Bruk av tabusøk og critical event memory på set partitioning-problemet

Christian Magnus Berg
Institutt for Informatikk, Universitetet i Bergen
christia@ii.uib.no

Arne Løkketangen
Institutt for Informatikk, Høgskolen i Molde
arne.lokketangen@himolde.no

Oversikt

Ideer fra en heuristikk skrevet av Glover og Kochenberger for bruk på 0-1 ryggsekk-problemet forsøkes tilpasset og brukt på set partitioning-problemet. Denne heuristikken var basert på tidligere arbeid med tabusøk og surrogate constraints. Den introduserte critical event memory, og benyttet en interessant oscillering mellom lovlige og ulovlige løsninger. Denne er ikke nødvendigvis helt overførbar til set partitioning-problemet, men tankegangen kan lett tilpasses de beslektede problemene set packing og set covering. Set partitioning-problemet forsøkes løst med en spesialtilpasset utgave av denne heuristikken.

1. Introduksjon

Et av flyselskapenes største problemer er planlegging av mannskapsdisposisjoner. Reglene for benyttelse av flymannskap er strenge. Flyselskapene er pålagt grenser for hvor mange timer om dagen et mannskap kan være i luften, hvor mange timer et mannskap kan være borte fra sin base før de må ha hotellovernatting, etc. Mannskap utgjør også den nest største utgiften til et flyselskap, etter drivstoff (Hoffman og Padberg, 1993). Set partitioning-problemet kan brukes til å finne det billigste oppsettet for bruk av mannskap på et sett med flygninger, og det å finne gode løsninger av dette problemet kan dermed innebære enorme økonomiske besparelser.

Dette gjøres ved at man først setter opp potensielle skift med beregnet kostnad. I dette oppsettet streber man etter at hvert mannskap returnerer til utgangspunktet ved endt arbeidsdag. Som man ser kan antallet mulige skift bli temmelig stort med kun et lite antall flyruter som skal gjennomføres. Det blir derfor en umulig oppgave å finne den billigste skiftplanen manuelt, eller ved enumerasjon av de forskjellige kombinasjonene som gjør at hver flygning dekkes med nøyaktig ett mannskap. I mange tilfeller vil det være vanskelig i det hele tatt å finne en slik kombinasjon. Derfor trenger vi effektive heuristikker til å løse dette problemet.

I mange tilfeller skjer det også forandringer planene som følge av forsinkelser, kanselleringer og/eller ekstraflygninger. Da kan det være nyttig å ha alternative skiftplaner. Man er altså ikke nødvendigvis bare interessert i de optimale løsningene, men også andre gode løsninger.

2. Set partitioning problemet

Set partitioning-problemet, eller mengdepartisjoneringsproblemet, går ut på å finne en rimeligst mulig partisjon av en mengde utifra et gitt sett med delmengder. Problemet kan formuleres som følger:

$$\min \sum_i c_i x_i$$

$$\sum_i a_{ij} x_i = 1, \forall j \qquad (1)$$

$$x_i \in \{0,1\} \qquad (2)$$

Der c_i er kostnaden til delmengde i, x_i er 1 hvis delmengde i er med i løsningen og 0 ellers, og a_{ij} spesifiserer hvorvidt delmengde i dekker element j i mengden, og vil således være enten 0 eller 1.

Problemet er altså å velge ut et sett av delmengder som er slik at de tilsammen nøyaktig dekker hele mengden. Samtidig skal vi finne den billigste av alle slike sett med delmengder. Disse delmengdene er i matrisen A å finne som kolonner, og resten av artikkelen vil derfor veksle mellom å referere til disse som kolonner og delmenger.

2.1 Anvendelser

Som nevnt i introduksjonen, kan optimal oppdragsfordeling for flymannskap kan finnes ved å løse dette problemet. De forskjellige tilgjengelige delmengdene vil da representere hvilke flyruter et bestemt mannskap kan dekke. Målet er å finne en ruteplan som vil dekke alle flyturer nøyaktig en gang (et mannskap per tur), og koste minst mulig penger. $a_{ij} = 1$ betyr at mannskap i kan dekke flytur nr j.

