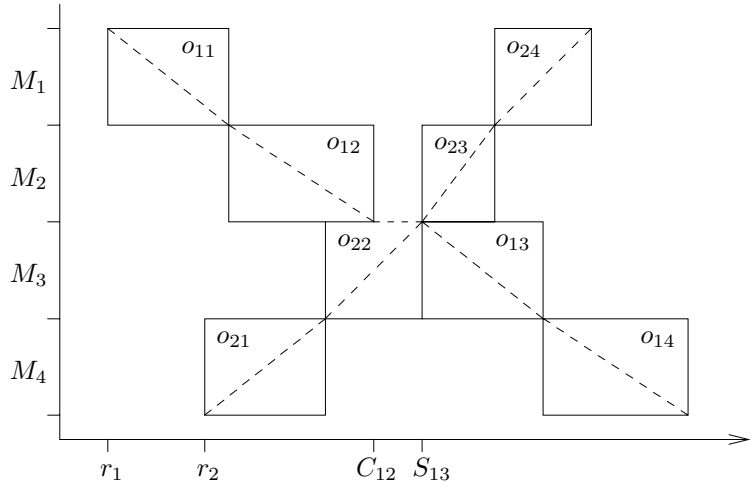
Togene kan tildeles vægt efter forskellige kriterier. Eksempelvis kan passagertog tildeles større vægt end godstog, idet gods som regel kan tolerere mere forsinkelse end passsagerer. Ligeledes kan tog med mange passagerer eller meget gods tildeles højere vægt end tog med få passagerer eller lidt gods.

Grundet naturen af jernbanedrift vil vægtningen i praksis formodentlig ofte være et politisk valg snarere end et direkte resultat af en matemetisk optimering. Kriterier for vægtning af tog vil således ikke blive omtalt yderligere i denne rapport. I [21] er nævnt et simpelt kriterium for generel JSS, som vil blive omtalt nærmere i afsnit 6.3.

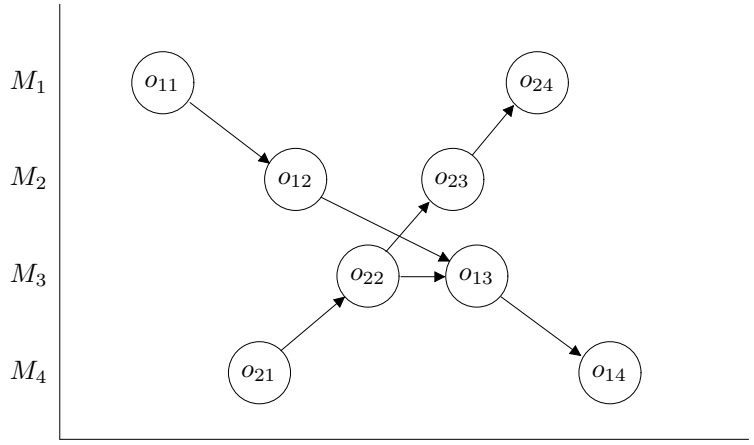
4.1 Gantt-diagram

Figur 4.1 viser et Gantt-diagram over en brugbar løsning af et problem med to tog og fire sporsegmenter. Hvert rektangel svarer til en operation og hver vandret række til et sporsegment. Tiden er afsat ad abscisse-aksen, og rektanglernes venstre og højre kant angiver således operationernes start- og afslutningstidspunkter, S_{ij} og C_{ij}. Heraf følger, at operationens behandlingstid p_{ij} er repræsenteret ved rektanglernes bredde.

Ordinat-aksen svarer til den samlede jernbanestrækning længde. Mærkerne i aksen angiver opdelingen i segmenter, og stykket mellem to mærker udgør et sporsegment.



Figur 4.1: Gantt-diagram over løsning af problem med to tog og fire sporsegmenter.



Figur 4.2: Grafrepræsentation af løsningen vist i figur 4.1.

De stiplede linjer svarer til togets fysiske rute, og togets fysiske placering til et givet tidspunkt kan således aflæses direkte[2]. Operationerne er dog stadig angivet som rektangler, idet vi i vores formulering af STRS kun tillader, at der befinder sig ét tog på et givet segment ad gangen.

Det ses, at linjen for J_1 er vandret mellem operationerne o_{12} og o_{13}, dvs. i perioden fra C_{12} til S_{13}. Det betyder, at toget holder og venter ved grænsen mellem segmenterne M_2 og M_3 på at kunne køre ind på M_3, idet dette er optaget af o_{22}. Denne venten svarer til en kant $o_{22} \rightarrow o_{13}$ i grafen. Grafrepræsentationen af løsningen er vist i figur 4.2. Knuderne er her indtegnet, så de har samme centrum som de tilsvarende rektangler i figur 4.1.

Bemærk, at ethvert JSS-problem kan illustreres vha. et Gantt-diagram. Mens ordinat-aksen ved STRS er kontinuert, er den i det generelle tilfælde diskret, og de stiplede linjer vil ikke benyttes. Ligeledes tillægges normalt ingen betydning til maskinernes rækkefølge, dvs. at maskinerne i det generelle tilfælde vil kunne indtegnes i tilfældig orden. Dette er ikke tilfældet ved STRS, hvor to maskiner M_i og M_{i+1} svarer til segmenter, der støder op til hinanden.

4.2 Begrænsninger i modellen

Denne model af STRS forudsætter, at der er et vigespor mellem hvert sporsegment, dvs. at et langt sporsegment ikke uden videre vil kunne opdeles i flere blot ved opsættelse af flere stopsignaler. Dette er nødvendigt for, at to tog kan passere hinanden. I ovennævnte eksempel passerer togene hinanden, hvor de stiplede linjer krydser hinanden, dvs. til tidspunktet S_{13} på grænsen mellem M_2 og M_3. Her er det altså nødvendigt med et vigespor.

Ligeledes antages, at alle vigespor har kapacitet til et vilkårligt antal tog, og at togene kan forlade vigesporet uafhængigt af den rækkefølge, hvori de er ankommet. Desuden regnes ikke med tid til rangering mv. i forbindelse med passage; ej heller regnes med tid til standsning ved stationer i forbindelse med passagerers af- og påstigning eller med af- og pålæsning af gods. Dette antyder, at modellen er mest relevant ved jernbaner, hvor strækningerne

[2]De rette linjer antyder, at toget bevæger sig med konstant hastighed. Dette er dog ikke en forudsætning, idet vi alene betragter linjens endepunkter.

er store og tidsforbruget i forbindelse med rangering mv. er lille sammenlignet med selve transporttiden. En del af litteraturen beskæftiger sig med toglinjer på tværs af Australien, og i tilfælde som dette må denne antagelse siges at være realistisk.

Som nævnt i afsnit 2.1.2 omtales i [17] et mere generelt problem, end de rapporterede testkørsler formår af afdække. Det problem, der rent faktisk kommer til udtryk i de rapporterede kørsler, ligner dog meget det, der betragtes i denne rapport. Sammenligning af de opnåede løsningsværdier viser dog, at der er forskel. Præcis hvori denne forskel ligger, er uklart. Dette vil omtalt nærmere i afsnit 6.10.

4.3 Særlige egenskaber ved STRS

Med baggrund i STRS' fysiske fortolkning, har problemet sammenlignet med generel JSS nogle særlige egenskaber, som måske kan udnyttes ved løsning.

Alle tog bevæger sig mellem jernbanestrækningens to endestationer, enten i den ene eller den anden retning, dvs. at togets retning kan repræsenteres vha. en binær konstant. Hvert job benytter således maskinerne enten i rækkefølgen $M_1, M_2, \ldots, M_m$ eller $M_m, M_{m-1}, \ldots, M_1$. Heraf følger blandt andet, at hvis to tog kører i hver sin retning, da skal togene nødvendigvis passere hinanden på et eller andet tidspunkt. Hvis de derimod kører samme vej, svarer en passage til en overhaling, hvilket er en mulighed men ikke en nødvendighed.

Specialtilfældet, hvor alle tog kører i samme retning, kaldes *flow-shop-skedulering*. Der findes en del litteratur om denne type problemer, og generelt vil de formentlig kunne løses hurtigere vha. specialiserede algoritmer end ved generelle JSS-metoder. STRS består i flow-shop-varianten ikke af at bestemme, hvor modkørende tog skal passere hinanden, men i stedet hvor tog skal overhale hinanden.

I praksis vil man formodentlig ofte opleve, at den tid, det tager et givet tog at passere et sporsegment, dvs. behandlingstiden p_{ij}, er nogenlunde proportional med segmentets længde. Hvis dette er konsekvent, vil man kunne repræsentere et togs hastighed som en konstant. Et eksempel er vist i figur 4.1, hvor $p_{1j} > p_{2l}$ for alle o_{1j}, o_{2l} på samme maskine. Her vil man

generelt kunne sige, at J_2 er hurtigere end J_1. På baggrund af dette vil man muligvis kunne opstille nogle regler, eksempelvis for hvornår hurtige tog skal have lov til at overhale langsomme.

Disse egenskaber har været overvejet i forbindelse med dette projekt, uden det dog har ført til nogen anvendelige resultater. De anvendte metoder til løsning af STRS kan således uden videre anvendes til løsning af generelle JSS-problemer.

Kapitel 5

Løsningsmetoder

I dette kapitel gennemgås forskellige metoder til løsning af JSS. Først omtales løsning til optimalitet vha. branch-and-bound, og dernæst omtales en heuristik.

I forbindelse med branch-and-bound gennemgås i afsnit 5.1 to branching-strategier: kronologisk konfliktløsning og kantindsættelse. I afsnit 5.2, introduceres et dominanskriterium, der kan bruges til at beskære søgetræet. Dernæst omtales i afsnit 5.3 funktioner til beregning af bounds for objektfunktionsværdien, herunder en ny metode for TWT.

I afsnit 5.4 gennemgås heuristikken shifting bottleneck, og slutteligt nævnes i afsnit 5.5 en måde at foretage lokalsøgning på.

5.1 Branching

Ved løsning af JSS vha. branch-and-bound tager man som regel udgangspunkt i den initielle løsning, hvortil der tilføjes kanter, indtil løsningen er brugbar.

Hver gang der indsættes en kant, bestemmes rækkefølgen af de to involverede operationer. Disse to operationer kan udvælges på mange måder, hvilket giver anledning til en række forskellige branching-strategier. I det følgende vil to forskellige blive gennemgået: kronologisk konfliktløsning samt kantindsættelse.

5.1.1 Kronologisk konfliktløsning

Hvis en ikke-brugbar løsning angiver, at to eller flere operationer skal udføres på samme maskine samtidig, tales der som nævnt i afsnit 3.2 om en konflikt.

Konflikter kan løses ved at bestemme en rækkefølge af de involverede operationer. Herved udskydes alle konflikter på nær den første i denne rækkefølge.

Løsning af en konflikt har derved kun konsekvenser for operationer, der forekommer senere i den midlertidige løsning. Ved iterativt at løse den tidligste konflikt ordnet efter starttidspunktet for den første involverede konflikt vil man således "fastfryse" planen fremad, indtil alle konflikter er løst, dvs. at løsningen er brugbar.

Denne procedure er beskrevet i algoritme 5.1. Bemærk at ikke alle dele af den benyttede algoritme er vist her men blot antydet vha. tre prikker. Øvrige relevante dele vil blive introduceret senere i dette kapitel.

GetEarliestConflict returnerer den tidligste konflikt mht. det tidligste starttidspunkt for de involverede operationer. Der returneres et par af to operationer, eller nil hvis ikke der er flere konflikter. Selvom en konflikt omfatter mere end to operationer, er det tilstrækkeligt blot at udvælge to ad gangen, idet alle rækkefølger af de involverede operationer alligevel vil blive undersøgt i løbet af nogle iterationer af branch-and-bound-proceduren.

Er der ikke flere konflikter (linje 2), er løsningen brugbar. Hvis den i givet fald er bedre end den hidtil bedste E_{best}, gemmer vi den og den deraf følgende øvre grænse for objektfunktionsværdien UB (linje 3–6). Under alle omstændigheder returneres den aktuelle løsning for at indikere, at dette er den bedste løsning fundet i den aktuelle branch E.

Hvis der derimod findes en konflikt (linje 8), brancher vi på rækkefølgen af de involverede operationer o_{ij}, o_{kl}. Dette sker ved at opdele den aktuelle løsning i to branches, der består af den aktuelle løsning tilføjet én af kanterne $o_{ij} \rightarrow o_{kl}$ eller $o_{kl} \rightarrow o_{ij}$ (linje 11), hvilket angiver, at konflikten løses ved at lave o_{ij} afvikle før hhv. efter o_{kl}.

Det undersøges, om den nedre grænse for objektfunktionsværdien efter indsættelse af kanten stadig er mindre end den hidtil bedste løsningsværdi (linje 15; LowerBound vil blive introduceret i afsnit 5.3). Er dette tilfældet, kaldes BranchAndBound rekursivt — alternativt forkastes den aktuelle delløsning E'.

```
BranchAndBound(E)
 1  c ← GetEarliestConflict(E)
 2  if c = nil then
 3      if O(E) < UB then
 4          E_best ← E
 5          UB ← O(E_best)
 6      end if
 7      return E
 8  else
 9      ...
10      (o_ij, o_kl) ← c
11      B ← {E ∪ o_ij → o_kl, E ∪ o_kl → o_ij}
12      E_min ← nil
13      for each E' ∈ B do
14          ...
15          if LowerBound(E') < UB then
16              E_b ← BranchAndBound(E')
17              if O(E_b) < O(E_min) then
18                  E_min ← E_b
19              end if
20          end if
21      loop
22      ...
23      return E_min
24  end if
```

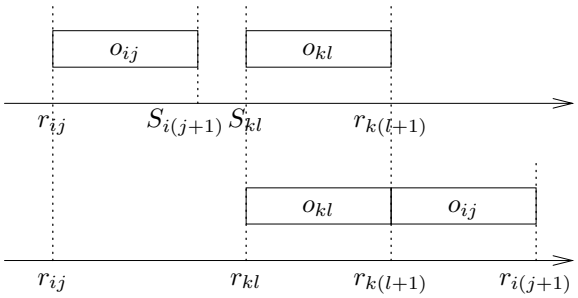
Algoritme 5.1: Branch-and-bound vha. kronologisk konfliktløsning.

Til sidst returneres den aktuelle bedste løsning af den aktuelle branch E_{min}, hvilket vil blive benyttet i forbindelse med en dominanskriteriet, der omtales i afsnit 5.2. I denne forbindelse defineres $O(\text{NIL}) \equiv \infty$ (benyttes ved første eksekvering af linje 17).

Korrekthed

Denne løsningsmetode vil altid finde den optimale løsning. Dette er måske ikke umiddelbart indlysende, så her følger et argument.

Betragt to på hinanden følgende operationer o_{ij}, o_{kl} på samme maskine til det tidspunkt, hvor alle tidligere konflikter er løst. Hvis operationerne overlapper hinanden, da brancher vi og undersøger både situationen, hvor o_{ij} kommer før o_{kl}, og vice versa. Hvis operationerne derimod ikke overlapper, da er rækkefølgen umiddelbart givet. Lad os sige, at o_{ij} kommer før o_{kl}. Spørgsmålet er nu, om det aldrig vil give en bedre løsning at lade o_{ij} komme efter o_{kl}.



Figur 5.1: De to mulige rækkefølger af o_{ij} og o_{kl}

Svaret er nej. På figur 5.1 vises de to mulige rækkefølger. r_{ij} og r_{kl} angiver, hvornår operationerne tidligst kan begynde under hensyntagen til tidligere planlagte operationer. $r_{i(j+1)}$ og $r_{k(l+1)}$ angiver, hvornår de efterfølgende operationer på de respektive jobs kan påbegyndes. Det ses, at værdien af $r_{k(l+1)}$ er den samme i de to situationer, mens $r_{i(j+1)}$ er større i den situation, hvor o_{ij} kommer efter o_{kl}. Vælger man at lade o_{ij} afvikle først, da vil man kunne opnå samme situation som hvis o_{kl} blev afvikles først, nemlig ved at lade o_{ij} afvikle og derefter vente til tidspunktet $r_{k(l+1)} + p_{ij}$

med at afvikle $o_{i(j+1)}$. I den beskrevne situation vil det således aldrig give en bedre løsning at lade o_{kl} afvikle først.

Bemærk, at o_{ij} ikke konflikter med andre operationer, og således ikke vil drage fordel af at blive afviklet senere end ellers for at undgå en evt. konflikt. Dette ville nemlig være i modstrid med, at vi løser konflikter i kronologisk rækkefølge, og at vi p.t. betragter operationerne o_{ij} og o_{kl}.

Dispatch-regler

Dispatch-regler er heuristikker, der ved at betragte en given konflikt helt lokalt løser konflikten ud fra en simpel regel. Disse kan dels bruges i forbindelse med ren heuristisk løsning, men de kan også anvendes til at guide en dybde-først-branch-and-bound-søgning.

I algoritme 5.1 linje 13 bruges en dispatch-regel til at bestemme den rækkefølge, hvori de to branches undersøges. Hvis man vælger den branch med den bedste løsning først, og denne giver anledning til en ny øvre grænse, da vil den anden branch hurtigere kunne blive beskåret.

Dispatch-regler benyttes også i lokalsøgningen omtalt i afsnit 5.5. Nabolaget her omfatter nemlig ikke-brugbare løsninger, der typisk kun indeholder få konflikter. Dispatch-reglerne bruges her som en hurtig metode til at danne en brugbar løsning på baggrund af disse løsninger.

Her følger en beskrivelse af et udvalg af dispatch-regler til løsning af konflikter mellem to operationer, o_{ij} og o_{kl}. Reglerne angiver et prioritetsindeks for den ene operation o_{ij}; indekset for den anden operation er givet symmetrisk. Den operation med laveste prioritetsindeks vælges til at blive afviklet først.

Reglerne omtales med forskellige betegnelser rundt omkring i litteraturen. Navngivningen her er derfor ikke udtryk for nogen særlig tradition på området.

Mindste ventetid (MV): $C_{ij} - S_{kl}$

Der vælges den løsning, der fører til korteste ventetid ved denne maskine, dvs. tid hvor det ene job venter på det andet[1]. Hvis o_{ij} afvikles

[1] Denne regel er anvendt i [17], hvor den dog betegnes "shortest processing time".

først, vil den afsluttes til tiden C_{ij}, hvorefter o_{kl} så kan påbegyndes. o_{kl} har været klar siden S_{kl}, dvs. ventetiden bliver $C_{ij} - S_{kl}$.

Mindste vægtede ventetid (MVV): $w_i\,(C_{ij} - S_{kl})$
Vægtet udgave af MVV.

Mindste makespan (MM): $S_{ij} + p_{ij} + p_{kl} + q_{kl}$
Operationen der resulterer i den mindste nedre grænse for makespan baseret på det aktuelle hoved S_{ij} og hale q_{kj} (denne grænse forklares nærmere i afsnit 5.1.2).

Først-til-mølle (FTM): S_{ij}
Operationen, der først er klar til behandling på den givne maskine, afvikles først.

Mindste behandlingstid (MB): p_{ij}
Operationen med den korteste behandlingstid afvikles først.

5.1.2 Kantindsættelse

Ved at betragte to operationer på samme maskine i en given delløsning kan man ved hjælp af en simpel beregning bestemme en nedre grænse for objektfunktionsværdien, hvis der blev indsat en kant mellem dem i hhv. den ene eller den anden retning. Hvis netop en kant i netop én af de to retninger medfører en nedre grænse, der er større eller lig den hidtil bedste objektfunktionsværdi, da kan kanten i den modsatte retning indsættes.

Resulterer indsættelse af såvel den ene som den anden kant i en større objektfunktionsværdi, kan den aktuelle løsning forkastes, idet den vil have en ringere løsning, uanset hvilken rækkefølge de to operationer afvikles i.

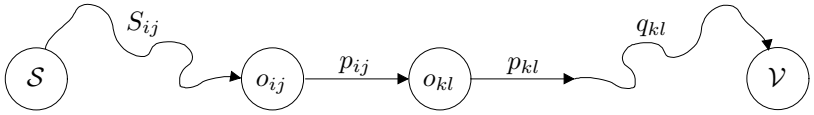
Den nedre grænse LB_{ij} for makespan i en løsning, hvor der indsættes en kant $o_{ij} \rightarrow o_{kl}$ er

$$LB_{ij} \quad = \quad S_{ij} + p_{ij} + p_{kl} + q_{kl}$$

Dette er illustreret i figur 5.2. Heraf fremgår, at $O_{makespan} \geq LB_{ij}$, idet makespan som nævnt i afsnit 3.5.3 udgør længden af den længste vej mellem kilde- og afløbsknuderne, $L(\mathcal{S}, \mathcal{V})$. LB_{kl} er defineret symmetrisk.

Ved TWT benyttes $S_{ij} + p_{ij} + p_{kl} + q^g_{kl}$ som nedre grænse for afslutningstidspunktet C_g for jobbet J_g efter indsættelse af kanten $o_{ij} \rightarrow o_{kl}$. Dette fører til følgende nedre grænse for objektfunktionsværdien:

$$LB_{ij} = \sum_{J_g \in \mathcal{J}} w_g (S_{ij} + p_{ij} + p_{kl} + q^g_{kl} - d_g)^+$$



Figur 5.2: Illustration af LB_{ij} ved makespan.

Lad nu UB betegne objektfunktionsværdien af den hidtil bedste løsning. Hvis både $LB_{ij} \geq UB$ og $LB_{kl} \geq UB$, da vil den aktuelle delløsning ikke kunne resultere i en bedre løsning end den hidtil bedste, og søgetræet kan beskæres.

Hvis $LB_{ij} < UB$ og $LB_{kl} \geq UB$, da vil en løsning indeholdende kanten $o_{kl} \rightarrow o_{ij}$ ikke resultere i en bedre løsning end den hidtil bedste. Vi kan således nøjes med at betragte den del af søgetræet under den aktuelle delløsning, hvor o_{ij} kommer før o_{kl}, dvs. vi kan indsætte kanten $o_{ij} \rightarrow o_{kl}$.

For hver maskine betragtes alle par af operationer, hvis rækkefølge ikke allerede er afgjort, dvs. hvor der ikke findes en vej $o_{ij} \rightsquigarrow o_{kl}$ eller $o_{kl} \rightsquigarrow o_{ij}$. Hvor det er muligt, indsættes en kant jvf. ovenstående. Denne procedure gentages, så længe der kan indsættes nye kanter.

Metoden kan anvendes ved kronologisk konfliktløsning, umiddelbart efter der er indsat en kant (linje 14 i algoritme 5.1). Dels løses potentielle konflikter, før de nås kronologisk, og dels indsættes ekstra kanter, hvilket kan udnyttes til at beregne en bedre nedre grænse (grænseberegning er omtalt i afsnit 5.3). Begge dele fører til en beskæring af søgetræet. Ligeledes kan den benyttes i forbindelse med shifting bottleneck, som er omtalt i afsnit 5.4.