En annen anvendelse kan være plasseringer av brannstasjoner eller sykehus i en by. Hver delmengde vil da angi hvilke områder av byen en brannstasjon med en gitt plassering kan dekke. a_{ij} er enten 1 eller 0. Er den 1 betyr det at delmengde nr i, dvs at den potensielle brannstasjon nr i, dekker bydel nr j.

Av de potensielle brannstasjonene vil det være flere som dekker samme område. Hvis man velger alle mulige plasseringer, vil hver bydel ha en betydelig overdekning av brannstasjoner. Dette er kanskje fint, men dyrt og unødvendig. Man vil derfor søke å velge ut stasjoner slik at hver bydel er dekt nøyaktig en brannstasjon, samtidig som kostnadene holdes lavest mulig. Dette vil ikke nødvendigvis innebære færrest mulig brannstasjoner, siden enkelte tomtealternativer vil gi høyere investeringskostnader enn andre.

2.2 Relaterte problem

I enkelte tilfeller vil det være umulig å finne en løsning som overholder alle sidekravene (1). I de to nevnte eksemplene, vil man da søke å finne en billigst mulig løsning med

overdekning, det vil si at en eller flere flyturer får mer enn ett mannskap, og mer enn en brannstasjon dekker en gitt bydel. Problemet formulert med tillatelse til slik overdekning kalles set covering. En annen variant, der underdekning tillates, kalles set packing. I disse problemene vil sidekravene (1) byttes ut henholdsvis med sidekravene (3) og (4).

$$\sum_i a_{ij} x_i \geq 1, \forall j \qquad (3)$$

$$\sum_i a_{ij} x_i \leq 1, \forall j \qquad (4)$$

3. Litt bakgrunn om tabusøk og critical event memory

3.1 Tabusøk

Tabusøk er en strategi for søk blant løsninger av et optimeringsproblem som er ment å forhindre følgende problem med enkle søkerutiner.

En generell lokal søkerutine vil til enhver tid velge den beste løsningen i sitt nabolag, der nabolaget er definert som et sett med nærliggende løsninger. Når denne algoritmen kommer til et lokalt optimum, vil den velge den beste av løsningene i nabolaget til optimumet, selv om denne er dårligere. I neste trekk risikerer vi da at den beste løsningen i nabolaget er det lokale optimumet. Søkerutinen vil da velge denne, for så å velge den beste av løsningene i nabolaget til det lokale optimumet igjen. Her blir søkerutinen stående og gå mellom disse to løsningene.

Tabusøk løser dette problemet ved at nylig besøkte løsninger blir tabu, eller forbudte, for en gitt periode. Det vil si at søket ikke umiddelbart får lov til å bevege seg tilbake til det lokalet optimumet, men må velge en annen løsning i stedet. Dermed vil søket være i stand til å bevege seg bort fra lokale optima.

Av praktiske årsaker vil som oftest tabusøket fungere slik at det er en variabel eller kolonne som er tabu, og ikke hele løsningen. Umiddelbart ser en at dette vil kunne medføre at søket avviser en løsning som er bedre enn den inkuberende beste løsning, fordi delmengden som må endre sin status ikke kan gjøre nettopp det på grunn av taburestriksjonen. For å unngå dette, definerer man et aspirasjonskriterium. Dette er et krav man stiller for at taburestriksjonen skal kunne omgås. Det kan for eksempel være at omgåelse av tabukriteriet gir en ny beste løsning. (Glover og Løkketangen, 1998. Wolsey, 1998)

3.2 Critical event memory

Tabusøk søker til enhver tid det beste tillatte trekket og gjennomfører det. Dette gjør at tabusøk kan bli svært konsentrert om et enkelt område. Det finnes flere teknikker for å bøte på dette, blant annet probabilistisk tabusøk, som oppfordrer til spredning i søkeområdet ved å innføre tilfeldigheter for hvorvidt et trekk skal velges.

Spredning i søket er målet til critical event memory også. Som navnet antyder er dette basert rundt såkalte ”critical events”, som kan være at en løsning er funnet eller et bytte fra

en konstruktiv til en destruktiv fase i et oscillerende søk. Når critical events inntreffer, gjennomføres en avstraffelse av de delmengdene som er med i den kritiske løsningen. Delmengdene blir da straffet etter hvor ofte de har vært å finne i slike løsninger, de blir med andre ord straffet etter frekvens. Delmengder som ofte er i slike løsninger vil etter hvert bli mindre attraktive, og sannsynligheten er da at søket vil dreie seg i retninger der disse ikke er med like ofte.

Ved også å definere overgangen fra konstruktiv til destruktiv fase som en kritisk hendelse, kan man gjøre critical event memory effektiv til å spre søket, selv når man ikke ennå har funnet noen lovlige løsninger på problemet.

4. Løsningsstrategi 1 – initielle løsninger

Lokalsøkheuristikker finner som regel en initiell løsning, som den deretter forsøker å forbedre ved å søke i nabolaget til denne løsningen. Denne heuristikken har tre forskjellige strategier for å finne slike initielle løsninger, nemlig tilfeldig, konstruktiv eller ingen, det vil si at ingen kolonner er valgt ut initielt.

4.1 Tilfeldig initiell løsning

Strategien for å finne en tilfeldig initiell løsning er å finne hvor mange kolonner man i gjennomsnitt må velge for å få en set partitioning-løsing. Hvis det er a rader som skal dekkes, og hver kolonne i gjennomsnitt dekker b rader, så trenger vi i gjennomsnitt $c = a / b$ kolonner for å dekke alle radene. Den tilfeldige heuristikken velger derfor opptil c kolonner tilfeldig, mens den hele tiden passer på å ikke innføre overdekning. Det finnes ingen garanti for at dette gir en partitioning-løsning, og det er dermed ikke sikkert at dette gir et godt utgangspunkt for lokalsøket.

4.2 Konstruktiv initiell løsning

Ved den konstruktive strategien velger man først en kolonne. Hvordan denne raden skal velges, kan variere; man kan for eksempel velge den med best kostnad/dekning-ratio, eller en tilfeldig kolonne. Tilfeldig kolonne er mest aktuelt ved det første valget av kolonne, slik at heuristikken kan gi forskjellige løsninger til bruk for eksempel ved restart av søket. Etter at valget er gjort, slettes alle rader kolonnen dekker, samt alle andre kolonner som dekker noen av de slettede radene. Disse kolonnene vil uansett ikke være å finne i noen løsning med den valgte kolonnen. Dette gjentas til det ikke er flere kolonner igjen å velge mellom. Da har man forhåpentligvis fått en god løsning på set partitioning-problemet. Det er ingen garanti, men man har i hvert fall en packing-løsning.

4.3 Sammenligning

Den konstruktive løsningsmetoden ser ut til å gi søket et noe bedre utgangspunkt enn den tilfeldige, i og med at den gir noe bedre løsninger. Det ser ikke ut til å være noen nevneverdig forskjell på å bruke en tilfeldig løsning eller ingen løsning som utgangspunkt for søket.

4.4 Restart

En strategi som benyttes er å starte lokalsøket på nytt etter en gitt mengde iterasjoner uten forbedring. Da bygges en ny løsning opp fra bunnen, helst ved hjelp av den konstruktive heuristikken, og lokalsøket startes på nytt derfra. Dette gir heuristikken blant annet mulighet til å rømme til et annet område i løsningsmengden dersom det skulle bli for konsentrert om et lite område som ikke ser ut til å være særlig fruktbart.