Metoden kan dog også indtage en mere aktiv rolle i forbindelse med branching, hvilket vil blive omtalt i næste afsnit.

5.1.3 Branching ved kantindsættelse

I det følgende beskrives en anden branching-strategi end kronologisk konfliktløsning. Metoden, der er beskrevet i [4], tager udgangspunkt i kantindsættelse som omtalt i sidste afsnit.

Ligesom ved kronologisk konfliktløsning branches der her ved at tilføje en kant i hhv. den ene og den anden retning mellem to operationer. Mens

det ved kronologisk konfliktløsning er operationerne involveret i den første konflikt, er det her det par af operationer, der vil resultere i største stigning i den nedre grænse for objektfunktionsværdien.

Metoden tager som nævnt udgangspunkt i kantindsættelse. Alle par af operationer på hver maskine undersøges, og der indsættes om muligt kanter imellem dem. Hver gang der mødes et par af operationer, hvor såvel LB_{ij} som LB_{kl} er mindre end UB, da registreres de to operationer som kandidater til branching, dvs. at den aktuelle løsning kan blive opdelt i to branches, hvor der er indsat en kant hhv. $o_{ij} \rightarrow o_{kl}$ og $o_{kl} \rightarrow o_{ij}$.

Alle kandiderende par o_{ij}, o_{kl} i den aktuelle delløsning skal ordnes før eller siden, inden en brugbar løsning nås, og derfor udgør den største værdi af $\min(LB_{ij}, LB_{kl})$ en nedre grænse for objektfunktionsværdien den aktuelle delløsningen. Vi vælger derfor at branche på det par med den største værdi heraf. I tilfælde af lighed vælges det med størst værdi af $\max(LB_{ij}, LB_{kl})$.

Algoritme

Proceduren er beskrevet i algoritme 5.2. Der benyttes en åbenliste af delløsninger R, der til at starte med kun indeholder den løsning, som operationen kaldes med, typisk den initielle løsning $\emptyset$ (linje 1).

Så længe der er flere løsninger i åbenlisten, udvælger vi den med den laveste nedre grænse (linje 3). Hvis denne løsning har en højere nedre grænse end den hidtil bedste løsning (repræsenteret ved den øvre grænse UB), vil det samme gælde for alle resterende åbne løsninger, og der er dermed ikke håb om at finde en bedre løsning, hvorfor algoritmen terminerer (linje 6).

I praksis vil det altid være her, algoritmen terminerer, og ikke fordi R er tom (linje 2) — udover i det teoretiske tilfælde, hvor LowerBound-funktionen er så ringe, at den ikke for nogen betragtet delløsning har angivet en nedre grænse, der er større end løsningsværdien af den bedste løsning.

Vi betragter nu hvert par af operationer o_{ij}, o_{kl} på samme maskine (linje 13–28), hvis rækkefølge ikke allerede er bestemt (linje 14). Vi beregner den nedre grænse for objektfunktionsværdien af de to potentielle branches, der består af den aktuelle løsning tilført en orienteret kant mellem o_{ij} og o_{kl} (linje 16–17).

Denne beregning sker ved rent faktisk at tilføje kanten og så benytte boundfunktionen omtalt i afsnit 5.3. Dette giver en potentielt bedre grænse på

```
BranchAndBound(E)
 1  R ← {E}
 2  while R ≠ ∅ do
 3      E' ← argmin_{Ê∈R}(LowerBound(Ê))
 4      R ← R \ {E'}
 5      if LowerBound(E') ≥ UB then
 6          return
 7      end if
 8      E_last ← NIL
 9      while E' ≠ E_last do
10          E_last ← E
11          C ← ∅
12          b ← TRUE
13          for each M_p ∈ M, o_ij, o_kl ∈ M_p, o_ij ≠ o_kl do
14              if ¬(o_ij ⇝ o_kl ∈ E' ∨ o_ij ⇝ o_kl ∈ E') then
15                  b ← FALSE
16                  LB_ij ← LowerBound(E' ∪ {o_ij → o_kl})
17                  LB_kl ← LowerBound(E' ∪ {o_kl → o_ij})
18                  if LB_ij ≥ UB ∧ LB_ij ≥ UB then
19                      goto 2
20                  else if LB_ij ≥ UB then
21                      E' ← E' ∪ o_kl → o_ij
22                  else if LB_kl ≥ UB then
23                      E' ← E' ∪ o_ij → o_kl
24                  else
25                      C ← C ∪ {(o_ij, o_kl)}
26                  end if
27              end if
28          loop
29          if b then
30              ...
31          end if
32      loop
33      (o_ij, o_kl) ← SelectCandidate(B)
34      R ← R ∪ {E' ∪ {o_ij → o_kl}, E' ∪ {o_kl → o_ij}}
35  loop
```

Algoritme 5.2: Branch-and-bound ved kantindsættelse.

bekostning af en større beregningstid, men principielt er det det samme der sker som ved beregning af den simple nedre grænse ved kantindsættelse. Ligeledes indsættes kanter ved samme kriterier som ved kantindsættelse (linje 20–23). Hvis begge kanter resulterer i en lavere objektfunktionsværdi, gemmes kanterne som nævnt som kandidater til branching (linje 25).

Når der findes en vej mellem alle par af operationer på samme maskine (b = TRUE i linje 29), da er løsningen brugbar. Det undersøges om den er bedre end den hidtil bedste og gemmes i givet fald (de tre prikker i linje 30 — svarer til linje 3–6 i algoritme 5.1).

Endelig udvælges de par af kanter, der skal branches på vha. ovennævnte kriterium (linje 33), og de to nye delløsninger tilføjes åbenlisten (linje 34).

5.2 Dominans

Branch-and-bound-algoritmen bevæger sig rundt i søgetræet, så hver delløsning besøges højst én gang. Dog sker det, at to delløsninger fra forskellige knuder i søgetræet ligner hinanden i en sådan grad, at viden om den ene kan udnyttes, når den anden nås.

Denne egenskab kaldes dominans [15]. I dette afsnit omtales et nyt såkaldt dominanskriterium, dvs. en måde at udnytte disse ligheder på.

Når vi løser kronologisk, bestemmer vi som før nævnt operationernes starttidspunkt i kronologisk orden. I en given knude i søgetræet kan vi således opdele den aktuelle løsning i tre dele: en fortid, en nutid og en fremtid[2].

"-tid" refererer her til forløbet af løsningsprocessen mellem branch-and-bound-søgetræets rod og et blad, og således ikke til tidspunkter i selve løsningen. Ved kronologisk konfliktløsning er der dog en vis proportionalitet de to begreber imellem, forstået på den måde, at tidspunktet for den tidligste konflikt stiger på vejen fra rod til blad i søgetræet.

Princippet er, at hvis to branches har en identisk nutid, da vil deres fremtid også være ens. Operationer i fortiden bidrager til objektfunktionsværdien og således også til bounding af den aktuelle løsning, men deres indbyrdes

[2]Disse betegnelser opfundet til lejligheden og således ikke udtryk for nogen almen terminologi på området.

rækkefølge har ikke indflydelse på den videre løsning. Derfor vil den optimale løsning af hver af de to branches være identisk mht. operationerne i fremtiden.

I det følgende vil vi opstille en funktion Q for fortidens bidrag til objektfunktionsværdien. Givet to delløsninger med samme nutid, E_1, E_2. Hvis $Q(E_1) > Q(E_2)$, da vil samme relation gælde for den samlede objektfunktionsværdi, dvs. $O(E_1) > O(E_2)$. Man siger, at delløsningen E_1 domineres af E_2, og E_1 kan således beskæres fra søgetræet.

Dette vil blive præciseret i det følgende. Først definerer vi nutiden, og derefter opstiller vi funktionen Q.

5.2.1 Definition af nutid

Antag at vi i en knude i søgetræet netop har fundet den tidligste konflikt (o_{ij}, o_{kl}). Lad $t = \min(S_{ij}, S_{kl})$. Nutiden $\mathcal{N} = (P, D, F)$ defineres nu som følger:

- P : mængden af operationer, der er afsluttet til tiden t, dvs. $C_{ij} \leq t$ [3]
- D : $C_{ij} - t$ for operationer, der er under afvikling til tiden t, dvs. for hvilke det gælder, at $S_{ij} < t < C_{ij}$
- F : kanterne i grafen $o_{ij} \rightarrow o_{kl}$, for hvilke det gælder, at $C_{ij} > t$

Hvis objektfunktionen er TWT, og $d_i > t$ for noget job, da indgår t tillige i nutiden, dvs. $\mathcal{N} = (P, D, F, t)$.

Fortid og fremtid kræver ingen præcis definition men udgør groft sagt vejen fra hhv. søgetræets rod til den aktuelle løsning og fra den aktuelle løsning til et blad.

Disse definitioner har den egenskab, at starttidspunktet for operationer i fortiden ligger fast. Det er nemlig kun operationer, der p.t. er — eller senere i løsningsprocessen bliver — involveret i en konflikt, hvis starttidspunkt kan ændres. Og den tidligste konflikt involverer jo netop operationer med $S_{ij} \geq t$, jvf. definitionen af t.

[3] Da den indbyrdes rækkefølge af operationerne på hvert job er givet, er det tilstrækkeligt blot at betragte den sidste operation — om nogen — på hvert job, der opfylder betingelsen.

Definitionerne har også den egenskab, at to løsninger med fælles nutid vil have samme løsning, evt. bortset fra objektfunktionsværdien, der ved TWT også afhænger af fortiden og af t. Dette kan indses ved at betragte tilsvarende problemer, hvor alle operationer i P er fjernet sammen med eventuelle tilhørende kanter i grafen, men hvor starttidspunkterne for den første tilbageværende operation på hvert job bevares som en nedre grænse for operationens starttidspunkt. De to problemer vil da være identiske og således have samme løsning, bortset fra fortidens bidrag til objektfunktionen.

5.2.2 Beregning af objektfunktion

Vi omskriver nu makespan af en løsning:

$$\begin{aligned} O_{makespan}(E) &= \max_{J_i \in \mathcal{J}}(C_i) \\ &= \max[\max_{J_i \in P}(C_i), \max_{J_i \in \mathcal{J} \setminus P}(C_i)] \\ &= \max_{J_i \in \mathcal{J} \setminus P}(C_i) \\ &= t + \max_{J_i \in \mathcal{J} \setminus P}(C_i - t) \end{aligned}$$

Her gælder det altså, at to løsninger med samme nutid adskiller sig ved

$$Q_{makespan}(E) = t \tag{5.1}$$

For TWT er det en anelse mere kompliceret:

$$\begin{aligned} O_{TWT}(E) &= \sum_{J_i \in \mathcal{J}} w_i T_i \\ &= \sum_{J_i \in P} w_i T_i + \sum_{J_i \in \mathcal{J} \setminus P} w_i T_i \\ &= \sum_{J_i \in P} w_i T_i + \sum_{J_i \in \mathcal{J} \setminus P} w_i t + \sum_{J_i \in \mathcal{J} \setminus P} w_i (T_i - t) \end{aligned}$$

For to løsninger med samme nutid vil sidste sumtegn give samme resultat. Jvf. definitionen af D er $C_i - t$ i de to løsninger nemlig ens, og da enten t er indeholdt i nutiden og dermed ens for de to løsninger, eller $d_i \leq t$ for alle jobs og dermed $d_i < C_i$, da er også $T_i - t$ ens.

For to løsninger med samme fortid vil objektfunktionsværdien således kun adskille sig ved de to første sumtegn. Denne størrelse betegnes Q_{TWT}:

$$Q_{TWT}(E) = \sum_{J_i \in P} w_i T_i + \sum_{J_i \in \mathcal{J} \setminus P} w_i t \tag{5.2}$$

Betragt nu to delløsninger E_1, E_2 med identisk nutid og således også samme fremtid. Hvis den optimale løsning af E_1 har objektfunktionsværdien $O(E_1)$, da vil den optimale løsning af E_2 give objektfunktionsværdien

$$O(E_2) = O(E_1) - Q(E_1) + Q(E_2) \tag{5.3}$$

Dette gælder for både makespan og TWT.

Vi kan således ikke blot bruge Q som dominanskriterium til at beskære søgetræet, men vi kan også bruge den til at beregne den optimale løsningsværdi for en delløsning, uden eksplicit at løse den.

5.2.3 Eksempel

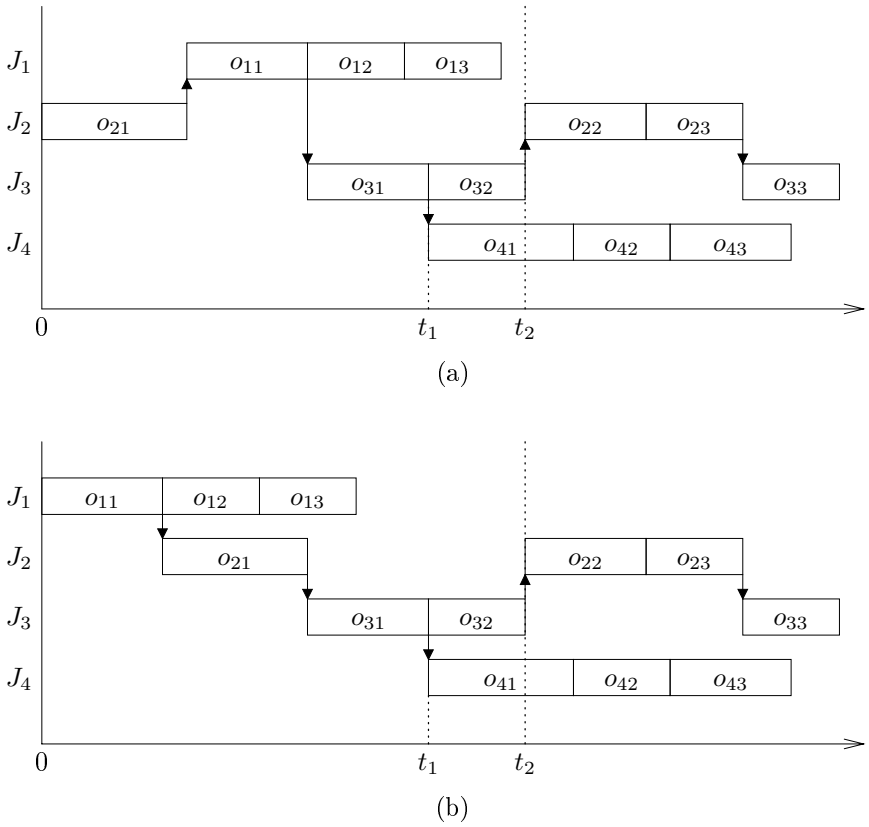
Figur 5.3 viser to delløsninger med samme nutid. I dette eksempel er o_{ij} tilknyttet maskine M_j, dvs. jobbene benytter maskinerne i samme rækkefølge. Deadline for jobbene er det tidligst mulige afslutningstidspunkt, $d_i = \sum p_{ij}$, og vægtene w_i er 1 for alle jobs.

Bortset fra o_{11}, o_{12}, o_{13} og o_{21} er starttidspunkterne for operationerne de samme i de to løsninger. Særlig kan nævnes, at $S_{31} = p_{11} + p_{21}$ i begge tilfælde. Pilene på figuren svarer til kanter i grafrepræsentationen, dog er kanterne mellem operation på samme job ikke angivet, idet disse kan ses direkte ved at læse hver vandret række fra venstre mod højre.

Tidligste konflikt i begge løsninger er mellem operationerne o_{22} og o_{42}, og tidspunktet for denne er $t_1 = \min(S_{22}, S_{42})$. Nutiden er således beskrevet ved følgende fire størrelser, jvf. ovenstående definition.

P : $\{o_{11}, o_{12}, o_{13}, o_{21}, o_{31}, o_{32}\}$
D : $\{(o_{41} : C_{41} - t_2)\}$
F : $\{o_{23} \rightarrow o_{33}\}$

Ser man bort fra operationerne P, hvis starttidspunkter ikke vil kunne ændres i denne branch, ses det, at de to problemer er ens, og at de således vil have samme løsning.



Figur 5.3: To delløsninger med samme nutid

Antag at vi allerede har undersøgt løsningen vist i (a), E_a, og nu møder løsningen vist i (b), E_b. Hvis vi løser for TWT og kender den optimale løsning af E_a, da vil vi umiddelbart kunne beregne løsningsværdien af E_b.

Af figuren ses, at $T_1 = p_{21}$ i E_a, mens $T_1 = 0$ i E_b. J_1 er det eneste job, der afsluttes i fortiden, dvs. at $Q(E) = w_1 T_1$. Jvf. (5.3) er objektfunktionsværdien af E_b således $O(E_b) = O(E_a) - w_1 p_{21}$. E_b dominerer altså E_a.

Hvis man løser for makespan, er $O(E_b) = O(E_a)$.

5.2.4 Anvendelse i branch-and-bound

Algoritme 5.3 skitserer, hvorledes dette kan udnyttes ved løsning vha. branch-and-bound. Algoritmen gælder for måde makespan og TWT.

GetPresent returnerer den aktuelle nutid $\mathcal{N}$. *knownPresents* er en afbildning mellem kendte nutider og deres tilhørende værdier. For hver nutid lagres den tidligere fundne værdi af $Q(E)$ samt den bedste løsning i den pågældende branch, eller NIL hvis der ikke er fundet en løsning for den givne nutid, dvs. hvis søgetræet blev beskåret vha. LowerBound-funktionen, før en brugbar løsning blev fundet.

Hvis søgningen møder en allerede kendt nutid $\mathcal{N}$(linje 8), forsøges det at udnytte dette. Den kendte og den aktuelle værdi af Q sammenlignes, og er den aktuelle ringere end den kendte, beskæres det aktuelle søgetræ (linje 11).

Hvis den kendte derimod er løst mindst lige så godt som tidligere, og den kendte situation i øvrigt førte til en brugbar løsning (linje 12), da kan den aktuelle objektfunktionsværdi umiddelbart beregnes vha. (5.3). Er værdierne af Q og dermed objektfunktionsværdien den samme (linje 13), returneres blot den kendte løsning (linje 14). Hvis den derimod viser sig at være bedre end den hidtil bedste, findes løsningen svarende til den beregnede objektfunktionsværdi ved at sammenflette fortiden hørende til den aktuelle løsning med fremtiden for den kendte løsning vha. MergeSolutions (linje 18).

Hvis $E_{prev} =$ NIL og $Q(E) < Q_{prev}$, da har proceduren tidligere mødt den aktuelle nutid, men søgetræet blev beskåret, så der ikke nåede at blive fundet en brugbar løsning. Det er dog muligt, at den nye og bedre løsning af

```
BranchAndBound(E)
 1  c ← GetEarliestConflict(E)
 2  if c = nil then
 3      ...
 4  else
 5      (o_ij, o_kl) ← c
 6      t ← min(E_ij, E_kl)
 7      N ← GetPresent(E, t)
 8      if N ∈ knownPresents then
 9          (Q_prev, E_prev) ← knownPresents[N]
10          if Q(E) > Q_prev then
11              return nil
12          else if E_prev ≠ nil then
13              if Q(E) = Q_prev then
14                  return E_prev
15              else
16                  O_cur ← O(E_prev) − Q_prev + Q(E)
17                  if O_cur < O(UB) then
18                      E_best ← MergeSolutions(N, E, E_prev)
19                      UB ← O(E_best)
20                      return E_best
21                  end if
22              end if
23          end if
24      end if
25      ... (algoritme 5.1, linje 10–21)
26      knownPresents[N] ← (Q(E), E_min)
27      return E_min
28  end if
```

Algoritme 5.3: Brug af dominanskriterium ved kronologisk konfliktløsning.

fortiden vil føre til en brugbar løsning, hvorfor branch-and-bound-søgningen fortsættes. Dette betyder ganske vist, at en allerede kendt del af søgetræet undersøges igen. Alternativet vil dog være, at man først søger videre fra en given nutid, når man har løst fortiden til optimalitet. Herved bliver søgningen reelt en bredde-først-søgning, hvilket bl.a. har den ulempe, at man skal lagre et meget stort antal delløsninger svarende til knuder i branch-and-bound-søgetræet, hvilket ofte i praksis viser sig at være umuligt.

Funktionens returværdi er den optimale løsning af den aktuelle delløsning (linje 20, samt linje 27 hvis betingelsen i linje 15 i algoritme 5.1 er sand mindst én gang) eller en løsning med samme objektfunktionsværdi (linje 14) eller NIL (linje 11 og evt. 27), hvis søgetræet blev beskåret, før en brugbar løsning blev fundet.

Algoritme 5.3 viser en dybde-først-søgning men vil let kunne modificeres til andre søgestrategier. Man skal blot sikre sig, at man først registrerer information om en given nutid (linje 26), når hele det underliggende træ er gennemsøgt.

5.2.5 Dominans ved forskellige nutider

Det generelle dominanskriterium ved minimering af en objektfunktion lyder, at hvis $N(E_1) \preceq N(E_2)$ og $Q_N(E_1) \geq Q_N(E_2)$, da er $N(E_1)$ domineret af $N(E_2)$ og behøver således ikke at blive betragtet yderligere. $N(E)$ skal opfattes som en del af løsningen E, og $Q_N(E)$ er et udtryk for denne dels bidrag til objektfunktionsværdien.

Antag at vi betragter to delløsninger E_1 og E_2 samt en del af hver af disse, $N(E_1), N(E_2)$. Dominanskriteriet siger her, at hvis $N(E_2)$ omfatter mindst lige så meget som $N(E_1)$ og til en mindre omkostning, da er $N(E_1)$ domineret af $N(E_2)$, dvs. der findes en optimal løsning, hvor $N(E_1)$ ikke indgår.

I afsnit 5.2.1 definerede vi et dominanskriterium, hvor N udgøres af nutiden $\mathcal{N}$, og i de følgende afsnit betragtede vi det tilfælde, hvor to delløsninger havde samme nutid, dvs. hvor $N_1 = N_2$. Vi vil her omtale en anvendelig ordning $\preceq$ af nutiderne.

Betragt to nutider $N(E_1) = \mathcal{N}_1 = (P_1, D_1, F_1, t_1)$ og $N(E_2) = \mathcal{N}_2 = (P_2, D_2, F_2, t_2)$. Det gælder da, at $N(E_1) \preceq N(E_2)$, hvis alle følgende punkter er opfyldt:

- $P_1 \supseteq P_2$
- for hvert element $(o_{ij} : e_1) \in D_1$, da findes et element $(o_{ij} : e_2) \in D_2$ hvor $e_2 \geq e_1$, eller der findes et element $(o_{il} : e_2) \in D_2$, hvor $l < j$, dvs. o_{il} er en tidligere operation på samme job
- $F_1 \subseteq F_2$
- $t_1 \leq t_2$

Med denne definition kan vi foretage yderligere beskæringer af søgetræet vha. dominans; dog kan vi ikke benytte (5.3) til at beregne objektfunktionsværdier.