5. Løsningsstrategi 2 - lokalsøk

Hvis man angriper problemet som et set packing problem, vil man kunne bevege seg fra en løsning til en annen ved hjelp av et sett add- og drop-operasjoner. Ved å ta en løsning, kan man legge til noen nye delmengder. Man vil da bevege seg fra å ha en gyldig set packing-løsning til en gyldig set covering-løsning. Ved å fjerne noen av de valgte delmengdene, kan man så bevege seg tilbake til en set packing-løsning. Vi bruker dette som definisjon på nabolaget til en løsning. Samtidig skal en huske at målet er å finne løsninger på set partitioning, og man må da prøve spesielt å sikte seg inn på disse løsningene som vil finnes i grensen mellom løsningene til covering og packing. Det er heller ikke gitt at om man har en set partitioning-løsning, så vil en slik add/drop-sekvens føre frem til en annen set partitioning-løsning.

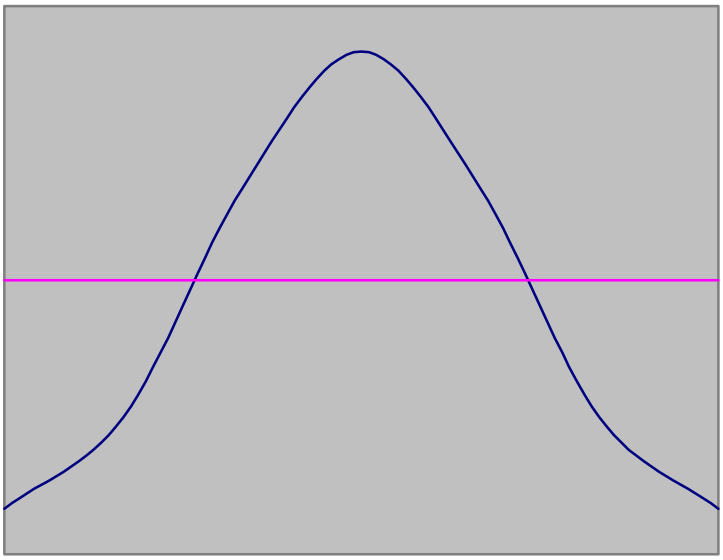

Fig 1. Visualisering av bevegelsen gjennom forskjellige løsninger i en iterasjon. Over den horisontale linjen har vi set cover-løsninger, under set packing-løsninger, og på linjen set partition-løsninger. Toppen og bunnen markerer overganger mellom add- og dropfaser.

Bevegelse mellom lovlige løsninger vil altså innebære en midlertidig bevegelse inn i et ulovlig område, før man finner en ny lovlig løsning. Dette gjøres i to faser.

1. Konstruktiv fase:
   Man legger til nye delmengder i løsningen, dermed vil man bevege seg inn på set covering-løsninger.
2. Destruktiv fase:
   Man fjerner overflødige delmengder, slik at man kommer tilbake til en ny set packing-løsning.

Inne i disse to fasene håper man å treffe løsninger på set partition-problemet. Derfor er de to fasene delt inn i to delfaser. Den første delfasen går ut på taktisk utvelgelse av delmengder slik at man sikter seg inn på en set partitioning-løsning. Når denne er funnet, eller man gir opp, går man over til delfase to, der man legger til de mest lovende eller fjerner de dårligste delmengdene, uten å bekymre seg om over- eller underdekning.

Utifra dette ser en at enhver konstruktiv fase starter med en set packing-løsning og ender i en set cover-løsning, mens enhver destruktiv fase starter med en set cover-løsning og ender i en set packing-løsning. Etter den destruktive fasen går man over til en ny konstruktiv fase, og en konstruktiv fase etterfulgt av en destruktiv fase vil dermed utgjøre en iterasjon.

Hvilke delmengder som velges for å legges til eller fjernes, bestemmes utifra en rekke kriterier.