Det er dog ikke ligetil at sammenligne disse værdier effektivt. Mens lighed kan testes ved binær søgning på en strengrepræsentation af nutiden, er det ikke trivielt at effektivt sammenligne en given nutid med alle hidtil mødte mht. $\prec$. Et lineært gennemløb må antages at være meget tidskrævende. Dette er ikke undersøgt nærmere, så denne rapport benytter sig kun af dominans ved lighed mellem nutiderne.

5.3 Bounding

I forbindelse med kantindsættelse omtalte vi i afsnit 5.1.2 en meget simpel nedre grænse objektfunktionsværdien for hhv. makespan og TWT. Disse grænser er udmærkede i forbindelse med kantindsættelse, idet der her skal ske mange grænseberegninger — mindst to for hvert par af operationer på samme maskine, hvis rækkefølge ikke allerede er fastlagt — men til bounding af branch-and-bound-søgningen kan vi tillade os at bruge lidt mere beregningstid mod til gengæld at få en tættere grænse.

En måde at finde en nedre grænse for løsningsværdien af en given løsning er at betragte en delmængde af problemet og finde en løsning til dette. Eksempelvis kan man betragte JSS-problemet bestående en enkelt eller en delmængde af maskinerne og så finde en løsning for dette.

For at beregne en nedre grænse for en given delløsning E, betragter vi isoleret operationerne tilknyttet en enkelt maskine. Vi søger nu at planlægge operationerne på denne maskine således, at objektfunktionsværdien af E stiger mindst muligt. Denne beregning af stigningen i objektfunktionsværdien som følge af en given løsning af delproblemet sker uden at ændre E. Denne procedure foretages for hver enkelt maskine, og den største resulte-

rende objektfunktionsværdi udgør en nedre grænse for objektfunktionsværdien af det samlede problem.

Imidlertid er JSS for problemer bestående af kun én maskine som nævnt i afsnit 3.8 også $\mathcal{NP}$-hårdt. Da nedre grænser skal beregnes mange gange i forbindelse med løsning, er løsningsværdien af en enkelt maskine derfor ikke en hensigtsmæssig nedre grænse for det samlede problem.

I stedet relakseres problemet yderligere, således at vi nu også tillader, at en operation må afbrydes for senere at genoptages. Denne variant kan nemlig løses effektivt.

Dette relakserede problem falder ikke ind under definitionen i kapitel 3. Bl.a. er det ikke tilfældet, at en maskine eksklusivt behandler én operation o_{ij} i et interval $[S_{ij}; S_{ij} + p_{ij}[$. En maskine kan stadig kun behandle én operation på et givet tidspunkt, men i stedet for ét interval sker behandlingen af o_{ij} nu i en række intervaller $[t_1; t_1 + p_1[\cup \ldots \cup [t_r; t_r + p_r[$, hvor $\sum_{s=1}^{r} p_s = p_{ij}$. Dette betyder bl.a., at løsningen ikke umiddelbart kan repræsenteres ved en graf som beskrevet i afsnit 3.5.

Den eksakte metode afhænger af objektfunktionen. Den har dog fælles træk for makespan og TWT. For begge gælder, at planlægningen sker kronologisk uden backtracking eller lign. ved et enkelt gennemløb af operationerne. Undervejs opdateres variablen t, så den svarer til sluttidspunktet for det senest planlagte interval.

Der vedligeholdes en mængde af operationer R, der kandiderer til at blive planlagt til tiden t, dvs. operationer hvor $S_{ij} \leq t$ og som endnu ikke er færdigbehandlede. Denne mængde kaldes de *frigivne* operationer.

I det følgende refererer S_{ij}, C_{ij} og q_{ij} til hhv. start- og sluttidspunkter samt hale i den aktuelle løsning af det oprindelige problem, mens S'_{ij} og C'_{ij} refererer til værdierne for den fundne løsning af det relakserede problem. Bemærk at pga. ovennævnte relaksering gælder det ikke nødvendigvis, at $S'_{ij} + p_{ij} = C'_{ij}$.

Desuden introduceres variablen l_{ij}, der angiver, hvor meget behandlingstid der resterer for operationen o_{ij}. Initielt er $l_{ij} = p_{ij}$, og for hvert interval, hvor operationen behandles, tælles l_{ij} tilsvarende ned, indtil hele behandlingstiden er afviklet, og $l_{ij} = 0$.

5.3.1 Makespan

Ved makespan kaldes den løsning, der resulterer af denne metode, for Jackson's Preemptive Schedule.

Når en operation afsluttes til tiden C'_{kl}, udgør $C'_{kl} + q_{kl}$ en nedre grænse for objektfunktionsværdien. Det betyder, at jo større hale en operation har, jo tidligere skal den afsluttes for ikke at øge objektfunktionsværdien.

Metoden består i til hver en tid at behandle den frigivne operation o_{kl} fra R, som har den længste hale q_{kl}. Herved sikres, at hver afsluttet operation har så lille indvirkning på objektfunktionsværdien som muligt. R er altså en prioritetskø ordnet mht. q_{kl}.

Hvis t under behandlingen af o_{kl} passerer starttidspunktet S_{ij} for en ny operation o_{ij}, opdateres R til også at indeholde denne operation. Hvis den tilføjede operation har længere hale end o_{kl}, da standses afviklingen af o_{kl} og i stedet fortsættes med o_{ij}, indtil denne er færdigbehandlet, eller indtil starttidspunktet for en operation med endnu længere hale passeres.

Eksempel

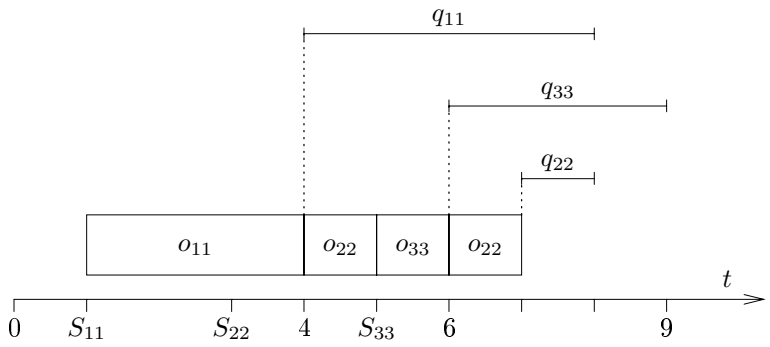
Betragt et problem bestående af tre jobs, hvor $o_{11}, o_{22}, o_{33} \in M_p$[4]. Disse operationer har i den aktuelle løsning tilknyttet følgende værdier:

	S_{ij}	p_{ij}	q_{ij}
o_{11}	1	3	4
o_{22}	3	2	1
o_{33}	5	1	3

Figur 5.4 viser, hvordan Jackson's Preemptive Schedule ser ud for denne maskine. De vandrette linjestykker over operationerne angiver operationernes haler. De højre endepunkter er $C'_{kl} + q_{kl}$ og udgør hver en nedre grænse for objektfunktionsværdien af det samlede problem.

Da o_{11} til at starte med er den eneste frigivne operation, er det således også den frigivne operation med længst hale, og behandling påbegyndes. Til tiden $t = 3$ frigives o_{22}, men da $q_{11} = 4 > q_{22} = 1$ fortsættes behandlingen af

[4]Bemærk at andet indeks, dvs. j i o_{ij}, ingen relevans har for dette eksempel.



Figur 5.4: Jackson's Preemptive Schedule for maskine med tre jobs.

o_{11}, indtil o_{11} er færdigbehandlet til tiden $C'_{11} = 4$, hvorefter o_{22} påbegyndes. Når $t = 5$ frigives o_{33}, og da denne også har en længere hale end o_{22}, afbrydes behandlingen af o_{22} til fordel for o_{33}. Sidstnævnte færdiggøres til tiden $C'_{33} = 6$, hvor o_{22} så genoptages og færdiggøres til tiden $C'_{22} = 7$.

Største værdi af $C'_{kl} + q_{kl}$ er 9, hvilket således udgør den nedre grænse for det samlede problem baseret på løsning af M_p med ovennævnte relaksering. Proceduren foretages for hver maskine $M_p \in \mathcal{M}$, og den største værdi udgør så metodens bedste bud på en nedre grænse.

Algoritme

Metoden er beskrevet i algoritme 5.4. t er tidspunktet for næste behandling af en operation. t_{next} er det tidligste starttidspunkt for operationer, der endnu ikke er frigivet (linje 9) — eller ∞, hvis alle operationer er frigivet eller afsluttede (linje 11).

Operationerne planlægges ved hele tiden at vælge den med den længste hale (linje 14), så længe $t < t_{next}$ (linje 13). Den valgte operation behandles så meget, der kan nås inden tidspunktet t_{next}, dvs. at t øges, og l_{kl} formindskes tilsvarende (linje 16–17). Hvis den valgte operation når at færdiggøres (linje 18), fjernes den fra mængden af frigivne operationer R, således at R stadig svarer til definitionen i linje 7, og den nedre grænse for objektfunktionsværdien af det samlede problem opdateres som beskrevet herover (linje 20).

```
LowerBound(E)
 1  return max_{M_p ∈ M}(SingleMachineLowerBound(M_p))

SingleMachineLowerBound(M_p)
 2  t_next ← 0
 3  LB ← 0
 4  l_ij ← S_ij ∀ o_ij ∈ M_p
 5  while t_next < ∞ do
 6      t ← t_next
 7      R ← {o_ij : S_ij ≤ t, l_ij > 0, o_ij ∈ M_p}
 8      if ∃ o_ij ∈ M_p, S_ij > t, then
 9          t_next ← min(S_ij : S_ij > t, o_ij ∈ M_p)
10      else
11          t_next ← ∞
12      end if
13      while t < t_next ∧ R ≠ ∅ do
14          (k, l) ← argmax_(i,j)(q_ij : o_ij ∈ R)
15          Δt ← min(l_kl, t_next − t)
16          t ← t + Δt
17          l_kl ← l_kl − Δt
18          if l_kl = 0 then
19              R ← R \ {o_kl}
20              LB ← max(LB, t + q_kl)
21          end if
22      loop
23  loop
24  return LB
```

Algoritme 5.4: Beregning af nedre grænse for makespan vha. Jackson's Preemptive Schedule.

Når t bliver lig t_{next}, eller når der ikke er flere frigivne operationer, da springes ud af den indre **while**-løkke (linje 13–22), således at den eller de operationer, der nu har $S_{ij} = t$, kan blive tilføjet R (linje 7).

Køretid

Dette problem kan løses til optimalitet i tiden $\mathcal{O}(n \log n)$ — hvor n som sædvanligt er $|M_p|$ — hvis prioritetskøen R repræsenteres som en hob sorteret mht. q_{ij}. Indsættelse af operationer i R (linje 7) kan således ske i $\mathcal{O}(\log n)$ pr. indsættelse, hvilket højst sker n gange, og udvælgelse af operationen med længst hale (linje 14) kan ske i konstant tid.

Derudover er den dominerende operation en initiel sortering af operationerne i M_p mht. starttidspunkt S_{ij} til brug i linjerne 7 og 9, hvilket kan ske i tiden $\mathcal{O}(n \log n)$.

Den ydre **while**-løkke (linje 5–23) gennemløbes maksimalt n gange, idet der i hvert gennemløb tilføjes en ny operation til R (linje 7).

Den indre **while**-løkke (linje 13–22) gennemløbes i alt højst $2n$ gange: hvis $l_{kl} \leq t_{next} - t$ i linje 15, da vil resten af o_{kl} blive afviklet, og operationen vil blive fjernet fra R (linje 19), hvilket højst kan ske n gange. Gælder derimod, at $l_{kl} > t_{next} - t$, da bliver t tildelt værdien t_{next}, og der springes ud af den indre løkke, hvilket højst kan ske så mange gange, som den indre løkke nås, dvs. antallet af gennemløb af den ydre løkke.

5.3.2 Total weighted tardiness

I det følgende foreslås en ny metode til bounding af TWT baseret på idéen bag Jackson's Preemptive Schedule, der er beskrevet i forrige afsnit i forbindelse med bounding af makespan.

Et bud på en nedre grænse-beregning for TWT ville være at køre metoden for makespan én gang for hvert job J_k, hvor alle forekomster af q_{ij} så erstattes af q_{ij}^k. Dette vil dog sjældent give en nedre grænse, der er bedre end selve objektfunktionsværdien af den aktuelle løsning.

Forklaringen herpå skal findes i, at $q_{ij}^k = -\infty$ for operationer, hvor der ikke findes en vej $o_{ij} \rightsquigarrow \mathcal{V}_k$. Ikke nok med, at disse operationer i givet fald ikke bidrager til den nedre grænse (linje 20 i algoritme 5.4), når de afsluttes,

men de vil også pga. deres lille hale altid vige pladsen for andre operationer, når der skal udvælges en operation til behandling (linje 14).

Hvis der eksisterer en vej $o_{ij} \rightsquigarrow \mathcal{V}_k$, og der samtidig findes en vej $o_{kl} \rightsquigarrow o_{ij}$, hvor o_{kl} er operationen fra J_k på samme maskine, da vil denne vej som regel have indgået i beregningen af den aktuelle værdi af S_{ij}[5], og således vil den indbyrdes ordning af o_{ij} og o_{kl} ikke direkte give anledning til yderligere skærpelse af den nedre grænse.

I stedet benyttes en anden metode: for operation o_{ij} betragtes deadlinen mht. det job, operationen tilhører, dvs. d^i_{ij}. Når operationerne planlægges, vil et afslutningstidspunkt C'_{ij} større end d^i_{ij} give en direkte forøgelse af objektfunktionen på $w_i(C'_{ij} - d^i_{ij})$.

Udover mængden af frigivne operationer R, hvor $S_{ij} \leq t$, benyttes også en delmængde heraf, D, som består af de operationer, hvis deadline d^i_{ij} er nået, dvs. hvor $d^i_{ij} \leq t$.

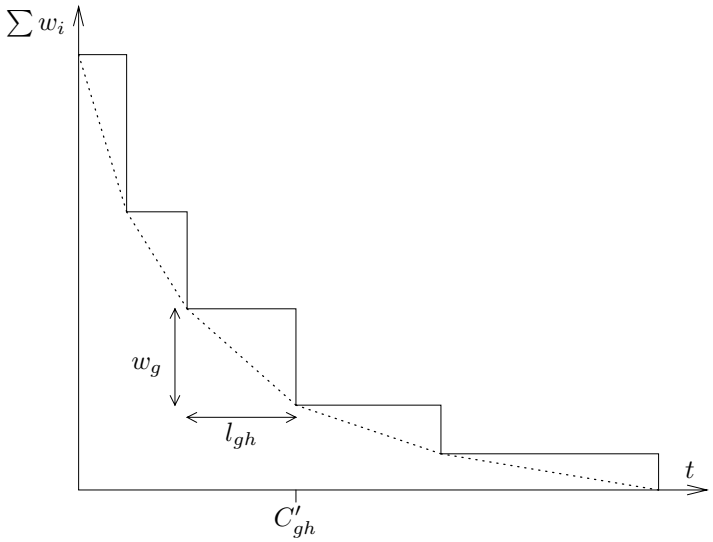
Vi betragter nu situationen, hvor $D \neq \emptyset$. Uanset hvilken operation o_{kl} vi vælger at behandle i et tidsrum Δt, da "skubber" vi samtidig de øvrige operationer foran os. Da deadline allerede er nået for operationerne i D, medfører behandlingen af o_{kl} således en forøgelse af tardiness på Δt for jobbet tilhørende hver af de øvrige operationer i D. Objektfunktionsværdien stiger dermed i alt med

$$\sum_{o_{ij} \in D \setminus \{o_{kl}\}} w_i \, \Delta t$$

Vi ønsker at afvikle operationerne i en sådan rækkefølge, at mindst muligt vægt skubbes foran os, og at objektfunktionsværdien derved stiger mindst muligt. Dette opnås ved at behandle operationerne i D i rækkefølge efter aftagende værdi af brøken w_i/l_{ij}.

At denne strategi fører til den optimale løsning, er illustreret i figur 5.5, der viser en sådan løsning. Her er den vægt, der skubbes foran, afbildet som funktion af tiden. Arealet under grafen udgør således den samlede stigning i objektfunktionsværdien.

[5] Dette afhænger af, hvilken metode der benyttes til beregning af S_{ij}. Ved benyttelse af længste vej-metoden som omtalt i afsnit 3.7.1 — der er den primære metode benyttet i dette projekt — er det tilfældet.



Figur 5.5: Optimal løsning af en maskine.

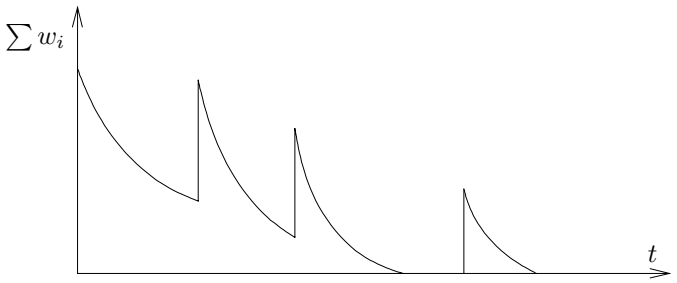
w_i/l_{ij} optræder som hældningen af den stiplede linje med modsat fortegn. Operationerne er planlagt mht. aftagende værdi af w_i/l_{ij}, hvilket er årsagen til kurvens konkave form. Af kurvens form kan konkluderes, at denne rækkefølge er optimal mht. minimering af stigningen i objektfunktionsværdien.

Hvis $D = \emptyset$ vil det ikke medføre en umiddelbar stigning i objektfunktionsværdien, uanset hvilken operation der vælges. Dog ønsker vi stadig af skubbe mindst mulig vægt foran os, så her ordnes operationerne på samme måde; dog vælges de fra R i stedet for D. Når en operations deadline passeres, vil D ikke længere være tom, og behandlingen af den aktuelle operation afbrydes. Der fortsættes herefter med behandling af operationer fra D.

Ved behandling af operationer fra såvel D som R gælder, at behandlingen af en operation afbrydes, såfremt en operation med større værdi af w_i/l_{ij} tilføjes mængden.

Figur 5.6 viser en fortegnet udgave af, hvordan kurven kunne se ud for en maskine med et større antal operationer. De lodrette spring svarer til, at deadline passeres for en operation og derved inkluderes i R. Når $D = \emptyset$,

har ingen operation overskredet sin deadline, og grafen er derfor flad med funktionsværdien 0. Det samme gælder selvfølgelig, når $R = \emptyset$.



Figur 5.6: Fortegnet løsning af maskine.

Algoritme

Algoritme 5.5 beskriver denne procedure. Der er mange ligheder med metoden for makespan (algoritme 5.4), så denne udgave vil ikke blive beskrevet i samme detalje.

Blandt forskellene kan nævnes, at operationerne her afvikles i mindre intervaller Δt end ved makespan. Begge steder afvikles indtil operationen er færdigbehandlet, eller starttidspunktet for en ny operation passeres, t_{next}. Her er dog en yderligere grænse, idet vi også standser behandlingen, hvis deadline for en frigivet operation nås (linje 22).

Som værdi for d_{ij}^i bruges et konstant udtryk for d_{ij}^i i den initielle løsning, hvilket giver en god øvre grænse for værdien med et betydeligt mindre beregningsarbejde end ved beregning af længste veje eller lign.:

$$d_{ij}^i = \max(d_i, C_i) - \sum_{l>j} p_{il}$$

dvs. den største værdi af enten jobbets deadline eller jobbets afslutningstidspunkt i den aktuelle løsning, fratrukket den samlede behandlingstid af de resterende operationer på jobbet.

Udvælgelsen af den operation, der skal behandles, sker som nævnt herover ved at vælge den operation i enten R eller D, for hvilken det gælder, at w_i/l_{ij} er størst (linje 17 og 19).

```
LowerBound(E)
 1  ΔWT_max = max_{M_p ∈ M}(AdditionalWeightedTardiness(M_p))
 2  return O_TWT(E) + ΔWT_max
AdditionalWeightedTardiness(M_p)
 3  t_next ← 0
 4  ΔWT ← 0
 5  l_ij ← p_ij ∀ o_ij ∈ M_p
 6  while t_next < ∞ do
 7      t ← t_next
 8      R ← {o_ij : S_ij ≤ t, l_ij > 0, o_ij ∈ M_p}
 9      if ∃ o_ij ∈ M_p, S_ij > t then
10          t_next ← min(S_ij : S_ij > t, o_ij ∈ M_p)
11      else
12          t_next ← ∞
13      end if
14      while t < t_next ∧ R ≠ ∅ do
15          D ← {o_ij : d^i_ij ≤ t, o_ij ∈ R}
16          if D ≠ ∅ then
17              (k, l) ← argmin_(i,j)(w_i/l_ij : o_ij ∈ R)
18          else
19              (k, l) ← argmin_(i,j)(w_i/l_ij : o_ij ∈ D)
20          end if
21          if ∃ o_ij ∈ R, d^i_ij > t then
22              d_next ← min(d^i_ij : d^i_ij > t, o_ij ∈ R)
23          else
24              d_next ← ∞
25          end if
26          Δt ← min(l_kl, , t_next − t, d_next − t)
27          t ← t + Δt
28          l_kl ← l_kl − Δt
29          if l_kl = 0 then
30              R ← R \ {o_kl}
31          end if
32          ΔWT ← ΔWT + Σ_{o_ij ∈ D\{o_kl}} w_i Δt
33      loop
34  loop
35  return ΔWT
```

Algoritme 5.5: Beregning af nedre grænse for TWT.

For hver behandling adderes til ΔWT den lokale stigning i objektfunktionsværdien (linje 32). Til sidst returneres den samlede stigning, som den lagte plan medfører.

Denne procedure foretages for hver maskine (linje 1), og den største af de beregnede stigninger anvendes som en nedre grænse for stigningen.

Køretid

Som ved makespan sorteres operationerne i M_p her initielt mht. S_{ij}.

R er her repræsenteret ved en hob sorteret mht. d^i_{ij}, mens D er repræsenteret ved en hob sorteret mht. w_i/l_{ij}. Indsættelse af operationer i R og D (hhv. linje 8 og linje 15) kan således ske i tiden $\mathcal{O}(\log n)$, mens udvælgelse af elementer (hhv. linje 15 og linje 19) sker i konstant tid. Til gengæld sker udvælgelse af det mindste element mht. w_i/l_{ij} fra R (linje 17) i lineær tid som funktion af $|R|$.

I lighed med makespan gennemløbes den ydre **while**-løkke (linje 6–34) maksimalt n gange.

Den indre **while**-løkke (linje 14–33) gennemløbes maksimalt $3n$ gange: Δt antager i linje 26 én af værdierne t_{next}, d_{next} eller l_{kl}. Antager den sidstnævnte værdi, vil operationen o_{kl} blive fjernet fra R (linje 30), hvilket højst kan ske n gange. t_{next} er defineret som S_{ij} for en eller anden operation $o_{ij} \in M_p$, dvs. den kan antage højst n forskellige værdier. Ligeledes er d_{next} er defineret som en eller anden d^i_{ij} og kan også højst antage n forskellige værdier. Da $\Delta t > 0$ vil den samme værdi ikke blive brugt ved bestemmelse af Δt mere end én gang. Alt i alt vil den indre løkke altså højst blive gennemløbet $3n$ gange.

Grundet den lineære køretid af linje 17, er kompleksiteten af metoden i værste fald $\mathcal{O}(n^2)$. Denne linje vil kun blive udført, når ingen frigivne operationer har nået deres deadline, dvs. at worst-case-køretiden primært vil komme til udtrykt i problemer med relativt sene deadlines set i forhold til det tidligste starttidspunkt r_i og varigheden af operationerne på hvert job. I problemer med tidlige deadlines vil køretiden nærmere sig $\mathcal{O}(n \log n)$ som for makespan.

Jo nærmere branch-and-bound-søgningen når på et blad i søgetræet, jo flere kanter vil der være indsat i grafen, og jo mindre spænd vil der være mellem

S_{ij} og d^i_{ij}. Så også her vil køretiden nærmere sig $\mathcal{O}(n \log n)$. Dette er en heldig egenskab, idet antallet af knuder jo stiger eksponentielt, jo dybere man når i søgetræet.

5.4 Shifting bottleneck

Shifting bottleneck (SB) er en heuristik, der har vist sig at være effektiv ved løsning af JSS med såvel makespan som total weighted tardiness som objektfunktion [1, 20].

Heuristikken kan eksempelvis bruges til hurtigt at finde en god initiel løsning, hvis objektfunktionsværdi bruges som undergrænse ved løsning af problemet til optimalitet.

Princippet i algoritmen er, at man — i lighed med bounding-funktionerne omtalt i afsnit 5.3 — opdeler problemet i en række delproblemer, som så løses ét ad gangen isoleret set, dvs. mht. til værdierne af S_{ij} og q_{ij}, der afhænger af den aktuelle løsning af hele problemet. Denne lokale løsning kan ske heuristisk eller eksakt.

Hvert delproblem består i én maskine. Udvælgelsen af det næste delproblem sker ved for hvert delproblem at finde en løsning, der minimerer stigningen i objektfunktionen af det samlede problem. Herefter vælges det delproblem, der giver den største stigning, og kanterne i løsningsgrafen overføres til det oprindelige problem.

Dette virker måske ulogisk at vælge den, der giver den største stigning, men da alle maskiner jo skal løses før eller siden, kan man lige så godt løse de mest problematiske først som sidst. Faktisk er SB baseret på antagelsen af, at den maskine, der giver størst stigning i objektfunktionsværdien, har dominerende betydning for strukturen af den samlede løsning, dvs. at den udgør flaskehalsen og derfor bør løses først.

Vi opererer altså med det oprindelige problem $\mathcal{P}$ med en tilhørende aktuel delløsning E, samt for hver maskine M_p et delproblem $\mathcal{P}_p$ og tilhørende løsning E_p. I det følgende er alle variable mærket med $'$ tilhørende et specifikt delproblem[6].

[6] Bemærk at der defineres ét delproblem pr. maskine $M_p \in \mathcal{M}$, dvs. at alle variable

Mere præcist er delproblemet $\mathcal{P}_p$ for maskine M_p et job-shop-skeduleringsproblem bestående udelukkende af operationerne tilknyttet maskine M_p i det oprindelige problem $\mathcal{P}$. Dette problem består således af lige så mange jobs $\mathcal{J}' = \{J'_1, \ldots, J'_n\}$, men kun én maskine $\mathcal{M}' = \{M_p\}$. Jobbet J'_i består kun af operationen $o_{ij} \in M_p$. Tidligste starttidspunkt for J'_i er $r'_i = S_{ij}$, dvs. starttidspunktet i den aktuelle delløsning E.

En DPC i E kan repræsenteres som en kant i E_p af samme længde. Dvs. at findes der en vej $o_{ij} \rightsquigarrow o_{kl}$ i E, hvor $o_{ij}, o_{kl} \in M_p$, da kan der indsættes en kant $o_{ij} \rightarrow o_{kl}$ med længden $L(o_{ij}, o_{kl})$ i E_p.

En løsning af delproblemet $\mathcal{P}_p$ består i en angivelse af rækkefølgen af operationerne på den pågældende maskine M_p. Denne løsning "overføres" så til $\mathcal{P}$ ved at tilføje kanterne i E_p til E, dvs. at sætte E til $E \cup E_p$. E er brugbar, når der er overført en brugbar løsning af hvert delproblem.

5.4.1 Objektfunktioner for delproblemer

Som sagt ønsker vi at finde det delproblem, hvis løsning maksimerer den minimale stigning i objektfunktionsværdien af $\mathcal{P}$.

Vi opstiller derfor en objektfunktion O' for $\mathcal{P}_p$, der er defineret som en nedre grænse for stigningen i objektfunktionsværdien af $\mathcal{P}$ ved overførsel af løsningen af $\mathcal{P}_p$ til $\mathcal{P}$. Ved at opstille denne funktion er vi fri for at overføre løsningen for at finde stigningen i objektfunktionsværdien.

For hver maskine M_p ønsker vi altså at finde den løsning E_p, der minimerer $O'(E_p)$, hvorefter vi udvælger den, hvor denne værdi er størst, og overfører løsningen heraf til E.

$$\underset{M_p \in \mathcal{M}}{\operatorname{argmax}}[\min_{E'_p}(O'(E'_p))]$$

Stigningen i objektfunktionsværdien af E afhænger bl.a. af operationernes deadline d_{ij}. Først når en operation i en løsning E_p afsluttes senere end d_{ij}, vil den bidrage til en stigning i $O'(E_p)$. Når løsningen E_p overføres til E, vil der tilsvarende ske en stigning i objektfunktionsværdien af E, $O(E)$.

---

mærket med $'$ knytter sig til delproblemet for en bestemt maskine. For at være helt stringent burde disse variable derfor være indekseret med p i stedet for $'$. Dette er dog undladt af typografiske hensyn, med undtagelse af $\mathcal{P}_p$ og E_p.

Til beregning af hale, som benyttes ved beregning af deadline, benyttes her længste vej-metoden som nævnt i afsnit 3.7.1.

Makespan

Det seneste tidspunkt for afslutning af o_{ij} uden at påvirke objektfunktionsværdien af E er givet ved (3.9). Det lokale bidrag fra o_{ij} til objektfunktionsværdien er således $(C'_{ij} - d_{ij})^+$. Objektfunktionsværdien kan godt påvirkes af flere operationer, men i så fald vil den største påvirkning dominere. Vi betragter derfor kun det største bidrag, og vores objektfunktion for E_p bliver så følgende:

$$O'(E_p) = \max_{o_{ij} \in M} (C'_{ij} - d_{ij})^+ \tag{5.4}$$

Ved anvendelse af længste vej-metoden til beregning af halen, er den benyttede øvre grænse for operationens deadline givet ved

$$d_{ij} = O_{makespan}(E) - L(o_{ij}, \mathcal{V}) + p_{ij}$$

Total weighted tardiness

Ved anvendelse af total weighted tardiness som objektfunktion for $\mathcal{P}$, vil en forøgelse ΔT_k af tardiness T_k for et job J_k betyde en tilsvarende forøgelse af objektfunktionen multipliceret med vægten w_k. Dette giver følgende objektfunktion for E_p:

$$O'(E_p) = \sum_k w_k \Delta T_k$$

ΔT_k kan bestemmes ud fra, hvor meget hvert enkelt operation påvirker tardiness for jobbet. Bidraget fra o_{ij} til T_k er $C'_{ij} - d^k_{ij}$. Som ved makespan betragter vi kun den største forøgelse, så vores nedre grænse for ΔT_k er

$$\max_{o_{ij} \in M_p} (C'_{ij} - d^k_{ij})^+.$$

Objektfunktionen for E_p bliver således

$$O'(E_p) = \sum_k w_k \max_{o_{ij} \in M_p} (C'_{ij} - d^k_{ij})^+ \tag{5.5}$$

Analogt med d_{ij} for makespan beregnes d_{ij}^k vha. det aktuelle tidspunkt for afvikling af pseudo-operationen hørende til afløbsknuden samt den længste vej fra operationen til afløbsknuden. Her tages dog højde for, at der evt. ikke findes en vej $o_{ij} \rightsquigarrow \mathcal{V}_k$. d_{ij}^k er således givet ved følgende:

$$d_{ij}^k = \begin{cases} \max(C_k, d_k) - L(o_{ij}, \mathcal{V}_k) + p_{ij} & \text{hvis der findes en vej } o_{ij} \rightsquigarrow \mathcal{V}_k \\ \infty & \text{i modsat fald} \end{cases}$$

5.4.2 Algoritme

Shifting bottleneck er beskrevet i algoritme 5.6. Algoritmen kaldes første gang med parametrene ShiftingBottleneck($\emptyset, \mathcal{M}$).

For hver maskine, der endnu ikke er løst, løser funktionen Solve denne mht. objektfunktionen O' (linje 3). Dette er omtalt nærmere i næste afsnit.

På basis af værdien af denne løsning udvælges det næste delproblem (linje 8), hvis løsning overføres til det samlede problem (linje 9). Algoritmen kalder herefter sig selv rekursivt (linje 14), indtil alle delproblemer er løst (når $\mathcal{M}_0 = \emptyset$ i linje 1) og dermed en brugbar løsning for det samlede problem fundet.

Kriteriet for udvælgelse af næste maskine — nemlig den maskine, hvis løsning medfører den største stigning i objektfunktionen af det samlede problem — er heuristisk, dvs. at det er ikke nødvendigvis vil føre til den bedste løsning at vælge netop den ene maskine, der medfører den største stigning i objektfunktionen.

Derfor udvælges i hvert kald de β maskiner, der har den højeste værdi af $O'(E_p)$ (linje 7), hvor β er en konstant, der kan bruges til at styre køretiden. Herved dannes β branches. Der overføres altså kun løsningen for én maskine pr. niveau, men der undersøges β forskellige muligheder. For et problem med 6 maskiner og $\beta = 3$ vil søgetræet således have $3 \times 3 \times 3 \times 3 \times 2 \times 1$ knuder — de to sidste faktorer er < 3, idet der kun kan vælges mellem maskiner, der endnu ikke er løst, dvs. at der længst fra søgetræets rod kun er én maskine at vælge imellem.

Selvom β sættes til maksimal værdi, $|\mathcal{M}|$, da vil ikke alle mulige løsninger nødvendigvis blive betragtet. Årsagen hertil er, at vi herved ikke undersøger alle mulige rækkefølger af operationer på alle maskiner, men kun alle mulige rækkefølger af maskiner. Hver gang en maskine løses, planlægges

```
ShiftingBottleneck(E, M_0)
 1  if M_0 ≠ ∅ then
 2      for each M_p ∈ M_0 do
 3          E_p ← Solve(M_p)
 4      loop
 5      M_1 ← M_0
 6      b ← 0
 7      while b < β ∧ M_1 ≠ ∅ do
 8          q ← argmax_p(O'(E_p) : M_p ∈ M_1)
 9          E_1 ← E ∪ E_q
10          E_1 ← Reoptimize(E_1)
11          if LowerBound(E_1) ≥ UB then
12              return
13          end if
14          ShiftingBottleneck(E_1, M_0 \ {M_q})
15          M_1 = M_1 \ {M_q}
16          b ← b + 1
17      loop
18  else
19      if O(E) < UB then
20          E_best ← E
21          UB ← O(E_best)
22      end if
23  end if
```

Algoritme 5.6: Shifting bottleneck med reoptimering.

den udelukkende mht. den aktuelle løsning af det samlede problem, dvs. uden hensyntagen til endnu ikke planlagte maskiner. Dette kan betyde, at metoden i ovennævnte form i visse situationer aldrig vil finde den optimale løsning, uanset hvor længe den får lov at køre. Ej heller kan det udelukkes, at den samme delløsning undersøges flere gange.

5.4.3 Løsning af delproblemer

Hvert delproblem kan som nævnt formuleres som et almindeligt JSS-problem, der skal løses mht. de omtalte objektfunktioner. Til det formål kan man principielt benytte de samme metoder som ved løsning af generel JSS, eksempelvis branch-and-bound som beskrevet tidligere i dette kapitel. Da delproblemet er et ret simpelt specialtilfælde, findes der dog mere hensigtsmæssige metoder.

Som nævnt i afsnit 3.8 er delproblemet også $\mathcal{NP}$-hårdt. Hvor vidt løsning skal ske heuristik eller eksakt, afhænger derfor af delproblemernes størrelse, dvs. antallet af jobs i det samlede problem. Heuristisk løsning kan tage udgangspunkt i dispatch-regler, mens eksakt løsning kan ske vha. branch-and-bound.

I dette projekt løses delproblemerne til optimalitet. Vi benytter metoden beskrevet i [20][7], hvor operationerne planlægges kronologisk en ad gangen. Løsning sker vha. branch-and-bound, der styres af en dispatch-regel. Søgetræet beskæres ved hjælp af selve objektfunktionerne (5.4) og (5.5)

Et alternativ til at betragte hvert delproblem som et selvstændigt problem er blot at løse den tilsvarende del af det samlede problem. Eksempelvis kan man benytte kronologisk konfliktløsning som omtalt i afsnit 5.1.1, hvor funktionen GetEarliestConflict i algoritme 5.1 udelukkende finder konflikter på den aktuelle maskine.

Ved denne løsningsmetode bruger man således ikke objektfunktionerne nævnt i afsnit 5.4.1 men betragter i stedet blot den faktiske stigning i objektfunktionsværdien af det samlede problem mht. den oprindelige objektfunktion.

[7] I [20] er metoden fremstillet som en heuristik, men i resultatafsnittet benyttes parametre, således at delproblemerne løses til optimalitet, hvilket vi også gør her.

Dette vil potentielt kunne give en bedre grænse, idet den giver den faktiske stigning i objektfunktionen i stedet for blot den nedre grænse, der er udtrykt ved (5.4) og (5.5). Det sker dog på bekostning af en højere beregningstid, idet det samlede problem er større og har mere komplicerede datastrukturer. Denne metode er derfor ikke blevet undersøgt nærmere.

5.4.4 Reoptimering

Metoden finder en efter en den maskine, der udgør flaskehalsen, og løser den til lokal optimalitet. Ved planlægning af senere maskiner kan det imidlertid vise sig, at den løsning, der var optimal mht. først planlagte maskiner, nu ikke længere er optimal mht. den aktuelle løsning af det samlede problem.

Derfor søges efter planlægning af hver maskine at foretage en reoptimering af de allerede planlagte maskiner (linje 10). Dette sker ved for en maskine ad gangen at fjerne alle kanter mellem operationer på maskinen og så løse det tilhørende delproblem igen. Da det samlede problem har ændret sig, siden delproblemet blev løst første gang, er det således muligt, at der findes en bedre løsning mht. den aktuelle løsning af det samlede problem. Da delproblemerne løses til optimalitet, vil det ikke ske, at den nye løsning er ringere end den gamle.

Andre former for reoptimering omfatter at fjerne kanterne for flere maskiner ad gangen og så løse maskinerne igen enkeltvis i en ny rækkefølge.

5.5 Lokalsøgning

I [17] benyttedes som nævnt i afsnit 2.1.1 en lokalsøgning til at forbedre løsningsværdien, når branch-and-bound-søgningen havde fundet en brugbar løsning. Her følger en omtale af et anvendeligt nabolag.

Nabolaget findes ved at betragte de *kritiske kanter* i grafen. En kant $o_{ij} \rightarrow o_{kl}$ er kritisk, hvis en forkortelse af dens længde p_{ij} vil medføre en mindre objektfunktionsværdien[8], dvs. at i den aktuelle løsning er $S_{kl} = S_{ij} + p_{ij}$, idet $S_{ij} + p_{ij}$ udgør den maksimale værdi for de indgående kanter, jvf. (3.8).

[8] "Forkortelse" skal opfattes hypotetisk, idet kanternes længde i JSS generelt er konstant.

Ved makespan udgøres de kritiske kanter af kanterne i den længste vej gennem grafen fra kildeknuden $\mathcal{S}$ til afløbsknuden $\mathcal{V}$. Ved total weighted tardiness er de kritiske kanter foreningsmængden af kanterne i de længste veje, der ender i den sidste operation i hvert job, dvs. fra $\mathcal{S}$ til hver af afløbsknuderne $\mathcal{V}_i$. Som nævnt i afsnit 3.5.3 afhænger objektfunktionsværdien af længden af netop disse veje.

Nabolaget defineres nu som følger: givet mængden af kritiske kanter, forsøg da én kant ad gangen at vende kanten, dvs. at fjerne kanten $o_{ij} \rightarrow o_{kl}$ og tilføje en kant i den modsatte retning $o_{kl} \rightarrow o_{ij}$. Da kanter mellem to operationer på samme job er givet og således ikke kan ændres, betragtes kun kanter mellem operationer tilhørende forskellige jobs, dvs. $k \neq$i.

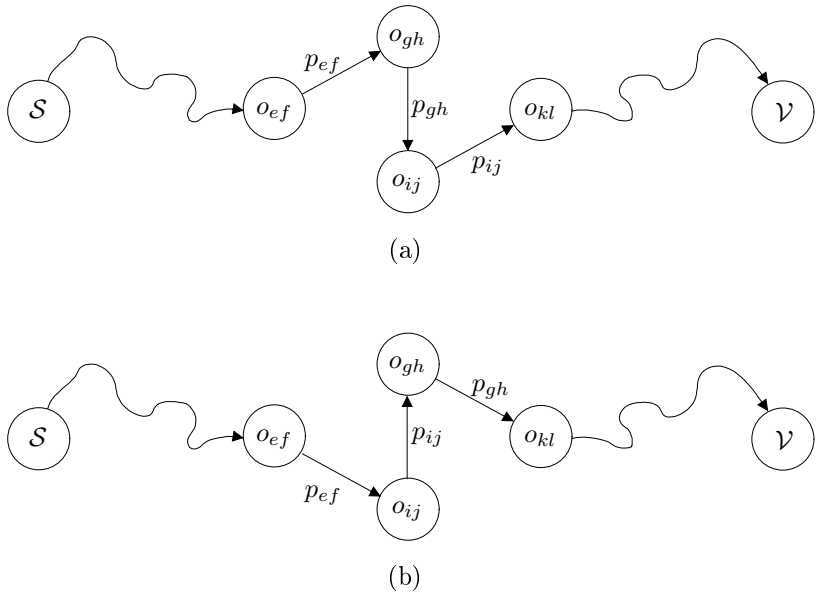
Resultatet af hver kantvending betragtes som en løsning i nabolaget. Denne løsning indeholder muligvis konflikter.

Det er ikke altid muligt at vende en given kant. Hvis der udover kanten $o_{ij} \rightarrow o_{kl}$ findes en anden vej fra o_{ij} til o_{kl}, da vil indsættelse af kanten $o_{kl} \rightarrow o_{ij}$ resultere i en ugyldig løsning. Der ses således bort fra disse løsninger.

For makespan er det endvidere kun relevant at betragte de yderste kanter i en blok bestående af tre eller flere sammenhængende kritiske kanter på samme maskine. I figur 5.7 er illustreret en situation, hvor den indre kant $o_{gh} \rightarrow o_{ij}$ vendes. Operationerne $o_{ef}, o_{gh}, o_{ij}, o_{kl}$ er alle tilknyttet sammen maskine. De ydre kanter er $o_{ef} \rightarrow o_{gh}$ og $o_{ij} \rightarrow o_{kl}$, og de skitserede veje $\mathcal{S}$ $\rightarrow o_{ef}$ og $o_{ef} \rightarrow \mathcal{V}$ angiver resten af den kritiske vej.

Da der er tale om kritiske kanter, er $S_{kl} = S_{ef} + p_{ef} + p_{gh} + p_{ij}$. Hvis orienteringen af kanten $o_{gh} \rightarrow o_{ij}$ vendes, da vil S_{kl} være udtrykt ved $S_{ef} + p_{ef} + p_{ij} + p_{gh}$, dvs. den samme sum, hvor leddene blot optræder i en anden rækkefølge. Det vil derfor ikke føre til en bedre objektfunktionsværdi at vende den indre kant.

Det samme argument vil principielt kunne anvendes ved TWT, men for at en ændring af orienteringen af en indre kant ikke skal påvirke objektfunktionsværdien, kræves her, at den samme blok af kanter er kritiske mht. alle jobs J_i. Dette vil sjældent være tilfældet, hvorfor det må antages ikke at være umagen værd at lede efter disse særtilfælde. Konsekvensen ved at vende en indre kant er jo blot en ny løsning med samme objektfunktionsværdi, som derefter forkastes af lokalsøgningen.



Figur 5.7: Ændring af orientering af indre kant i kritisk blok.

Ved lokalsøgning bevæger man sig som regel rundt blandt brugbare løsninger. Ændring af orienteringen af en kant vil som regel introducere konflikter andre steder i løsningen. Disse løses ved en hurtig heuristik, eksempelvis blot ved at løse konflikterne kronologisk vha. en dispatch-regel, således at naboløsningen også bliver brugbar.

Kapitel 6

Resultater

De i kapitel 5 omtalte metoder er implementeret og testet på en række forskellige testinstanser. I dette afsnit præsenteres resultaterne af testkørslerne, og de enkelte metoder sammenlignes med hinanden samt med tidligere rapporterede resultater fra litteraturen.

Først omtales programmellet, afviklingsmiljøet og de benyttede testinstanser. Herefter undersøges de enkelte metoder: kronologisk konfliktløsning, herunder dispatch-regler og dominanskriteriet, bounding-funktioner, branching ved kantindsættelse, lokalsøgning, shifting bottleneck samt løsning vha. CPLEX. Desuden sammenlignes med resultaterne opnået i [17].

6.1 Programmel

Programmellet er bygget i Java. Hensigten med projektforløbet har været at eksperimentere med en bred vifte af forskellige metoder snarere end at lave en effektiv implementering af en specifik algoritme. Opbygningen er således relativt modulariseret, således at algoritmer, bound-funktioner, objektfunktioner mv. let kan udskiftes.

Denne modularitet har imidlertid også ulemper. Dels er der valgt datastrukturer, der kan fungere godt med flere forskellige andre komponenter, i stedet for datastrukturer, der er tilpasset en bestemt metode. Dels kan det

ske, at de samme værdier beregnes flere gange, fordi komponenter ikke har mulighed for at interagere og dele resultater. Man må derfor forvente, at denne modularitet har en vis negativ indflydelse på programmets ydelse.

6.2 Afviklingsmiljø

Java-kørslerne er afviklet på en Sun Fire 3800 med 8×750 MHz CPU og 8 GB RAM under Solaris 8[1]. Programmerne er ikke konstrueret med henblik på parallelprocessering, men Java-afvikleren kan dog tilsyneladende drage en smule nytte af flere processorer, antageligt i forbindelse med garbage-collection. Køretider er opgivet i ren CPU-tid, dvs. at antallet af processorer ikke burde have indflydelse på resultatet.

Java-programmerne er afviklet under Java 2 Runtime, version 1.3.1 for Solaris fra Sun Microsystems.