5.1 Surrogate constraints

Surrogate constraints (Glover og Løkketangen, 1997) er det viktigste kriteriet for utvelgelse av kolonner. Som surrogate constraint velges delmengdenes kostnad/antall-ratio. Det høres fornuftig ut å rangere delmengdene etter kostnad per dekt element. I konstruktive faser foretrekkes derfor delmengder som har en lav kostnad/antall-ratio, mens delmengder men en høy kostnad per dekt element forsøkes fjernet i destruktive faser.

5.2 Tabu

Kolonner som nettopp har fått sin status (med/ikke med) endret, får ikke umiddelbart endre sin status igjen. Dette medfører at en iterasjon ikke innebærer opprettholdelse av status quo. Samme løsning vil ikke bli reprodusert kun en iterasjon etter at den sist var aktiv. Dette vil gjøre at man unngår å sirkle rundt lokale optima. Det eksperimenteres med forskjellige tabustrategier.

Under forsøkene har det vist seg at å skille mellom tabu for kolonner som ikke er med i en løsning, et såkalt add-tabu, med kolonner som ikke kan legges til, og tabu for kolonner i løsning, drop-tabu, med kolonner som ikke kan fjernes, ikke fungerer spesielt godt. Egentlig følger dette ganske enkelt av at drop-tabuet setter en minimumsbegrensning på antall kolonner i en løsning, og at løsninger med færre kolonner enn størrelsen på drop-tabu tenure dermed vil være vanskelige å finne.

Det virker dog å være minimale forskjeller på å ha ett generelt tabu, alle kolonner som har byttet status i det siste er tabu, eller å bare bruke add-tabuet.

Tabumekanismen i seg selv innebærer at man kan unngå å legge til eller trekke fra en kolonne som ville gitt den optimale, eller i hvert fall en ny beste, løsning. Dette er det selvsagt ønskelig å unngå, og vi bruker derfor som aspirasjonskriterium at tabu ignoreres dersom kolonnen gir en ny beste løsning.

5.3 Probabilistisk valg av trekk

Det eksperimenteres også med probabilistisk tabusøk. Denne såkalte probabilismen har rett nok lite med selve tabumekanismen å gjøre, men viser seg å gi gode resultatforbedringer. Det hele går ut på at det gis en sannsynlighet for at den tilsynelatende beste delmengden basert på surrogate constraints, velges. Hvis ikke den beste delmengden da blir valgt, velges neste med samme sannsynlighet, osv. Dette er motivert utifra at det slett ikke alltid er slik at de delmengdene som har den beste kostnaden per element er med i den optimale løsningen, og denne taktikken sørger for at det er gode muligheter for å få andre delmengder inn i løsningen. Det er vist at probabilistisk tabusøk er garantert å finne optimal løsning, gitt at søket får gå lenge nok (Løkketangen og Glover, 1996).

Når dette har vært testet, ser det ut til at å sette sjansen for å akseptere beste delmengde til 33% gir noe bedre resultater enn 67%, til gjengjeld ser sistnevnte ut til å konsentrere søket raskere, slik at man raskere kommer frem til de gode løsningene. Forskjellen mellom de to ser ut til å være nærmest ubetydelig.

33% sjanse for å velge beste delmengde høres kanskje ut til å gi store tilfeldigheter i søket, men dette innebærer i virkeligheten at det er cirka 70% sjanse for å velge å velge en av de tre beste delmengdene.

5.4 Critical event memory

Delmengder som stadig er med i løsninger, vil ikke nødvendigvis være med i den optimale løsningen. En mekanisme for å bli kvitt slike delmengder er bruk av critical event memory. Hver gang en delmengde er med i en løsning, blir denne tildelt en liten straff ved at dens surrogate constraint svekkes, slik at det er større mulighet for at nettopp denne delmengden blir oversett ved neste korsvei. Delmengder som ofte er med i løsninger, vil selvsagt være oftere utsatt for dette enn andre. Denne strategien har vist seg å gi meget gode resultater innen styring av søk.