Afviklingen er sket under Java HotSpot Server VM (parametren `-server` til `java`). I forhold til standard-VM'en (HotSpot Client VM) er denne hurtigere på bekostning af højere opstartstid. Ved længere kørsler, hvor opstartstiden har marginal betydning, har dette givet en hastighedsforøgelse i omegnen af 50%. I de rapporterede køretider er der kompenseret for opstartstiden, hvilket dog kun har betydning for kørsler med meget lille køretid.

I håb om en hastighedsforøgelse i forhold til afvikling under en virtual machine er programmerne forsøgt kompileret til (platformsafhængig) maskinkode med GCJ[2]. Dette gav dog ikke anledning til bedre køretider, hvorfor denne metode ikke er anvendt yderligere.

Løsning vha. CPLEX er foretaget på en HP J7000 med 4×440Mhz og 4 GB RAM under HP-UX 11i[3].

[1] sunfire.imm.dtu.dk

[2] GCJ er en del af GNU Compiler Collection

[3] serv2.imm.dtu.dk

6.3 Testinstanser

Der er ved testkørslerne anvendt to samlinger af instanser: instanser fra OR-Library [6] samt instanserne benyttet i [17]. Sidstnævnte er en lettere bearbejdet version af datasættet benyttet i Andrew Higgens: "Optimisation of Train Schedules to Minimise Transit Time and Maximise Reliability" og vil derfor blive betegnet Higgins-instanser[4].

Instanserne fra OR-Library stammer fra forskellige artikler om JSS. Fælles for dem alle gælder, at alle jobs har tidligste starttidspunkt $r_i = 0$.

Instanserne benævnt CAR er flow-shop-instanser, jvf. omtalen heraf i afsnit 4.3. Derudover har de behandlingstider p_{ij}, der tilsyneladende er ligeligt fordelt i intervallet $[0; 1000]$, hvilket er noget større end de øvrige instanser, hvor behandlingstiderne typisk fordeler sig i intervallet $[0; 100]$. Denne størrelsesforskel bør dog ikke have noget at skulle have sagt mht. løsningstiden, idet der blot er tale om en skalering, hvilket ikke har betydning for de benyttede algoritmer.

Higgins-instanserne baserer sig oprindeligt på autentiske data for tog. Modsat instanserne fra OR-Library er jobbenes tidligste starttidspunkter fordelt jævnt over en periode. Fordelingen er relativt tynd, dvs. at der ved kronologisk konfliktløsning er relativt få konflikter, der skal løses.

De fleste Higgins-instanser er ret små, og i [17] rapporteres også om køretider på < 1 sekund. Derfor er de mindste udeladt ved de fleste kørsler.

Testinstanserne rummer ikke deadlines d_i og vægte w_i, så for løsning mht. TWT skal vi vælge nogle værdier heraf.

Deadline fastsættes som i [21]:

$$d_i = r_i + \left\lfloor f \sum_j p_{ij} \right\rfloor$$

hvor f er en konstant, der afgør, hvor "stramme" deadlines skal være. Der er udført eksperimenter for TWT, hvor f antager værdierne 1,3 og 1,6. Disse vil blive betegnet hhv. TWT(1,3) og TWT(1,6). Desuden er der — som i [17] — undersøgt total tardiness (TT), hvor $f = 1$.

[4] Instanserne er venligst stillet til rådighed af Elias Oliveira.

Problem(er)	Antal jobs	Antal maskiner
MT06	6	6
CAR5	10	6
CAR6	8	9
CAR7	7	7
CAR8	8	8
LA01–05	10	5
H20–29	7	5
H40–46	9	5
H50–51	25	7
H60	30	16

Tabel 6.1: Størrelser af de anvendte instanser.

Vægtene bestemmes vha. en variant af metoden benyttet i [21]. De to første jobs tildeles vægten 4, de to sidste vægten 1, og resten tildeles vægten 2, dvs. at $w_1, w_2, w_3, \ldots, w_{n-2}, w_{n-1}, w_n$ er hhv. $4, 4, 2, \ldots, 2, 1, 1$[5]. Begrundelsen er, at man angiveligt i praksis ofte oplever problemer, hvor $2 \times 20\%$ af jobbene har høj hhv. lav prioritet, mens hovedparten har middel prioritet. Disse procentsatser passer nogenlunde med den nævnte vægtning ved de problemstørrelser, der undersøges her.

Af praktiske hensyn har det været ønskeligt at benytte nogle testinstanser, der kan løses inden for rimelig tid. Derfor benyttes "beskårne" udgaver af nogle af instanserne fra OR-Library. Således består instansen MT10-8x9 af de første 8 jobs fra MT10, hvor operationer på maskiner ud over de 9 første er fjernet.

I tabel 6.1 findes en oversigt over størrelserne af de anvendte probleminstanser målt i antal jobs n og antal maskiner m. Størrelserne af de beskårne probleminstanser følger direkte af navnet og er derfor ikke anført.

[5] Bemærk at i [21] behandles generel JSS, dvs. der opereres ikke med nogen rækkefølge af maskinerne. Her refereres således blot til den rækkefølge, hvori maskinerne er nævnt i datafilen.

6.4 Generelt om resultater

I det følgende præsenteres resultaterne af kørsler af et udvalg af testinstanserne. Resultaterne er primært opgjort i køretid eller antal knuder i søgetræet ved branch-and-bound.

Alle køretider er opgivet i sekunder. Hvor et resultat mangler og er erstattet med "–", skyldes det — medmindre andet er nævnt — at kørslen ikke blev afsluttet inden for 10 timer.

Køretider er i tabellerne normalt afrundet til hele sekunder, undertiden endda til hele hundredetusinder. Ved eventuelle beregnede kolonner bruges dog den oprindelige måling.

Meget tæt på tidspunktet for aflevering af denne rapport blev der konstateret en implementeringsfejl i bounding-funktionen omtalt i afsnit 5.3.2. Fejlen betød, at de beregnede bounds i visse tilfælde var mindre end værdien af den optimale løsning af en given delløsning, og derved blev dele af søgetræet beskåret på et forkert grundlag. Fejlen kom til udtryk, når der var flere operationer i R, hvis deadline ikke var nået. Den havde således størst betydning for problemer med sene deadlines, dvs. primært TWT(1,6). TWT(1,3) og TT er også berørt, men makespan er ikke.

Fejlen blev opdaget så sent, at kun få kørsler kunne nå at blive kørt om. Stikprøver viste, at køretid og antal af knuder i søgetræet ved branch-and-bound generelt er noget højere efter fejlrettelsen. Tallenes indbyrdes størrelsesforhold ser dog ud til at være det samme, og det gælder således også i vid udstrækning de konklusioner, der bringes i dette kapitel. I særdeleshed kan nævnes, at shifting bottleneck ikke ser ud til at give bedre resultater med den rettede version.

De eneste kørsler, der er kørt om efter rettelsen, er testen af bounding-funktionerne, der omtales i afsnit 6.6. De øvrige præsenterede resultater er således resultatet af kørsler med den fejlbehæftede version.

6.5 Dispatch-regler

Vi ønsker at vælge en af de i afsnit 5.1.1 omtalte dispatch-regler til at guide branch-and-bound-søgningen ved kronologisk konfliktløsning. Til det formål har vi undersøgt antallet af knuder i søgetræet ved anvendelse af

forskellige dispatch-regler på et udvalg af testinstanser. Målet er at vælge én regel, som vi bruger til senere forsøg med kronologisk konfliktløsning.

Da alle dispatch-regler er meget simple, antages det, at beregningstiden for selve dispatch-reglen er uden betydning, hvorfor antallet af besøgte knuder anvendes som mål for reglernes effektivitet.

De betragtede regler er: mindste ventetid (MV), mindste vægtede ventetid (MVV), mindste makespan (MM), først-til-mølle (FTM) og mindste behandlingstid (MB).

Tabel 6.2–6.5 viser resultatet af kørslerne for hhv. makespan, TWT(1,3), TWT(1,6) og TT. De nederste rækker i tabellerne viser, hvor mange gange hver dispatch-regel har været tæt på at resultere i det mindste antal knuder for et givet problem.

Eksempelvis betyder tallet 15 ud for 110% under MV i tabel 6.2, at i 15 af de 24 undersøgte instanser lå antallet af knuder ved brug af MV inden for 110% af det mindste antal knuder for instansen. En af disse instanser er eksempelvis H50, hvor MV resulterede i 29 knuder. Det mindste antal var 28, som blev opnået ved FTM. 29 ligger her inden for $28 \times 110\% = 30{,}8$.

Generelt gælder, at der er stor varians i antallet af knuder. Mens en given regel ved én instans er mange gange bedre end de andre, så kan den ved en anden instans være mange gange dårligere. Ved makespan ligger antallet af knuder ved brug af de forskellige regler for en given instans dog tættere end ved TWT og TT.

For makespan resulterer MV flest gange i færrest knuder, og antallet ligger generelt relativt tæt på det mindste antal. En bemærkelsesværdig undtagelse er dog kørslen af ABZ9-20x6, som ved MM blev løst i løbet af to minutter, mens de øvrige regler ikke fandt noget resultat i inden for 10 timer. Til gengæld resulterer brug af MM ved løsning af CAR6 i over 20 gange så mange knuder som den næstbedste regel.

Vi vælger derfor at benytte MV ved kronologisk konfliktløsning mht. makespan.

Ved TWT(1,3) klarer MVV i hovedparten af tilfældene sig væsentligt bedre end de øvrige. MV er næstbedst og kommer sjældent langt fra det mindste antal, men dog heller ikke så tæt på som MVV. Ved TWT vælger vi derfor at bruge MVV.

	Antal knuder			
Problem	MV	MM	FTM	MB
H20	290	67	103	362
H50	29	36	28	44
H51	34	36	36	35
H60	61	57	61	57
ABZ5-8x8	6 124	4 316	4 537	10 193
ABZ9-20x6	–	78 190	–	–
CAR5	$2{,}3 \cdot 10^6$	$5{,}5 \cdot 10^6$	$3{,}1 \cdot 10^6$	$2{,}3 \cdot 10^6$
CAR6	54 699	1 640 149	79 704	50 233
CAR7	9 528	24 226	19 635	12 270
CAR8	97 208	88 686	73 609	103 879
LA01	124	553	1 923	542
LA02	44 751	36 095	19 160	129 193
LA03	27 957	144 364	100 396	30 267
LA04	9 260	8 734	9 888	12 581
LA05	63	118	69	263
MT06	121	37	101	153
MT10-6x10	450	454	401	570
MT10-10x8	$12{,}6 \cdot 10^6$	–	$13{,}2 \cdot 10^6$	$12{,}8 \cdot 10^6$
MT10-8x10	41 092	122 014	101 508	43 218
MT10-8x8	38 106	36 745	122 980	32 463
MT10-9x9	$1{,}2 \cdot 10^6$	$0{,}8 \cdot 10^6$	$1{,}2 \cdot 10^6$	$1{,}1 \cdot 10^6$
ORB1-8x8	71 705	75 227	93 670	80 191
ORB2-8x8	2 289	4 218	11 036	4 057
ORB2-9x9	103 362	135 904	199 925	121 009
$< 110\%$	15	9	9	8
$< 200\%$	20	17	16	17
$< 500\%$	23	21	22	21
$< 1000\%$	23	22	22	23

Tabel 6.2: Løsning af 24 instanser med forskellige dispatch-regler mht. makespan.

	Antal knuder				
Problem	MV	MVV	MM	FTM	MB
H20	1 098	446	4 626	580	4 415
H50	77	114	355	28	1 696
H51	42	42	1 202	52	405
H60	61	61	59	61	537
ABZ5-8x8	97 249	19 368	108 013	111 756	191 948
CAR5	51 487	43 803	1 627 553	767 859	88 236
CAR6	77 054	63 042	860 169	151 417	86 698
CAR7	9 232	8 854	84 488	20 420	12 219
CAR8	2 654	5 633	49 026	11 412	10 700
LA01	$2{,}4 \cdot 10^6$	$2{,}6 \cdot 10^6$	$6{,}2 \cdot 10^6$	$3{,}9 \cdot 10^6$	$3{,}2 \cdot 10^6$
LA02	$10{,}0 \cdot 10^6$	$7{,}2 \cdot 10^6$	$10{,}1 \cdot 10^6$	$6{,}2 \cdot 10^6$	$11{,}8 \cdot 10^6$
LA03	$1{,}5 \cdot 10^6$	$2{,}3 \cdot 10^6$	$3{,}4 \cdot 10^6$	$1{,}6 \cdot 10^6$	$1{,}6 \cdot 10^6$
LA04	815 333	539 774	1 418 261	1 521 974	851 506
LA05	$10{,}7 \cdot 10^6$	$10{,}3 \cdot 10^6$	$11{,}3 \cdot 10^6$	$10{,}5 \cdot 10^6$	$12{,}2 \cdot 10^6$
MT06	177	134	1 690	3 124	314
MT10-6x10	9 573	7 408	16 040	18 139	8 341
MT10-10x8	$24{,}0 \cdot 10^6$	$30{,}9 \cdot 10^6$	–	$101{,}3 \cdot 10^6$	$24{,}0 \cdot 10^6$
MT10-8x10	720 718	719 208	3 909 826	1 963 492	687 556
MT10-8x8	94 848	90 445	419 795	304 106	73 422
MT10-9x9	$6{,}3 \cdot 10^6$	$5{,}4 \cdot 10^6$	$27{,}1 \cdot 10^6$	$21{,}4 \cdot 10^6$	$6{,}4 \cdot 10^6$
ORB1-8x8	42 350	36 413	149 170	105 718	81 034
ORB2-8x8	92 894	70 023	297 178	244 503	108 345
ORB2-9x9	$1{,}8 \cdot 10^6$	$1{,}8 \cdot 10^6$	$4{,}2 \cdot 10^6$	$6{,}7 \cdot 10^6$	$2{,}0 \cdot 10^6$
$< 110\%$	10	17	2	5	4
$< 200\%$	20	21	3	8	14
$< 500\%$	22	23	11	20	18
$< 1000\%$	23	23	15	21	22

Tabel 6.3: Løsning af 23 instanser med forskellige dispatch-regler mht. TWT(1,3).

	Antal knuder				
Problem	MV	MVV	MM	FTM	MB
H20	459	304	466	281	1 219
H50	29	34	76	28	225
H51	33	33	108	38	34
H60	61	61	57	61	59
ABZ5-8x8	86 390	1 273	11 706	18 057	140 356
CAR5	107 817	74 973	1 945 091	939 175	141 411
CAR6	157 554	128 063	1 856 413	464 176	157 321
CAR7	38 219	40 408	294 216	131 839	44 678
CAR8	27 235	40 486	74 881	75 232	39 585
LA01	785 702	315 064	6 293 508	5 252 795	756 914
LA02	$57{,}4 \cdot 10^6$	$52{,}2 \cdot 10^6$	$61{,}4 \cdot 10^6$	$51{,}8 \cdot 10^6$	$67{,}0 \cdot 10^6$
LA03	$1{,}9 \cdot 10^6$	$1{,}9 \cdot 10^6$	$2{,}6 \cdot 10^6$	$1{,}6 \cdot 10^6$	$1{,}8 \cdot 10^6$
LA04	$3{,}4 \cdot 10^6$	$2{,}8 \cdot 10^6$	$3{,}5 \cdot 10^6$	$4{,}6 \cdot 10^6$	$3{,}8 \cdot 10^6$
LA05	$10{,}6 \cdot 10^6$	$12{,}8 \cdot 10^6$	$11{,}7 \cdot 10^6$	$20{,}7 \cdot 10^6$	$11{,}5 \cdot 10^6$
MT06	591	771	924	2 559	767
MT10-6x10	76	102	5 020	2 469	557
MT10-8x10	61	10 436	1 870 370	710 206	2 866
MT10-8x8	185 044	157 913	499 249	636 193	126 171
MT10-9x9	$14{,}4 \cdot 10^6$	$30{,}3 \cdot 10^6$	$72{,}2 \cdot 10^6$	$25{,}0 \cdot 10^6$	$11{,}4 \cdot 10^6$
ORB1-8x8	114 579	84 714	270 047	135 658	192 625
ORB2-8x8	183 211	342 909	283 378	267 165	336 455
ORB2-9x9	$7{,}8 \cdot 10^6$	$8{,}0 \cdot 10^6$	$1{,}1 \cdot 10^6$	$4{,}3 \cdot 10^6$	$9{,}4 \cdot 10^6$
$< 110\%$	10	11	3	5	6
$< 200\%$	19	19	9	10	14
$< 500\%$	20	20	14	16	17
$< 1000\%$	21	21	17	17	20

Tabel 6.4: Løsning af 22 instanser med forskellige dispatch-regler mht. TWT(1,6).

	Antal knuder			
Problem	MV	MM	FTM	MB
H20	41	1 456	251	96
H50	5 157	15 197	4 702	51 338
H51	4 619	34 265	6 139	18 888
H60	–	–	–	–
ABZ5-8x8	82 254	132 278	150 536	103 949
CAR5	340 216	3 054 000	759 093	372 488
CAR6	79 881	3 064 865	163 146	97 821
CAR7	8 543	166 016	19 692	11 017
CAR8	17 179	102 667	38 071	46 809
LA01	$1{,}2 \cdot 10^6$	$13{,}2 \cdot 10^6$	$3{,}1 \cdot 10^6$	$1{,}3 \cdot 10^6$
LA02	$10{,}1 \cdot 10^6$	$11{,}1 \cdot 10^6$	$10{,}2 \cdot 10^6$	$10{,}7 \cdot 10^6$
LA03	345 662	2 051 160	407 415	373 632
LA04	$2{,}5 \cdot 10^6$	$5{,}1 \cdot 10^6$	$6{,}4 \cdot 10^6$	$2{,}3 \cdot 10^6$
LA05	$17{,}7 \cdot 10^6$	$18{,}1 \cdot 10^6$	$13{,}1 \cdot 10^6$	$20{,}3 \cdot 10^6$
MT06	411	2 121	3 417	554
MT10-6x10	6 867	21 934	27 316	5 549
MT10-10x8	14 393 384	–	–	9 911 936
MT10-8x10	504 211	7 035 337	3 286 185	455 543
MT10-8x8	153 691	1 555 906	1 123 679	72 121
MT10-9x9	$3{,}5 \cdot 10^6$	$72{,}6 \cdot 10^6$	$24{,}1 \cdot 10^6$	$1{,}9 \cdot 10^6$
ORB1-8x8	71 817	185 339	167 718	175 568
ORB2-8x8	77 062	391 322	226 273	97 928
ORB2-9x9	$2{,}2 \cdot 10^6$	$3{,}6 \cdot 10^6$	$2{,}7 \cdot 10^6$	$2{,}8 \cdot 10^6$
$< 110\%$	16	1	3	9
$< 200\%$	21	4	7	17
$< 500\%$	22	8	16	21
$< 1000\%$	22	14	19	21

Tabel 6.5: Løsning af 23 instanser med forskellige dispatch-regler mht. TT.

Også ved TWT(1,6) er det MV og MVV, der klarer sig bedst. Her klarer de to sig dog samlet set omtrent lige godt, selvom der er store udsving, hvis instanserne betragtes enkeltvis. Hvis vi skal vælge én regel, så må det dog blive MVV på baggrund af et marginalt bedre resultat.

MV og MVV er jo nært beslægtede, og i den forbindelse er det værd at bemærke, at det tilsyneladende ikke er en ubetinget fordel af tage vægten i betragtning — og for TWT(1,6) endda i endnu mindre grad end for TWT(1,3).

Ved TT er tendensen derimod mere klar. Her er MV klart bedst, mens MB er næstbedst uden dog at klare sig nær så godt som MV.

Hvis jobbenes vægt w_i antages at være 1 for makespan og TT, da er reglerne MV og MVV ækvivalente for disse objektfunktioner. Reglen MVV klarer sig altså bedst ved alle fire undersøgte objektfunktioner, på trods af deres forskellighed.

Noget af forklaringen skal måske søges i, at når dispatch-reglerne bliver kaldt af den kronologiske konfliktløsning, da er der meget lidt struktur på løsningen fremad, og det er derfor svært at sige noget særligt præcist om konsekvenserne af at løse konflikten på den ene eller den anden måde. Derfor virker det umiddelbart fornuftigt at gøre som MVV, nemlig at sørge for at afvikle de to involverede operationer på en sådan måde, at den samlede ventetid bliver mindst mulig.

6.6 Bounding

I afsnit 5.3 omtalte vi en bound-funktion for makespan, og vi introducerede en ny metode for TWT, og i afsnit 5.1.2 nævnte vi, hvordan søgetræet kan beskæres vha. kantindsættelse. Vi ønsker nu at undersøge disse metoders effektivitet.

Bound-funktionerne undersøges ved at sammenligne med kørsler, hvor vi som grænse blot benytter objektfunktionsværdien af den ikke-brugbare løsning. De indgående størrelser heri tager udgangspunkt i estimater for hoved og hale S_{ij}, q_{ij}, der beregnes vha. længste vej-metoden nævnt i 3.7.1.

Alle de her viste resultater tager udgangspunkt i kronologisk konfliktløsning. Tabel 6.6 viser antal knuder i søgetræet samt køretider med og uden

brug af bound-funktion samt tillige med og uden brug af kantindsættelse til beskæring af søgetræet. Tabellens anden og tredje række, der er mærket hhv. "Kantinds." og "Bound", gælder for hele kolonnen. Således er første kolonne indeholdende tal resultatet er kørslen uden kantindsættelse og uden bound.

Det ses, at de to metoder til beskæring af søgetræet begge er effektive og desuden supplerer hinanden godt, forstået på den måde, at brug af begge metoder samtidig beskærer mange flere knuder end ved brug af dem hver for sig — typisk en faktor 10.

Målt i antal knuder kan der være stor forskel på effekten af reglerne hver for sig for en given instans, men samlet set klarer reglerne sig nogenlunde lige godt. Betragter man derimod køretiderne, ses det, at kantindsættelse er væsentligt mere beregningstung end bounding-funktionen. Hvis de to metoder for en given instans beskærer det samme antal knuder, da bruger kantindsættelse omkring 15 gange så lang tid på dette.

Brug af begge regler samtidig er — målt i køretid — ikke nogen ubetinget succes. I ca. en tredjedel af tilfældene er køretiden ved brug af begge regler større end ved brug af kun bounding-funktionen. I denne rapport er der dog i de øvrige forsøg med makespan anvendt begge metoder.

For TT og TWT benyttes ikke kantindsættelse. For at benytte denne skal der nemlig regnes med haler q_{ij}^k, hvilket ellers ikke er nødvendigt. Derudover er selve kantindsættelsen er også endnu mere beregningstung, idet der for hvert par af operationer i forhold til makespan tillige skal summeres over halen til hvert job. Disse to ting tilsammen gør, at brug af kantindsættelse for TT og TWT forlænger køretiden betragteligt, hvorfor dette ikke er undersøgt nærmere.