Straffen blir satt på bakgrunn av delmengdens frekvens i løsninger, og forskjellige formler for å beregne straffen har vært utprøvd. Den som gir best resultater ser ut til å være den utledet av Glover og Kochenberger (Glover og Kochenberger 1996) til deres tilsvarende heuristikk for det flerdimensjonale 0-1 ryggsekk-problemet.

$$TABU\_F(j) * PEN\_F = TABU\_F(j) * r^*/s$$

PEN_F er straffen en delmengde tildeles for å være med i en kritisk løsning. Straffen beregnes på bakgrunn av delmengdens frekvens i slike løsninger. r^* er den største r_j for

alle delmengder j, der r_j er en ratio beregnet for denne delmengden, i vårt tilfelle kostnad delt på antall dekte elementer, altså sammenfallende med surrogate constraintet som benyttes for rangering av delmengdene. s er produktet av antall iterasjoner og en stor, positiv konstant C. *TABU_F(j)* er delmengde j sin frekvens i kritiske løsninger.

Grunnen til å ta med en slik konstant, er å kunne avveie straffen. Den skal ikke være for stor, for vi vil ikke revolusjonere rangeringen av delmengdene, men samtidig må den være stor nok til å ha den ønskede effekt. Dette oppnås bra for det flerdimensjonale 0-1 ryggsekk-problemet med C=100.000, og dette tallet ser ut til å fungere godt også for set partition-problemet. Likevel testes andre verdier for C, og det kan se ut som mange av set partitioning-problemene løses bedre ved en noe lavere C.

Andre definisjoner av avstraffelse, og da spesielt av *PEN_F* testes også ut. Blant annet kan nevnes at varianten $f^* * r^*/s$, der f^* = max (*TABU_F(j)*) gir omtrent like gode resultater som Glover og Kochenbergers metode. Ved bruk av Glover og Kochenberger ser en imidlertid ut til å komme frem til de gode løsningene en god del raskere enn ved denne varianten for en del problemer for C=100.000.

5.5 Add/drop syklus - oscillering

Problemene har blitt testet med en meget enkelt oscillering for hvor mange delmengder som skal legges til eller fjernes i andre del av add- og drop-fasene, det vil si etter at man har passert linjen der set partition-løsningene ligger. Denne oscilleringen øker antallet som legges til eller fjernes med 1 for hver iterasjon inntil en maksimumsgrense er nådd. Deretter reduseres antallet med 1 inntil en minimumsgrense er nådd. Det kan være forskjellige minimums- og maksimumsgrenser for add og drop.

Glover og Kochenberger beskriver en noe mer avansert oscillering. I deres versjon er antall delmengder som skal legges til eller fjernes lagret i en variabel *span*. Denne er fast for et gitt antall iterasjoner og endres deretter systematisk. Til å begynne med er *span*=1, og to parametre, *p1* og *p2*, har gitte verdier, og et flagg viser at *span* øker. Beregningene av *span* gjøres i overgangen til andre del av add- og dropfasene som følger:

Span øker:

> Hvis *span* er mellom 1 og p1, inklusive: Tillat *p2***span* iterasjoner og øk deretter *span* med 1.
>
> Hvis *span* er mellom *p1* + 1 og *p2*, inklusive: Tillat *p2* iterasjoner og øk deretter *span* med 1. Når *span* blir større enn *p2*, sett den tilbake til *p2* og sett flagget til å vise at span avtar.

Span avtar:

> Hvis *span* er mellom *p2* og *p1* + 1, inklusive: Tillatt *p2* iterasjoner og reduser så *span* med 1.
>
> Hvis *span* er mellom *p1* og 1, inklusive: Tillatt *p2***span* iterasjoner og reduser så *span* med 1. Når *span* blir 0, sett den til 1, og sett flagget til å vise at *span* øker.