Resultaterne af kørsler med og uden bounding fremgår af tabel 6.7–6.9. Heraf fremgår det, at brug af bound-funktion giver en reduktion i antal knuder i størrelsesorden 10–50%. For enkelte instanser reduceres køretiden også, men generelt sker en forøgelse — i flere tilfælde op mod en fordobling. Det kan således generelt ikke betale sig at benytte den omtalte bounding-funktion i forbindelse med TT og TWT.

Før rettelsen af implementationsfejlen nævnt i afsnit 6.4 resulterede brug af bounding-funktionen i en væsentlig reduktion af såvel antal knuder som køretid. Derfor er funktionen brugt i de øvrige kørsler, hvor bounding er relevant. Såfremt tiden inden rapportafleveringen tillod det, burde disse

Problem	Antal knuder				Køretid			
– Kantinds.	Uden	Med	Uden	Med	Uden	Med	Uden	Med
– Bound	Uden	Uden	Med	Med	Uden	Uden	Med	Med
H20	126 991	2 519	1 976	290	17	3	1	1
H50	29	29	29	29	1	1	1	1
H51	34	34	34	34	1	1	1	1
H60	61	61	61	61	1	2	1	2
ABZ5-8x8	5 786 915	141 742	33 409	6 124	846	151	9	10
CAR5	–	–	55 477 089	2 333 468	35 941	>10 t	12 568	4 792
CAR6	–	492 366	504 206	54 699	35 847	1 859	138	102
CAR7	26 150 204	37 826	93 438	9 528	3 474	77	19	14
CAR8	–	3 522 488	293 148	97 208	35 903	11 780	74	150
LA01	–	–	339	124	>10 t	>10 t	1	1
LA02	–	–	434 522	44 751	>10 t	>10 t	85	60
LA03	–	–	240 995	27 957	35 846	35 952	53	43
LA04	–	3 979 344	187 121	9 260	35 910	10 812	32	13
LA05	–	–	67	63	35 934	35 934	<1	1
MT06	22 813	1 298	408	121	4	2	1	1
MT10-6x10	298 704	483	22 201	450	48	3	6	3
MT10-10x8	–	–	46 306 793	12 589 492	35 954	35 954	14 752	28 191
MT10-8x10	–	104 892	2 345 797	41 092	35 904	501	701	131
MT10-8x8	–	483 416	339 048	38 106	35 851	1 445	84	68
MT10-9x9	–	–	9 530 258	1 197 222	>10 t	>10 t	3 036	3 546
ORB1-8x8	–	6 032 422	256 185	71 705	>10 t	18 173	63	112
ORB2-8x8	109 472 326	308 449	27 931	2 289	17 498	862	7	5
ORB2-9x9	–	1 403 480	2 126 791	103 362	35 905	7 036	524	243

Tabel 6.6: Effekten af bounding og kantindsættelse ved makespan.

	Antal knuder		Køretid	
Problem	Uden	Med	Uden	Med
H20	2 856	2 088	1	2
H50	170	170	<1	1
H51	42	42	<1	1
H60	61	61	1	1
ABZ5-8x8	407 169	286 050	32	66
CAR5	19 152 593	845 988	1 985	222
CAR6	985 299	776 996	87	210
CAR7	130 691	106 246	10	21
CAR8	170 940	100 841	17	25
LA03	–	118 176 902	>10 t	23 744
LA04	24 561 965	8 342 014	1 598	1 628
MT06	4 576	3 163	2	2
MT10-6x10	44 395	41 284	5	11
MT10-8x10	24 036 058	19 517 500	2 249	5 385
MT10-8x8	5 172 174	3 931 890	411	933
ORB1-8x8	642 341	314 015	61	77
ORB2-8x8	6 156 231	3 497 796	481	831
ORB2-9x9	162 887 620	91 770 946	14 770	25 141

Tabel 6.7: Effekten af bounding ved TWT(1,3).

kørsler være foretaget uden bounding. Som nævnt vurderes det dog, at de øvrige konklusioner i vidt omfang stadig er gyldige.

	Antal knuder		Køretid	
Problem	Uden	Med	Uden	Med
H20	543	408	1	1
H50	34	34	<1	<1
H51	33	33	<1	<1
H60	61	61	1	1
ABZ5-8x8	3 361	3 079	1	2
CAR5	61 557 898	4 601 637	5 866	1 148
CAR6	4 401 464	3 891 519	393	993
CAR7	944 657	882 583	62	157
CAR8	2 820 680	2 067 648	228	462
LA01	–	–	>10 t	>10 t
LA02	–	–	>10 t	>10 t
LA03	–	–	>10 t	>10 t
LA04	104 822 274	43 234 337	6 708	8 232
LA05	–	–	>10 t	>10 t
MT06	16 287	14 718	2	3
MT10-6x10	110	110	<1	1
MT10-8x10	75 709	74 909	7	20
MT10-8x8	13 106 218	12 588 396	920	2 773
MT10-9x9	–	–	>10 t	>10 t
ORB1-8x8	1 750 671	1 376 839	159	328
ORB2-8x8	14 340 164	12 698 727	1 004	2 907
ORB2-9x9	–	–	>10 t	>10 t

Tabel 6.8: Effekten af bounding ved TWT(1,6).

	Antal knuder		Køretid	
Problem	Uden	Med	Uden	Med
H20	141	71	<1	<1
H50	22 110	12 363	3	7
H51	28 492	15 904	4	9
ABZ5-8x8	3 240 338	1 717 819	234	387
CAR5	44 754 591	6 517 198	4 223	1 568
CAR6	1 440 248	978 931	135	261
CAR7	110 885	83 368	9	17
CAR8	710 967	361 745	65	89
LA03	–	60 469 040	>10 t	11 454
LA04	234 597 270	56 124 029	15 117	10 342
MT06	14 527	6 879	2	3
MT10-6x10	54 740	43 317	6	11
MT10-8x10	27 823 682	18 779 204	2 661	6 882
MT10-8x8	10 905 759	6 726 638	882	1 625
ORB1-8x8	2 337 359	1 211 837	204	290
ORB2-8x8	11 035 874	4 848 336	911	1 156

Tabel 6.9: Effekten af bounding ved TT.

6.7 Dominans

For at undersøge effekten af anvendelse af det i afsnit 5.2 omtalte dominanskriterium, er der udført en række forsøg hhv. med og uden. Effektiviteten måles ved at sammenligne antallet af besøgte knuder i søgetræet samt køretiden.

Resultaterne er vist i tabel 6.10–6.12. Kolonnen "%" angiver den procentvise forbedring ved anvendelse af dominanskriteriet, dvs. en værdi i intervallet $]-\infty; 100\%]$.

Nogle af resultaterne er mærket $\star$; dette er forklaret i afsnit 6.7.1.

Desværre viste brugen af dominanskriteriet sig ikke at være så effektivt som først antaget. Mens det som regel reducerer antallet af knuder, så medfører det også en forøgelse i køretid pr. knude.

For makespan er effekten blandet. I mange tilfælde lykkes det at reducere antallet af knuder med omkring 15%, men denne reduktion er ikke nok til at sænke køretiden. Eksempelvis reduceres antallet af knuder for MT10-8x10 med 13%, men på trods af dette er køretiden 2% højere. I hovedparten af tilfældene er køretiden i omegnen af 20% højere end uden brug af dominanskriteriet.

For TWT er tendensen mere klar. Reduktionen i antal knuder er op til ca. 5%, lidt højere for TWT(1,6) end TWT(1,3), mens køretiden i mange tilfælde fordobles.

Kigger vi på TT er billedet derimod anderledes, særligt hvis man betragter de største instanser, H50 og H51, hvor over 75% af knuderne beskæres, hvilket giver en reduktion i køretid på 50%. For H60 terminerer kørslen uden brug af snit ikke inden 10 timer, mens den med brug af snit er afsluttet på under 8 minutter.

Konklusionen på dette må være, at det for OR-Library-problemerne ikke kan betale sig at benytte dominanskriteriet, mens det for Higgins-instanserne ser mere lovende ud. Effekten af dominanskriteriet for de øvrige Higgins-instanser vil blive undersøgt i afsnit 6.10.

6.7.1 Hukommelsesforbrug

Antallet af besøgte nutider er i samme størrelsesorden som antallet af knuder. For hver nutid gemmes selve nutiden samt den tilhørende løsning. Ved

	Antal knuder			Køretid		
Problem	Uden	Med	%	Uden	Med	%
H20	290	220	24	1	1	−15
H50	29	29	0	1	1	11
H51	34	34	0	1	1	6
H60	61	61	0	2	2	4
ABZ5-8x8	6 124	6 082	1	12	13	−11
CAR5	2 333 468	1 900 860⋆	19	5 391	4 922	9
CAR6	54 699	42 786⋆	22	113	98	13
CAR7	9 528	9 269	3	14	16	−11
CAR8	97 208	93 005⋆	4	171	172	−1
LA01	124	124	0	1	1	−25
LA02	44 751	38 655	14	65	63	3
LA03	27 957	23 973	14	46	45	1
LA04	9 260	8 910	4	15	16	−7
LA05	63	63	0	1	1	3
MT06	121	121	0	1	1	−16
MT10-6x10	450	450	0	3	3	−4
MT10-10x8	12 589 492	11 029 108⋆	12	31 558	30 530	3
MT10-8x10	41 092	35 807⋆	13	146	142	2
MT10-8x8	38 106	33 307⋆	13	68	68	0
MT10-9x9	1 197 222	969 208⋆	19	3 865	3 417	12
ORB1-8x8	71 705	66 640⋆	7	112	114	−2
ORB2-8x8	2 289	2 159	6	5	6	−9
ORB2-9x9	103 362	91 226⋆	12	280	266	5

Tabel 6.10: Dominanskriterier ved makespan.

	Antal knuder			Køretid		
Problem	Uden	Med	%	Uden	Med	%
H20	446	443	1	1	1	−58
H50	114	84	26	1	1	−5
H51	42	38	10	1	1	2
H60	61	61	0	1	1	−13
ABZ5-8x8	19 368	18 908	2	7	9	−37
CAR5	43 803	43 573	1	15	21	−35
CAR6	63 042	60 898	3	22	29	−34
CAR7	8 854	8 826	0	3	5	−44
CAR8	5 633	5 626	0	3	4	−39
LA01	2 553 484	2 549 419⋆	0	562	1 159	−106
LA02	7 173 640	7 089 085⋆	1	1 679	3 342	−99
LA03	2 268 229	2 222 782⋆	2	493	948	−92
LA04	539 774	522 420⋆	3	118	215	−82
LA05	10 297 068	9 925 063⋆	4	2 253	4 386	−95
MT06	134	129	4	<1	1	−24
MT10-6x10	7 408	7 205	3	4	5	−42
MT10-10x8	30 883 211	30 812 977⋆	0	10 916	15 791	−45
MT10-8x10	719 208	717 159⋆	0	248	337	−36
MT10-8x8	90 445	89 647	1	26	37	−45
MT10-9x9	5 425 843	5 419 608⋆	0	1 957	2 789	−43
ORB1-8x8	36 413	36 040	1	12	17	−42
ORB2-8x8	70 023	69 428	1	20	29	−48
ORB2-9x9	1 754 610	1 751 975⋆	0	570	829	−45

Tabel 6.11: Dominanskriterier ved TWT(1,3).

	Antal knuder			Køretid		
Problem	Uden	Med	%	Uden	Med	%
H20	304	303	0	1	1	−20
H50	34	34	0	<1	1	−37
H51	33	33	0	<1	1	−38
H60	61	61	0	1	1	11
ABZ5-8x8	1 273	1 223	4	1	2	−31
CAR5	74 973	74 535	1	24	31	−29
CAR6	128 063	123 791⋆	3	42	54	−30
CAR7	40 408	40 288	0	10	14	−42
CAR8	40 486	40 462	0	13	18	−36
LA01	315 064	311 759⋆	1	68	129	−89
LA02	52 195 377	52 068 845⋆	0	11 083	23 962	−116
LA03	1 880 421	1 865 264⋆	1	384	754	−96
LA04	2 756 512	2 620 623⋆	5	588	1 037	−76
LA05	12 787 409	12 518 295⋆	2	2 618	5 508	−110
MT06	771	761	1	1	2	−67
MT10-6x10	102	102	0	1	1	4
MT10-8x10	10 436	10 427	0	5	7	−55
MT10-8x8	157 913	155 306⋆	2	42	56	−33
MT10-9x9	30 288 654	30 267 847⋆	0	10 078	14 182	−41
ORB1-8x8	84 714	84 279	1	26	35	−34
ORB2-8x8	342 909	340 142⋆	1	85	126	−48
ORB2-9x9	8 018 619	8 005 471⋆	0	2 451	3 612	−47

Tabel 6.12: Dominanskriterier ved TWT(1,6).

	Antal knuder			Køretid		
Problem	Uden	Med	%	Uden	Med	%
H20	41	36	12	<1	<1	58
H50	5 157	1 235	76	4	2	52
H51	4 619	992	79	4	2	51
H60	–	435 597⋆	–	>10 t	442	–
ABZ5-8x8	82 254	80 159	3	23	31	−38
CAR5	340 216	339 428⋆	0	96	166	−73
CAR6	79 881	78 745	1	26	34	−29
CAR7	8 543	8 523	0	3	5	−58
CAR8	17 179	17 103	0	7	9	−35
LA01	1 178 097	1 172 627⋆	0	257	570	−121
LA02	10 084 989	10 045 477⋆	0	2 168	4 706	−117
LA03	345 662	341 990⋆	1	81	159	−98
LA04	2 514 488	2 474 817⋆	2	530	1 098	−107
LA05	17 707 825	16 581 270⋆	6	3 713	8 052	−117
MT06	411	405	1	1	1	−43
MT10-6x10	6 867	5 669	17	3	4	−30
MT10-10x8	14 393 384	14 368 215⋆	0	4 908	7 457	−52
MT10-8x10	504 211	503 641⋆	0	169	234	−38
MT10-8x8	153 691	151 896⋆	1	42	56	−36
MT10-9x9	3 474 216	3 470 759⋆	0	1 151	1 666	−45
ORB1-8x8	71 817	70 966	1	22	29	−33
ORB2-8x8	77 062	76 134	1	20	28	−38
ORB2-9x9	2 246 381	2 241 523⋆	0	713	1 059	−48

Tabel 6.13: Dominanskriterier ved TT.

løsning af større instanser kan hukommelsesforbruget således udgøre et problem.

En kritisk faktor for antallet af lagrede nutider er selvfølgelig, hvor kompakt man repræsenterer nutider og løsninger. Men man skal ikke bruge ret mange bytes pr. styk, før problemer med millioner af knuder i søgetræet bliver et problem. Som det fremgår af tabellerne, er dette tal ikke urealistisk.

Derfor lagres værdierne her vha. Javas *soft references*. Modsat med almindelige referencer vil de refererede objekter, dvs. nutiderne, blive fjernet af garbage collectoren *før* Java evt. terminerer pga. mangel på hukommelse. Der ydes ingen garanti mht. hvilke objekter, der slettes først, men Java-afvikleren tilstræber ikke at slette nyligt oprettede eller anvendte objekter. I tabellerne er det markeret med en stjerne $\star$, hvor der af hukommelseshensyn er slettet fra listen af kendte nutider.

Ved disse kørsler lagres selve løsningen slet ikke i algoritme 5.3 linje 26 men kun løsningsværdien. Dette betyder, at algoritmen ikke nødvendigvis finder selve den optimale løsning men kun værdien heraf. Kørselsforløbet — og således antallet af knuder i søgetræet — er dog det samme, så målingerne er gyldige. Programmet vil uden videre kunne ændres til også at finde selve løsningen alene på bekostning af et betydeligt større hukommelsesforbrug. Dette vil dog medføre, at der kan lagres endnu færre nutider med deraf følgende ringere ydelse. Dette betyder, at effekten af brug af snit i praksis bliver endnu ringere, end det er angivet i tabellerne.

6.8 Sammenligning af branch-and-bound-metoder

Vi har nu fået valgt en dispatch-regel samt fået fastslået, at det generelt ikke kan betale sig at benytte det omtalte dominanskriterium. Vi har således bestemt parametrene til brug ved kronologisk konfliktløsning (KK), og vi kan nu sammenligne det med den anden branch-and-bound-metode, vi har omtalt, nemlig branching ved kantindsættelse (BK).

Da metoderne er meget forskellige, er antallet af knuder i de respektive søgetræer ikke sammenligneligt, så i stedet sammenlignes køretiderne.

I tabel 6.8 vises køretiderne for løsning af en række problemer mht. makespan med hhv. KK og BK. Desuden vises køretider for [3], en C-implemen-

	Køretid			
Problem	[3]-edge	[3]-dumbo	BK	KK
H20	–	–	3	1
H50	–	–	>10 t	1
H51	–	–	>10 t	1
H60	–	–	>10 t	2
ABZ5-8x8	<1	3	27	12
CAR5	249	>10 t	5 969	5 391
CAR6	9	102	956	113
CAR7	4	14	203	14
CAR8	3	77	1 161	171
LA01	<1	1	–	1
LA02	5	6	90	65
LA03	<1	1	9	46
LA04	<1	24	77	15
LA05	<1	1	>10 t	1
MT06	<1	<1	4	1
MT10-6x10	<1	1	10	3
MT10-10x8	21	9 868	>10 t	31 558
MT10-8x10	2	16	149	146
MT10-8x8	<1	26	215	68
MT10-9x9	16	594	>10 t	3 865
ORB1-8x8	2	197	2 118	112
ORB2-8x8	<1	21	128	5
ORB2-9x9	10	102	741	280

Tabel 6.14: Køretider for to programmer fra [3] samt kronologisk konfliktløsning (KK) og branching ved kantindsættelse (BK) ved makespan.

tation af metoderne omtalt i [4] udgivet af forfatterne selv. Disse kørsler er foretaget på samme maskine.

Dumbo svarer mere eller mindre til BK, mens Edge (i artiklen kaldet *edge-finder*) er en mere raffineret udgave af samme. Bemærk at [3] ikke kan løse Higgins-instanserne, da disse ikke er begrænset til det tilfælde, hvor alle jobs tidligste starttidspunkt $r_i = 0$.

Sammenligner man KK og BK, er der ingen tvivl om, at KK er væsentligt hurtigere. BK og [3]-dumbo er implementationer af samme grundliggende algoritme, og ved implementationen af førstnævnte er der endda skelet til sidstnævntes kildetekst. Alligevel er [3]-dumbo omkring 10 gange hurtigere. Forklaringen herpå skal nok delvis søges i, at nærværende programmel som nævnt i afsnit 6.1 ikke er optimeret til nogen speciel algoritme med de deraf følgende ulemper. Derudover må man også formode, at en implementation i C kører hurtigere end en tilsvarende i Java. Forskellen i relativ køretid for hvert problem svinger meget, hvilket antyder, at BK ikke blot er en ineffektiv udgave af [3]-dumbo men decideret har andre karakteristika.

Endelig konstateres det, at [3]-edge i alle tilfælde er langt hurtigere end de øvrige. Denne metode er ikke forsøgt implementeret i dette projekt.

I tabel 6.15 vises køretiderne for løsning af en række problemer mht. TWT-(1,3), TWT(1,6) og TT. Hvor et resultat er udeladt, skyldes det, at programmet terminerede pga. mangel på hukommelse, dvs. at antallet af delløsninger i åbenlisten er for stort. Dette skete typisk efter ca. 2 timer.

Programmerne [3] kan kun håndtere objektfunktionen makespan, så vi kan her ikke sammenligne resultaterne med dem.

Vi må konstatere, at BK løb tør for hukommelse i hovedparten af tilfældene, og i de resterende var væsentligt langsommere end KK. Ved makespan skete det kun en enkelt gang, at programmet løb tør for hukommelse, nemlig ved løsningen af LA01. Det er dog muligt, at det også ville været sket ved nogle af de kørsler, der blev stoppet efter 10 timer, hvis de havde fået lov at køre videre.

Resultaterne for BK sammenlignet med KK synes endnu ringere ved TWT og TT end ved makespan. Objektfunktionerne er meget forskellige og kan således ikke direkte sammenlignes, men en del af forskellen ligger formodentlig i, at kantindsættelsen har højere kompleksitet ved TWT og TT end ved makespan som nævnt i afsnit 6.6.

	Køretid					
	TWT(1,3)		TWT(1,6)		TT	
Problem	BK	KK	BK	KK	BK	KK
H20	4	1	4	1	2	<1
H50	–	1	–	<1	–	4
H51	–	1	–	<1	–	4
H60	–	1	–	1	–	>10 t
ABZ5-8x8	1 321	7	–	1	–	23
CAR5	–	15	–	24	–	96
CAR6	–	22	–	42	–	26
CAR7	303	3	2 582	10	323	3
CAR8	–	3	–	13	–	7
LA01	–	562	–	68	–	257
LA02	–	1 679	–	11 083	–	2 168
LA03	–	493	–	384	–	81
LA04	–	118	–	588	–	530
LA05	–	2 253	–	2 618	–	3 713
MT06	5	<1	4	1	5	1
MT10-6x10	72	4	628	1	73	3
MT10-10x8	–	10 916	–	>10 t	–	4 908
MT10-8x10	–	248	383	5	–	169
MT10-8x8	–	26	1 282	42	485	42
MT10-9x9	–	1 957	–	10 078	–	1 151
ORB1-8x8	–	12	–	26	12 595	22
ORB2-8x8	3 939	20	189	85	2 258	20
ORB2-9x9	–	570	–	2 451	–	713

Tabel 6.15: Køretider for kronologisk konfliktløsning (KK) og branching ved kantindsættelse (BK).

6.9 Lokalsøgning

Vi ønsker nu at undersøge effekten af brug af lokalsøgning sammen med kronologisk konfliktløsning som nævnt i 2.1.1.

I tabel 6.16–6.19 er vist resultatet af kørsler hhv. med og uden brug af lokalsøgning. Den første kolonne, LS, angiver antallet af gange, branch-and-bound-søgningen er nået et blad i søgetræet og således har fundet en brugbar løsning. Denne løsning er ikke nødvendigvis bedre end den hidtil bedste, men det er tæt på, idet undergrænsen for objektfunktionsværdien umiddelbart før løsning af den sidste konflikt var mindre end objektfunktionsværdien af den hidtil bedste løsning — ellers ville søgetræet nemlig være blevet beskåret. Den anden kolonne, LSF, viser, hvor mange af de lokalsøgninger nævnt i LS der resulterede i en bedre løsning end den hidtil bedste.

Ved TWT og TT lykkes det i de fleste tilfælde lokalsøgningen at finde en bedre løsning et sted mellem 0 og 10 gange. Disse tal stemmer overens med resultaterne nævnt i [17]. Dette medfører en reduktion i køretid på op mod 10%.

Det ses, at den procentvise reduktion i knuder hhv. køretid er nogenlunde ens. Årsagen hertil er, at hver lokalsøgning kun tager få sekunder, dvs. at den udgør en meget lille del af køretiden for de fleste instanser. Idet køretiden for kronologisk konfliktløsning uden lokalsøgning antages at være proportional med antallet af knuder, da vil en reduktion i antal knuder pga. lokalsøgningen således give en proportional reduktion i køretid.

Ved makespan lykkedes det sjældent lokalsøgningen at finde en bedre løsning, og i så godt som alle kørsler blev den samlede køretid forøget en del ved brug af lokalsøgning.

			Antal knuder		Køretid	
Problem	LS	LSF	Uden	Med	Uden	Med
H20	10	1	290	290	1	1
H50	1	0	29	29	1	1
H51	1	1	34	33	1	1
H60	1	0	61	61	2	1
ABZ5-8x8	26	1	6 124	6 124	12	16
CAR5	16	1	2 333 468	2 256 803	5 391	6 924
CAR6	25	1	54 699	54 560	113	213
CAR7	17	0	9 528	9 502	14	27
CAR8	42	0	97 208	96 012	171	327
LA01	7	0	124	124	1	1
LA02	19	0	44 751	44 816	65	74
LA03	21	0	27 957	28 335	46	69
LA04	20	0	9 260	9 248	15	15
LA05	3	0	63	63	1	1
MT06	11	1	121	118	1	1
MT10-6x10	20	1	450	449	3	3
MT10-10x8	26	0	12 589 492	–	31 558	>10 t
MT10-8x10	23	0	41 092	41 035	146	299
MT10-8x8	18	0	38 106	38 034	68	121
MT10-9x9	34	0	1 197 222	1 195 746	3 865	7 323
ORB1-8x8	21	0	71 705	71 623	112	189
ORB2-8x8	17	0	2 289	2 262	5	6
ORB2-9x9	27	2	103 362	103 249	280	473

Tabel 6.16: Effekten af lokalsøgning ved makespan.

			Antal knuder		Køretid	
Problem	LS	LSF	Uden	Med	Uden	Med
H20	9	2	446	440	1	1
H50	5	1	114	112	1	1
H51	2	0	42	42	1	1
H60	1	0	61	61	1	1
ABZ5-8x8	11	5	19 368	19 349	7	6
CAR5	10	1	43 803	46 696	15	16
CAR6	13	2	63 042	62 882	22	21
CAR7	9	1	8 854	8 840	3	3
CAR8	9	2	5 633	5 647	3	3
LA01	35	1	2 553 484	2 553 397	562	540
LA02	20	2	7 173 640	7 173 578	1 679	1 601
LA03	21	2	2 268 229	2 268 268	493	482
LA04	12	3	539 774	539 440	118	115
LA05	5	1	10 297 068	10 216 261	2 253	2 171
MT06	3	2	134	124	<1	1
MT10-6x10	13	5	7 408	7 383	4	4
MT10-10x8	29	9	30 883 211	26 559 671	10 916	9 239
MT10-8x10	8	5	719 208	714 799	248	240
MT10-8x8	5	5	90 445	90 191	26	26
MT10-9x9	18	5	5 425 843	5 423 323	1 957	1 905
ORB1-8x8	8	4	36 413	36 391	12	12
ORB2-8x8	21	4	70 023	69 975	20	20
ORB2-9x9	15	6	1 754 610	1 713 448	570	552

Tabel 6.17: Effekten af lokalsøgning ved TWT(1,3).

			Antal knuder		Køretid	
Problem	LS	LSF	Uden	Med	Uden	Med
H20	19	2	304	290	1	1
H50	1	0	34	34	<1	1
H51	1	0	33	33	<1	1
H60	1	0	61	61	1	1
ABZ5-8x8	10	4	1 273	1 027	1	1
CAR5	20	1	74 973	86 699	24	27
CAR6	20	8	128 063	98 973	42	34
CAR7	18	4	40 408	39 790	10	10
CAR8	14	1	40 486	40 538	13	13
LA01	28	3	315 064	315 021	68	71
LA02	23	1	52 195 377	52 195 358	11 083	10 719
LA03	14	3	1 880 421	1 801 715	384	356
LA04	7	4	2 756 512	765 667	588	155
LA05	18	2	12 787 409	12 782 690	2 618	2 566
MT06	3	2	771	767	1	1
MT10-6x10	5	2	102	85	1	1
MT10-8x10	14	10	10 436	8 721	5	5
MT10-8x8	15	6	157 913	157 368	42	41
MT10-9x9	35	12	30 288 654	30 261 190	10 078	9 680
ORB1-8x8	14	6	84 714	84 527	26	26
ORB2-8x8	26	4	342 909	309 607	85	75
ORB2-9x9	17	3	8 018 619	8 019 064	2 451	2 269

Tabel 6.18: Effekten af lokalsøgning ved TWT(1,6).

			Antal knuder		Køretid	
Problem	LS	LSF	Uden	Med	Uden	Med
H20	2	0	41	41	<1	<1
H50	2	2	5 157	4 893	4	4
H51	3	3	4 619	4 315	4	4
H60	4	4	–	–	>10 t	>10 t
ABZ5-8x8	15	5	82 254	81 642	23	23
CAR5	5	0	340 216	333 325	96	95
CAR6	14	7	79 881	79 038	26	27
CAR7	7	1	8 543	8 513	3	3
CAR8	26	3	17 179	16 920	7	7
LA01	24	3	1 178 097	1 174 584	257	262
LA02	19	0	10 084 989	10 085 812	2 168	2 190
LA03	9	0	345 662	345 640	81	80
LA04	27	8	2 514 488	2 306 080	530	501
LA05	37	2	17 707 825	17 705 931	3 713	3 702
MT06	5	3	411	387	1	1
MT10-6x10	30	1	6 867	6 866	3	3
MT10-10x8	23	7	14 393 384	13 976 542	4 908	4 713
MT10-8x10	15	2	504 211	504 194	169	167
MT10-8x8	16	3	153 691	153 684	42	43
MT10-9x9	19	4	3 474 216	3 473 375	1 151	1 145
ORB1-8x8	38	12	71 817	64 864	22	20
ORB2-8x8	17	4	77 062	74 147	20	21
ORB2-9x9	23	10	2 246 381	2 239 042	713	704

Tabel 6.19: Effekten af lokalsøgning ved TT.

6.10 Løsning af Higgins-problemer

Vi har nu konstateret, at kronologisk konfliktløsning er den bedste af de to implementerede branch-and-bound-metoder, og vi ønsker nu at sammenligne denne med resultaterne rapporteret i [17], der som nævnt også bygger på kronologisk konfliktløsning. Vi anfører køretider for såvel med som uden brug af dominanskriteriet. Resultatet heraf er vist i tabel 6.10.

Det er svært at konkludere ret meget på baggrund af de små køretider, men overordnet set løser nærværende implementation af kronologisk konfliktløsning instanserne noget hurtigere. Gevinsten ved brug af dominanskriterier her ses at være betragtelig.

Som nævnt i afsnit 2.1.2 er problemet omtalt i [17] mere generelt end det, der er betragtet i denne rapport, så tallene er ikke umiddelbart sammenlignelige. Men — som nævnt i 4.2 — så er de problemer, der rent faktisk blev behandlet ved testkørslerne, meget lig den begrænsede version, der betragtes i nærværende rapport, hvorfor en sammenligning alligevel er relevant.

Som det fremgår af tabellen, er løsningsværdierne ens for alle på nær to problemer, H21 og H51. For disse to problemer er den her opnåede løsningsværdi bedre end den rapporteret i [17], dvs. at de i [17] må være løst med nogle flere begrænsninger end her. Det er dog uklart, hvad forskellen er.

I afsnit 2.1.2 er nævnt, at en række af de ekstra betingelser, der omtales i [17], tilsyneladende ikke kommer til udtryk i testinstanserne. Den umiddelbart eneste yderligere begrænsning, der ser ud til at være til stede i instanserne, er betingelsen om en maksimal længde af vigesporene[6].

Studerer man den her opnåede løsning af H21 — illustreret på figur 6.1 — vil man dog indse, at denne begrænsning ikke har relevans her, idet der aldrig holder mere end ét tog på hvert vigespor ad gangen. Tallene i grafen referer til det job, hver operation tilhører.

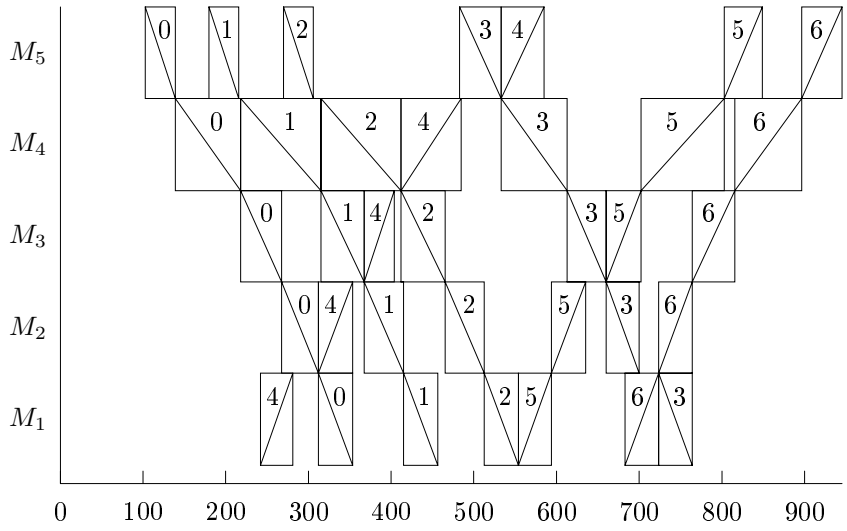
Det står således hen i det uvisse, hvilke yderligere begrænsninger løsningerne i [17] ligger under for.

Hvis man på trods af dette sammenligner køretiderne, så ses, at de her opnåede køretider generelt er bedre end dem rapporteret i [17]. Det er

[6] Dette omtales med en vis usikkerhed, idet datafilerne ikke er blevet ledsaget med en specielt udførlig forklaring på formatet heraf.

	Løsningsværdi		Køretid		
Problem	[17]	Her	[17]	Uden dom.	Med dom.
H20	1 224	1 224	5,27	0,16	0,12
H21	**275**	**174**	0,15	0,10	0,06
H22	137	137	0,15	0,10	0,08
H23	116	116	0,07	0,01	0,02
H24	349	349	0,25	0,08	0,13
H25	285	285	0,14	0,17	0,11
H26	338	338	0,15	0,10	0,05
H27	161	161	0,17	0,09	0,11
H28	249	249	0,09	0,05	0,05
H29	223	223	0,24	0,04	0,07
H40	215	215	0,16	0,17	0,06
H41	199	199	0,12	0,15	0,12
H42	388	388	0,58	0,41	0,32
H43	560	560	1,03	0,69	0,61
H44	389	389	0,26	0,46	0,23
H45	228	228	0,11	0,19	0,09
H46	526	526	0,11	0,25	0,19
H50	555	555	22,85	4,66	2,23
H51	**650**	**565**	463,78	4,06	2,06
H60	≤ 318	262	>10 t	>10 t	441,51

Tabel 6.20: Sammenligning af [17] og kronologisk konfliktløsning hhv. med og uden brug af dominanskriterier ved løsning af Higgins-problemer mht. TT. Den ved [17] anførte løsningsværdi for H60 på 318 er den bedste løsning, der blev fundet inden for 10 timer.



Figur 6.1: Løsning af H21.

svært at konkludere så meget på grundlag af de små instanser, men for de større er tendensen klar, og især ved H60 er der stor forskel.

Udover den ukendte forskel i problemformuleringerne, så må forbedringen dels tilskrives brug af dominanskriterier, hvilket kraftigt antydes ved sammenligning af de her opnåede køretider for H60. Desuden må det antages, at brug af en stærkere bound-funktion også har en vis effekt — som nævnt i afsnit 2.1.1 omtales i [17] ingen bound-funktion, hvorfor det må antages, at der blot benyttes en triviel værdi, eksempelvis objektfunktionen af den ikke-brugbare løsning.

6.11 Shifting bottleneck

Vi vil nu undersøge virkningen af heuristikken shifting bottleneck.

I afsnit 5.4.2 nævnedes parameteren β, der angiver antallet af branches pr. niveau. Vi forsøger os nu med forskellige værdier af β, samt med og uden reoptimering.

Da shifting bottleneck er en heuristik, finder den ikke nødvendigvis den optimale løsning. Derfor er såvel løsningsværdier som køretider angivet i tabel 6.21–6.24. Desuden er resultatet af løsning til optimalitet vha. kronologisk konfliktløsning vist.

For TWT(1,6) gav stigende værdier af β bedre resultater t.o.m. $\beta = 5$, mens forbedringen for de øvrige objektfunktioner stort set ophørte ved $\beta = 3$ eller før. Bemærk derfor, at der i tabel 6.23 for TWT(1,6) angives resultater for andre værdier af β end i de øvrige tabeller.

For såvel makespan som TWT gælder, at de opnåede resultater forbedres svagt som følge af højere værdier af β. Dog er de stadig relativt langt fra den optimale løsningsværdi, typisk mindst 10%. Bortset fra enkelte tilfælde synes reoptimering ikke at have nogen effekt på løsningsværdien.

I enkelte tilfælde lykkes det dog at finde den optimale løsning. Det drejer sig primært og Higgins-instanserne samt MT06. Fælles for disse er, at de har korte køretider ved løsning vha. kronologisk konfliktløsning.

Det bemærkes, at det ved TT ikke lykkedes at løse problemerne H50, H51 og H60 i løbet af 10 timer. Dette skyldes formentlig det høje antal jobs (25 stk. for H50 og H51, og 30 for H60) sammenlignet med de øvrige instanser, hvilket nok gør det meget tidskrævende at løse enkeltmaskineproblemerne til optimalitet.

I litteraturen rapporteres generelt om løsninger af bedre kvalitet [4, 20] og i særdeleshed større effekt af reoptimering og stigende værdier af β. Det er ikke lykkes at finde nogen forklaring på denne forskel, men den kan muligvis tilskrives en decideret fejl i implementationen sammenlignet med algoritmerne beskrevet i litteraturen.

Løsning af delproblemer er forsøgt med forskellige metoder såvel heuristisk som eksakt, og værdien af O' er sammenlignet med den faktiske stigning i det samlede problems objektfunktionsværdi. Der er tillige forsøgt med en branching-parameter $\beta = |\mathcal{M}|$, dvs. at algoritmen undersøger et fuldt søgetræ[7]. Begge dele var dog uden positivt resultat. Et kvalificeret bud ville være, at problemerne ligger i reoptimeringsdelen, idet denne del er undersøgt mindst.

[7] Som nævnt i afsnit 5.4.2 betyder dette ikke nødvendigvis, at alle mulige løsninger undersøges.

	Løsningsværdi					Køretid				
Problem	$\beta = 1$	$\beta = 2$	$\beta = 3$	$\beta = 3$ m. reopt.	KK	$\beta = 1$	$\beta = 2$	$\beta = 3$	$\beta = 3$ m. reopt.	KK
H20	1 057	1 057	1 057	1 057	1 013	1	1	1	1	1
H50	2 890	2 890	2 890	2 890	2 890	9	9	8	9	1
H51	3 030	3 030	3 030	3 030	3 030	9	9	9	9	1
H60	1 793	1 793	1 793	1 793	1 793	270	286	274	282	2
ABZ5-8x8	1 047	1 047	1 047	1 047	985	5	6	8	10	12
CAR5	8 077	8 077	8 077	8 077	7 702	359	374	370	383	5 391
CAR6	9 048	9 048	8 842	8 842	8 313	13	33	64	71	113
CAR7	7 129	6 749	6 636	6 636	6 558	3	5	5	6	14
CAR8	9 012	8 988	8 988	8 988	8 264	12	24	59	60	171
LA01	678	678	678	678	666	67	67	67	67	1
LA02	705	694	694	694	655	90	144	147	134	65
LA03	622	622	622	622	597	132	148	157	159	46
LA04	611	611	611	611	590	75	76	74	80	15
LA05	593	593	593	593	593	78	78	79	80	1
MT06	59	58	55	55	55	1	2	2	3	1
MT10-6x10	796	790	790	790	774	4	13	34	39	3
MT10-10x8	966	950	950	950	861	232	317	461	524	31 558
MT10-8x10	874	846	837	837	824	15	52	128	131	146
MT10-8x8	763	763	763	763	737	10	12	18	23	68
MT10-9x9	937	899	898	898	855	23	42	67	78	3 865
ORB1-8x8	852	852	845	845	787	8	15	24	26	112
ORB2-8x8	752	752	752	752	726	6	8	11	13	5
ORB2-9x9	874	874	837	837	793	18	25	59	64	280

Tabel 6.21: Shifting bottleneck ved makespan.

	Løsningsværdi					Køretid				
Problem	$\beta = 1$	$\beta = 2$	$\beta = 3$	$\beta = 3$ m. reopt.	KK	$\beta = 1$	$\beta = 2$	$\beta = 3$	$\beta = 3$ m. reopt.	KK
H20	1 712	1 692	1 639	1 639	1 253	2	1	2	3	1
H50	0	0	0	0	0	1 109	1 173	1 182	1 135	1
H51	0	0	0	0	0	31	31	31	31	1
H60	0	0	0	0	0	132	133	135	180	1
ABZ5-8x8	1 057	1 057	1 057	1 057	687	6	12	37	50	7
CAR5	29 471	29 471	29 471	29 471	26 114	446	543	841	819	15
CAR6	16 388	16 388	15 861	15 861	11 201	24	59	255	293	22
CAR7	8 967	8 913	8 913	8 913	7 785	4	9	18	22	3
CAR8	16 870	16 870	16 870	16 870	14 024	9	11	14	15	3
LA01	2 504	2 504	2 504	2 504	2 072	155	196	244	238	562
LA02	2 579	2 579	2 579	2 579	2 080	191	228	296	269	1 679
LA03	2 637	2 460	2 460	2 460	2 232	201	289	424	427	493
LA04	2 923	2 490	2 490	2 490	1 713	97	134	182	185	118
LA05	2 863	2 746	2 698	2 698	2 242	184	236	295	292	2 253
MT06	37	37	37	37	37	1	2	3	4	<1
MT10-6x10	396	281	241	241	225	4	19	100	154	4
MT10-10x8	2 747	2 745	2 531	2 531	2 107	191	273	368	386	10 916
MT10-8x10	1 207	1 055	659	659	657	11	43	132	185	248
MT10-8x8	851	843	843	843	801	6	12	31	40	26
MT10-9x9	1 654	1 444	1 444	1 444	1 309	33	69	135	146	1 957
ORB1-8x8	1 298	1 298	1 271	1 271	1 101	7	13	30	39	12
ORB2-8x8	1 803	1 419	1 178	1 178	960	7	24	60	78	20
ORB2-9x9	1 232	1 232	1 232	1 232	1 016	21	50	127	147	570

Tabel 6.22: Shifting bottleneck ved TWT(1,3).

	Løsningsværdi					Køretid				
Problem	$\beta = 1$	$\beta = 3$	$\beta = 5$	$\beta = 5$ m. reopt.	KK	$\beta = 1$	$\beta = 3$	$\beta = 5$	$\beta = 5$ m. reopt.	KK
H20	777	777	777	777	681	1	1	2	2	1
H50	0	0	0	0	0	5	5	5	6	<1
H51	0	0	0	0	0	11	11	11	11	<1
H60	0	0	0	0	0	136	134	138	175	1
ABZ5-8x8	154	21	9	9	0	10	65	150	197	1
CAR5	14 505	13 061	13 061	13 061	11 874	366	621	767	818	24
CAR6	4 549	3 163	3 163	3 163	1 586	18	362	1 169	1 371	42
CAR7	2 402	2 305	1 883	1 883	1 207	2	11	35	43	10
CAR8	11 554	11 103	8 918	8 918	5 868	22	184	160	183	13
LA01	1 010	1 010	1 010	1 010	929	96	161	184	204	68
LA02	1 035	1 035	1 035	1 035	908	134	171	198	216	11 083
LA03	1 471	1 282	1 233	1 233	1 110	172	294	405	433	384
LA04	1 484	1 484	1 368	1 368	783	77	112	159	174	588
LA05	1 453	1 453	1 453	1 453	1 138	175	251	250	257	2 618
MT06	10	8	8	8	4	1	1	2	3	1
MT10-6x10	0	0	0	0	0	2	1	1	2	1
MT10-10x8	1 236	1 010	922	922	–	97	478	1 268	1 324	>10 t
MT10-8x10	12	12	12	12	0	6	17	29	66	5
MT10-8x8	130	119	38	38	36	7	50	110	143	42
MT10-9x9	597	222	222	230	66	20	231	1 077	1 485	10 078
ORB1-8x8	466	298	298	298	194	7	43	96	118	26
ORB2-8x8	359	210	162	162	33	6	31	86	99	85
ORB2-9x9	239	239	159	159	40	13	106	424	525	2 451

Tabel 6.23: Shifting bottleneck ved TWT(1,6).

	Løsningsværdi					Køretid				
Problem	$\beta = 1$	$\beta = 2$	$\beta = 3$	$\beta = 3$ m. reopt.	KK	$\beta = 1$	$\beta = 2$	$\beta = 3$	$\beta = 3$ m. reopt.	KK
H20	1 420	1 415	1 415	1 415	1 224	1	1	1	3	<1
H50	–	–	–	–	555	>10 t	>10 t	>10 t	>10 t	4
H51	–	–	–	–	565	>10 t	>10 t	>10 t	>10 t	4
H60	–	–	–	–	–	>10 t	>10 t	>10 t	>10 t	>10 t
ABZ5-8x8	2 182	2 182	2 152	2 152	1 769	7	17	37	48	23
CAR5	22 264	22 264	22 264	22 264	20 932	572	653	840	820	96
CAR6	18 106	18 106	17 525	17 525	15 879	28	108	336	377	26
CAR7	10 829	10 540	10 540	10 540	10 040	4	9	13	19	3
CAR8	19 386	19 386	19 386	19 386	17 603	24	49	110	128	7
LA01	2 202	2 202	2 165	2 165	1 983	269	301	366	375	257
LA02	2 355	2 355	2 108	2 108	1 816	370	389	449	449	2 168
LA03	2 217	2 217	2 157	2 157	1 768	372	374	539	569	81
LA04	2 318	2 318	2 281	2 281	1 752	235	264	289	315	530
LA05	2 399	2 160	2 160	2 160	1 789	287	301	345	374	3 713
MT06	83	83	83	83	68	1	2	2	4	1
MT10-6x10	820	820	820	820	820	6	26	117	181	3
MT10-10x8	2 743	2 743	2 649	2 649	2 170	391	442	586	629	4 908
MT10-8x10	1 603	1 603	1 603	1 603	1 400	17	59	211	304	169
MT10-8x8	1 782	1 674	1 502	1 502	1 303	11	20	53	67	42
MT10-9x9	2 344	2 170	2 170	2 170	1 747	63	166	383	494	1 151
ORB1-8x8	2 029	1 851	1 851	1 851	1 504	12	33	80	93	22
ORB2-8x8	1 609	1 609	1 564	1 564	1 389	8	15	38	50	20
ORB2-9x9	2 057	2 002	1 919	1 919	1 736	36	64	122	157	713

Tabel 6.24: Shifting bottleneck ved TT.

Selvom kvaliteten af løsningerne ikke er så god, vil man alligevel kunne bruge objektfunktionsværdierne af disse som en initiel bound ved branch-and-bound-søgning.

Dette er forsøgt, og resultaterne heraf er vist i tabel 6.25–6.28. Kolonnerne KK angiver antallet af knuder hhv. køretid for kørsler af ren kronologisk konfliktløsning. Kolonnen SB angiver køretiden af den initielle løsning vha. shifting bottleneck, og SB+KK angiver den samlede køretid for shifting bottleneck og den efterfølgende branch-and-bound-søgning samt antallet af knuder i søgetræet. Shifting bottleneck er kaldt med $\beta = 3$ for TWT(1,3) og TT, og med $\beta = 5$ for TWT(1,6).

For instanser, hvor løsningstiden for shifting bottleneck alene overstiger køretiden for eksakt løsning, er der selvfølgelig ikke håb om, at den kombinerede metode opnår et bedre resultat. Disse instanser er dog vist i skemaet alligevel, idet det stadig er interessant at se, hvor meget køretiden af kronologisk konfliktløsning forbedres, når den initielle overgrænse er relativt tæt på den optimale løsningsværdi.

I de fleste tilfælde er reduktionen i antal knuder marginal, hvorfor tidsforbruget på at finde en initiel løsning vha. shifting bottleneck overstiger reduktionen i køretiden af kronologisk konfliktløsning. Med den kvalitet af løsninger, som shifting bottleneck kan præstere, kan det således ikke betale sig at bruge metoden til at generere initielle løsninger. Formodentlig skal løsningerne være noget tættere på den optimale, før antallet af knuder reduceres væsentligt.

For visse instanser er det anførte antal knuder for den kombinerede metode større end for ren kronologisk konfliktløsning. Dette vil normalt ikke være tilfældet, men er et resultat af fejlimplementeringen nævnt i afsnit 6.4. Som nævnt i den forbindelse skønnes denne fejl dog ikke at have indflydelse på selve konklusionen.

	Antal knuder		Køretid		
Problem	KK	SB+KK	KK	SB+KK	SB
H20	290	35	1	2	1
H50	29	1	1	9	8
H51	34	1	1	9	9
H60	61	1	2	275	274
ABZ5-8x8	6 124	5 788	12	18	8
CAR5	2 333 468	2 333 432	5 391	5 677	370
CAR6	54 699	53 216	113	177	64
CAR7	9 528	9 301	14	20	5
CAR8	97 208	93 992	171	225	59
LA01	124	81	1	68	67
LA02	44 751	44 693	65	211	147
LA03	27 957	27 323	46	199	157
LA04	9 260	8 607	15	88	74
LA05	63	1	1	79	79
MT06	121	9	1	2	2
MT10-6x10	450	325	3	36	34
MT10-10x8	12 589 492	12 589 481	31 558	33 692	461
MT10-8x10	41 092	37 854	146	272	128
MT10-8x8	38 106	38 028	68	90	18
MT10-9x9	1 197 222	1 197 084	3 865	3 966	67
ORB1-8x8	71 705	71 640	112	138	24
ORB2-8x8	2 289	2 241	5	16	11
ORB2-9x9	103 362	103 232	280	340	59

Tabel 6.25: Kombination af shifting bottleneck (SB) og kronologisk konfliktløsning (KK) ved makespan.

	Antal knuder		Køretid		
Problem	KK	SB+KK	KK	SB+KK	SB
H20	446	446	1	3	2
H50	114	0	1	1 182	1 182
H51	42	0	1	31	31
H60	61	0	1	135	135
ABZ5-8x8	19 368	17 038	7	43	37
CAR5	43 803	43 803	15	856	841
CAR6	63 042	63 042	22	276	255
CAR7	8 854	8 648	3	21	18
CAR8	5 633	3 573	3	16	14
LA01	2 553 484	2 547 619	562	793	244
LA02	7 173 640	7 173 617	1 679	1 946	296
LA03	2 268 229	2 268 162	493	919	424
LA04	539 774	539 774	118	300	182
LA05	10 297 068	10 297 068	2 253	2 563	295
MT06	134	83	<1	3	3
MT10-6x10	7 408	6 862	4	103	100
MT10-10x8	30 883 211	24 963 206	10 916	9 539	368
MT10-8x10	719 208	688 225	248	378	132
MT10-8x8	90 445	88 702	26	57	31
MT10-9x9	5 425 843	5 336 353	1 957	2 040	135
ORB1-8x8	36 413	35 592	12	43	30
ORB2-8x8	70 023	54 684	20	77	60
ORB2-9x9	1 754 610	1 753 419	570	708	127

Tabel 6.26: Kombination af shifting bottleneck (SB) og kronologisk konfliktløsning (KK) ved TWT(1,3).

	Antal knuder		Køretid		
Problem	KK	SB+KK	KK	SB+KK	SB
H20	304	132	1	2	2
H50	34	0	<1	5	5
H51	33	0	<1	11	11
H60	61	1	1	138	138
ABZ5-8x8	1 273	173	1	151	150
CAR5	74 973	17 476	24	775	767
CAR6	128 063	127 397	42	1 213	1 169
CAR8	40 486	40 540	13	174	160
LA01	315 064	415 436	68	284	184
LA03	1 880 421	1 940 572	384	838	405
LA04	2 756 512	2 775 691	588	776	159
LA05	12 787 409	12 884 165	2 618	3 894	250
MT06	771	863	1	3	2
MT10-6x10	102	0	1	1	1
MT10-8x10	10 436	3 476	5	31	29
MT10-8x8	157 913	140 795	42	151	110
MT10-9x9	30 288 654	30 394 617	10 078	12 427	1 077
ORB1-8x8	84 714	85 190	26	124	96
ORB2-8x8	342 909	240 697	85	151	86
ORB2-9x9	8 018 619	8 165 571	2 451	3 009	424

Tabel 6.27: Kombination af shifting bottleneck (SB) og kronologisk konfliktløsning (KK) ved TWT(1,6).

	Antal knuder		Køretid		
Problem	KK	SB+KK	KK	SB+KK	SB
H20	41	41	<1	2	1
H50	5 157	–	4	–	>10 t
H51	4 619	–	4	–	>10 t
ABZ5-8x8	82 254	80 959	23	60	37
CAR5	340 216	340 214	96	937	840
CAR6	79 881	75 049	26	361	336
CAR7	8 543	8 458	3	16	13
CAR8	17 179	15 775	7	116	110
LA01	1 178 097	1 177 446	257	636	366
LA02	10 084 989	10 084 989	2 168	2 664	449
LA03	345 662	345 662	81	622	539
LA04	2 514 488	2 514 437	530	841	289
LA05	17 707 825	17 707 505	3 713	4 315	345
MT06	411	362	1	3	2
MT10-6x10	6 867	3 743	3	119	117
MT10-10x8	14 393 384	14 393 187	4 908	5 487	586
MT10-8x10	504 211	503 885	169	384	211
MT10-8x8	153 691	152 616	42	95	53
MT10-9x9	3 474 216	3 474 102	1 151	1 557	383
ORB1-8x8	71 817	65 844	22	101	80
ORB2-8x8	77 062	72 919	20	59	38
ORB2-9x9	2 246 381	2 218 368	713	845	122

Tabel 6.28: Kombination af shifting bottleneck (SB) og kronologisk konfliktløsning (KK) ved TT.

6.12 Løsning vha. CPLEX

I afsnit 3.9 anførte vi en matematisk formulering af problemet til brug ved løsning mht. CPLEX.

I tabellerne 6.29–6.32 er vist løsningstiden for CPLEX sammenlignet med køretiden af kronologisk konfliktløsning. Desuden er vist den opnåede objektfunktionsværdi. Løsningsværdien er mærket med en ring $\circ$ i de tilfælde, hvor CPLEX-løsningen er termineret pga. overskridelse af den maksimale tidsgrænse på 10 timer, samt med en asterisk *, hvor CPLEX terminerede pga. mangel på hukommelse.

Det fremgår tydeligt, at CPLEX er langt mindre effektiv end den specialiserede algoritme, med løsningstider der er i omegnen af 10 gange så høje, mest udpræget ved makespan. I de tilfælde, hvor kørslen er termineret pga. hukommelsesmangel eller for lang køretid, er den fundne løsning god, men dog som regel stadig omkring 10% fra den optimale.

Det blev forsøgt at "hjælpe" CPLEX ved at tilføje yderligere uligheder, der repræsenterer betingelser, der altid er opfyldt i en brugbar løsning. Eksempelvis forsøgtes med trekantsbetingelsen nævnt i [4], der siger, at hvis der findes to kanter $o_{gh} \rightarrow o_{ij}$ og $o_{ij} \rightarrow o_{kl}$, hvor $g < i < k$, dvs. $x^p_{gi} = x^p_{ik} = 1$, da afvikles o_{gh} før o_{kl}, dvs. at $x^p_{gh} = 1$. Uligheden lyder således, at

$$x^p_{gi} + x^p_{ik} + (1 - x^p_{gk}) \leq 2$$

Dette så dog umiddelbart ud til at føre til endnu højere køretider, hvorfor det ikke blev udforsket yderligere.

	Løsningsværdi		Køretid	
Problem	CPLEX	KK	CPLEX	KK
H20	1 013	1 013	<1	1
H50	2 890	2 890	1	1
H51	3 030	3 030	3	1
H60	1 793	1 793	5	2
ABZ5-8x8	985	985	332	12
CAR5	8 035 *	7 702	21 183	5 391
CAR6	8 469 *	8 313	31 847	113
CAR7	6 558	6 558	225	14
CAR8	8 264	8 264	29 284	171
LA01	666	666	486	1
LA02	655	655	837	65
LA03	597	597	11 647	46
LA04	590	590	149	15
LA05	593	593	17 327	1
MT06	55	55	<1	1
MT10-6x10	774	774	1	3
MT10-10x8	907 *	861	26 408	31 558
MT10-8x10	824	824	6 590	146
MT10-8x8	737	737	6 382	68
MT10-9x9	858 ∘	855	36 000	3 865
ORB1-8x8	787	787	8 398	112
ORB2-8x8	726	726	1 367	5
ORB2-9x9	793	793	9 174	280

Tabel 6.29: Løsning vha. CPLEX mht. makespan.

	Løsningsværdi		Køretid	
Problem	CPLEX	KK	CPLEX	KK
H20	1 253	1 253	2	1
H50	0	0	2	1
H51	0	0	2	1
H60	0	0	31	1
ABZ5-8x8	687	687	792	6
CAR5	29 931 *	26 114	19 674	37
CAR6	11 201	11 201	1 869	20
CAR7	7 785	7 785	112	3
CAR8	14 024	14 024	4 770	3
LA01	2 140 *	2 072	18 054	508
LA02	2 327 *	2 080	17 620	1 606
LA03	2 245 ∘	2 232	36 000	476
LA04	1 713	1 713	695	111
LA05	2 442 *	2 242	17 622	2 064
MT06	37	37	1	1
MT10-6x10	225	225	10	3
MT10-10x8	2 782 *	2 107	17 799	10 243
MT10-8x10	657	657	8 602	228
MT10-8x8	783	783	569	21
MT10-9x9	1 224	1 309	17 804	1 840
ORB1-8x8	1 101	1 101	727	11
ORB2-8x8	900	960	2 921	20
ORB2-9x9	1 211 *	1 016	21 040	559

Tabel 6.30: Løsning vha. CPLEX mht. TWT(1,3).

	Løsningsværdi		Køretid	
Problem	CPLEX	KK	CPLEX	KK
H20	681	681	<1	1
H50	0	0	1	<1
H51	0	0	1	<1
H60	0	0	10	1
ABZ5-8x8	0	0	71	1
CAR5	15 656 *	11 874	19 645	24
CAR6	1 586	1 586	1 301	42
CAR7	1 207	1 207	70	10
CAR8	5 868	5 868	16 418	13
LA01	904	929	2 969	68
LA02	1 041 *	908	18 068	11 083
LA03	1 125 *	1 110	19 206	384
LA04	783	783	15 290	588
LA05	1 240 *	1 138	17 377	2 618
MT06	1	4	1	1
MT10-6x10	0	0	<1	1
MT10-8x10	0	0	11	5
MT10-8x8	36	36	25	42
MT10-9x9	227 ∘	66	36 000	10 078
ORB1-8x8	194	194	2 382	26
ORB2-8x8	28	33	145	85
ORB2-9x9	40	40	4 007	2 451

Tabel 6.31: Løsning vha. CPLEX mht. TWT(1,6).

	Løsningsværdi		Køretid	
Problem	CPLEX	KK	CPLEX	KK
H20	1 224	1 224	<1	<1
H50	555	555	50	4
H51	565	565	124	4
ABZ5-8x8	1 769	1 769	13 812	23
CAR5	21 843 *	20 932	18 220	96
CAR6	15 879	15 879	14 024	26
CAR7	10 039	10 040	140	3
CAR8	17 603	17 603	19 502	7
LA01	2 051 *	1 983	19 026	257
LA02	1 960 *	1 816	17 397	2 168
LA03	1 921 *	1 768	16 214	81
LA04	1 838 *	1 752	21 414	530
LA05	1 937 *	1 789	20 581	3 713
MT06	68	68	2	1
MT10-6x10	820	820	43	3
MT10-10x8	2 451 *	2 170	21 825	4 908
MT10-8x10	1 400	1 400	32 485	169
MT10-8x8	1 303	1 303	4 873	42
MT10-9x9	1 988 *	1 747	19 638	1 151
ORB1-8x8	1 504	1 504	7 546	22
ORB2-8x8	1 389	1 389	13 723	20
ORB2-9x9	1 835 *	1 736	20 595	713

Tabel 6.32: Løsning vha. CPLEX mht. TT.

6.13 Opsummering

Vi har nu undersøgt de i kapitel 5 omtalte løsningsmetoder.

Køretiderne har været meget forskellige, selv for instanser med samme antal maskiner og jobs.

Vi har undersøgt metoderne for flere objektfunktioner, hvor især makespan er væsentligt anderledes end de øvrige. Selve målingerne har ofte været meget forskellige objektfunktionerne imellem, men konklusionerne mht. brug af en given metode har flere steder været den samme for dem alle.

Kapitel 7

Konklusion

Målet med dette projekt har været at undersøge forskellige metoder til løsning af JSS med udgangspunkt i et praktisk problem, togskedulering.

Vi introducerede job-shop-skeduleringsproblemet samt en række af de begreber, der anvendes i forbindelse med forskellige typer af løsningsmetoder, herunder to objektfunktioner. Derudover har vi formuleret problemet som et mixed integer-problem.

Herefter formulerede vi togskeduleringsproblemet som et JSS-problem. Det skete med nogle antagelser om jernbanenettets struktur, der synes rimelige i visse praktiske tilfælde.

Vi har omtalt og implementeret et antal løsningsmetoder og -komponenter, heraf to branching-strategier, en bounding-metode, et dominanskriterium, en heuristisk løsningsmetode samt en lokalsøgningsstrategi. De fleste af disse er omtalt i to varianter — en for hver objektfunktion.

De implementerede metoder er blevet testet med en række instanser, dels med generelle JSS-instanser fra forskellige kilder, dels med Higgins-instanserne, der er baseret på autentiske data i forbindelse med togskedulering og således har væsentligt anderledes karakteristika. De fleste af sidstnævnte var dog relativt små, hvilket gør det svært at konkludere ret meget generelt på baggrund heraf.

Desværre viste der sig kort før afleveringen af projektet en implementeringsfejl, hvorfor nogle af de præsenterede resultater er fremkommet ved en

fejlbehæftet version af programmellet. Umiddelbart ser dette ikke ud til at ændre på konklusionerne, men det kan ikke udelukkes, at fejlen kan have haft indflydelse på visse — især de kvantitative — konklusioner.

Vi undersøgte et antal dispatch-regler, og selvom der ikke var nogen klar favorit, var det dog den samme regel, mindste vægtede ventetid, der så mest lovende ud for begge de betragtede objektfunktioner.

Af de to betragtede branching-strategier viste kronologisk konfliktløsning sig at fungere bedst. Dog skal det siges, at branching ved kantindsættelse havde væsentligt højere køretider end en publiceret implementation af nogenlunde samme algoritme, hvilket antyder, at det måske snarere er et problem i implementeringen end i metoden isoleret set. Dette problem kunne meget vel også gøre sig gældende ved kronologisk konfliktløsning, forstået på den måde, at metoden i en anden implementation ville have en bedre ydelse.

Det undersøgte dominanskriterium gav ikke anledning til bedre køretider ved undersøgelse af de generelle job-shop-instanser, mens det ved Higgins-instanserne så ud til at være mere frugtbart.

Løsning af Higgins-instanserne skete noget hurtigere end i [17]. Det er her ikke helt samme problemstilling, der er betragtet. Alligevel antages, at løsningen i [17] kunne drage nytte af den heri omtalte bounding-funktion samt dominanskriteriet.

Brug af lokalsøgning i forbindelse med branch-and-bound så ud til at give en smule bedre løsningstider.

Den omtalte boundingmetode for makespan viste sig at være effektiv. Sammen med kantindsættelse kunne søgetræet beskæres betragteligt, hvilket dog ikke altid gav anledning til bedre køretider. Den nye metode for TWT gav anledning til væsentlig beskæring af søgetræet, dog for det meste på bekostning af en noget højere køretid, hvorfor det generelt ikke kan betale sig at bruge denne metode til bounding.

Shifting bottleneck gav — uanset løsningstiden — noget ringere løsningsværdier end tidligere publicerede resultater, hvilket antyder en fejl i implementeringen. De fundne løsninger var således ikke gode nok til at reducere antallet af knuder i søgetræet væsentligt, hvis deres objektfunktionsværdi blev brugt som initiel overgrænse ved kronologisk konfliktløsning.

For at opsummere har de beregningsmæssige resultater overordnet set været skuffende sammenlignet med rapporterede resultater fra litteraturen;

dog har visse enkeltdele af metoderne givet lovende resultater, herunder dominanskriteriet for kronologisk konfliktløsning. Ved videre udforskning af dette kræves et større sæt testinstanser af samme type som Higgins-instanserne. Det bør dog bemærkes, at selvom dominanskriteriet forbedrer køretiden for kronologisk konfliktløsning, så udelukker det ikke, at der findes andre branching-strategier, der giver endnu bedre resultater.

Litteratur

[1] J. Adams, E. Balas og D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988.

[2] J. Aerts. A survey of optimization algorithms for job shop scheduling. Technical report, Department of Mathematics and Computing Sciences, Eindhoven University of Technology, October 1997.

[3] D. Applegate og W. Cook. Jobshop. C-programmer baseret på [4]. ftp://ftp.caam.rice.edu/pub/people/applegate/jobshop/.

[4] D. Applegate og W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, 1991.

[5] E. Balas, J. K. Lenstra og A. Vazacopoulos. The one-machine problem with delayed precedence constraints and its use on job shop scheduling. *Management Science*, 41(1):94–109, 1995.

[6] J. E. Beasley. OR-Library. http://www.ms.ic.ac.uk/jeb/info.html.

[7] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11(1):42–47, 1982.

[8] R. W. Conway, W. L. Maxwell og L. W. Miller. *Theory of Scheduling*. Addison Wesley, 1967.

[9] E. Demirkol, S. Mehta og R. Uzsoy. A computational study of shifting bottleneck procedures for shop scheduling problems. *Journal of Heuristics*, 3(2):111–137, 1997.

[10] M. Garey, D. Johnson og R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[11] O. Holthaus og C. Rajendran. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics*, 48(1):87–105, 1997.

[12] V. Jain og I. Grossmann. Algorithms for hybrid MILP/CP models for

a class of optimization problems. *INFORMS Journal on Computing*, 13(4):258–76, 2001.

[13] S. Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3(3):125–138, 2000.

[14] J. Lenstra og A. Rinnooy Kan. Complexity results for scheduling chains on a single machine. *European Journal of Operational Research*, 4(4):270–275, 1980.

[15] S. Martello og P. Toth. *Knapsack Problems : algorithms and computer implementations*. Wiley-Interscience series in discrete mathematics. John Wiley & Sons, 1990.

[16] E. Oliveira og B. M. Smith. A job-shop scheduling model for the single-track railway scheduling problem. Technical Report 2000.21, University of Leeds, School of Computing, august 2000.

[17] E. Oliveira og B. M. Smith. A hybrid constraint-based method for the single-track railway scheduling problem. Technical Report 2001.04, University of Leeds, School of Computing, august 2001.

[18] M. Perregaard. Branch and bound methods for the multi-processor job-shop and flow-shop scheduling problem, 1998.

[19] M. Perregaard og J. Clausen. Parallel branch-and-bound methods for the job-shop scheduling problem. *Annals of Operations Research*, 83:137–160, 1998.

[20] M. Pinedo og M. Singer. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics*, 46(1):1–17, 1999.

[21] M. Singer og M. Pinedo. A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE Transactions*, 30(2):109–118, 1998.

Bilag A

Symbolliste

$\mathcal{M}$	Mængden af maskiner, $\{M_1, \ldots, M_m\}$
m	Antallet af maskiner, $\lvert\mathcal{M}\rvert$
M_p	Maskine nr. p
$\mathcal{J}$	Mængden af jobs, $\{J_1, \ldots, J_n\}$
n	Antallet af jobs, $\lvert\mathcal{J}\rvert$
J_i	Job nr. i bestående af operationerne $o_{i1}, \ldots, o_{im}$
r_i	Tidligst tilladte starttidspunkt for jobbet J_i
C_i	Afslutningstidspunkt for jobbet J_i
d_i	Deadline for jobbet J_i
T_i	Tardiness af jobbet J_i
w_i	Vægt af jobbet J_i
o_{ij}	Operation nr. j på jobbet J_i
S_{ij}	Starttidspunkt (hoved) for operationen o_{ij}
C_{ij}	Afslutningstidspunkt for operationen o_{ij}
p_{ij}	Behandlingstid for operationen o_{ij}
l_{ij}	Resterende behandlingstid for operationen o_{ij} (bounding)
q_{ij}	Hale for operationen o_{ij}
q_{ij}^k	Hale for operationen o_{ij} mht. jobbet J_k
d_{ij}	Deadline for operationen o_{ij}

$\mathcal{S}$	Kildeknude
$\mathcal{V}$	Afløbsknude (makespan)
$\mathcal{V}_i$	Afløbsknude for jobbet J_i (TWT)
$L(o_{ij}, o_{kl})$	Længste vej mellem knuderne o_{ij} og o_{kl}
$\mathcal{P}$	JSS-instans
O	Objektfunktion
E	Løsning repræsenteret ved kanterne i en grafen
$\mathcal{N}$	Nutid (dominans)
Q	Fortidens bidrag til objektfunktionsværdien (dominans)
p, q	Indeks for maskiner, dvs. M_p og M_q
g, i, k	Indeks for jobs, dvs. J_g, J_i og J_k
h, j, l	Indeks for operationer på jobs, dvs. o_{gh}, o_{ij} og o_{kl}