Denne formen for oscillering har vist seg å bringe søket inn på mer interessante områder og dermed bedre løsninger, med enkelte verdier for *p1* og *p2*. En ser umiddelbart at man på denne måten først konsentrerer seg om et relativt avgrenset område, før man går over til å spre søket mer i en periode for så å gå tilbake til å søke nøyere gjennom det nye området. Den enkle oscilleringen fjerner seg lettere fra områder som kan inneholde interessante løsninger, da den setter av tid til å søke grundig gjennom dem.

6. Videre testing

6.1 Preprosessering

Det er fremdeles et par teknikker som skal testes for å se hvordan de påvirker heuristikken. En av de man har størst forhåpninger til, er preprosessering av problemene, der redundante rader og kolonner fjernes. Blant annet finnes i noen av testtilfellene kolonner som er identiske, men med forskjellig kostnad. Meningen er å fjerne de dyreste av disse, og se hvordan dettte påvirker søket.

Det er liten tvil om at av slike delmengder, så er det en viss fare for å velge en dårligere, spesielt ved benyttelse av probabilistisk valg av trekk.

Beasley og Chu brukte også preprosessering av disse problemene i sin genetiske heuristikk (Beasley og Chu, 1998). De viste at ved bruk av fem regler for reduksjon av problemene kunne man redusere noen av dem ganske dramatisk. Blant andre kan nevnes problemet ”us01” som ble redusert fra 1 053 137 kolonner og 145 rader til 351 018 kolonner og 86 rader ved bruk av fem regler.

Mye av årsaken til disse gode resultatene, ligger i at de er hentet fra den virkelige verden. Tilfeldig genererte problemer vil mest sannsynlig ikke ha så stor grad av redundante kolonner og rader.

I følgende avsnitt er

- β_x mengden rader dekt av kolonne x i den tidligere nevnte matrisen A.
- α_x mengden kolonner dekt av rad x i matrisen A.
- c_x kostnaden forbundet med å velge kolonne x i løsningen.
- I mengden av rader.
- J mengden av kolonner.

Det fem reduksjonsreglene til Beasley og Chu er:

- Hvis $\beta_j = \beta_k$ for et par $(j, k) \in J, j \neq k$ og $c_k \geq c_j$, slett kolonne k, fordi denne er et minst like dyrt duplikat av kolonne j.

- Hvis $|\alpha_i| = 1$, $i \in I$, så må kolonne j i α_i være i den optimale løsningen. Slett j og alle rader $k \in \beta_j$, samt alle kolonnene i α_k, $k \in \beta_j$.

- Hvis $\alpha_i \subseteq \alpha_k$ for et par $(j, k) \in I$, $i \neq k$, så slett alle kolonner $j \in (\alpha_k - \alpha_i)$ og rad k. Denne reduksjonen kommer av at alle kolonner som dekker rad k også dekker rad i. Det betyr at ingen kolonner som dekker rad k, men ikke rad i, kan være med i noen løsning på problemet, og disse kolonnene, samt rad k, kan dermed fjernes.

- Hvis $|\alpha_i - (\alpha_i \cap \alpha_k)| = |\alpha_k - (\alpha_i \cap \alpha_k)| = 1$, $i \neq k$, $i, k \in I$, så lar vi kolonne $j = \alpha_i - (\alpha_i \cap \alpha_k)$ og kolonne $p = \alpha_k - (\alpha_i \cap \alpha_k)$. Da vil p og k være like bortsett fra at de i tillegg har en unik kolonne hver, nemlig j og p.

  1. Hvis $\beta_j \cap \beta_p = \varnothing$, kan kolonnene j og p slås sammen til én kolonne med kostnaden $c_j + c_p$, siden de ikke dekker noen felles rader. Rad k kan slettes, da den etter dette vil være lik rad i.

  2. Hvis $\beta_j \cap \beta_p \neq \varnothing$, sletter vi kolonnene j og p, da valget av en av disse vil gjøre den andre uvalgbar, og man kan dermed ikke få dekket den av radene i og k som den valgte kolonnen ikke dekker. Ingen av disse kan dermed være med i noen løsning. Vi sletter også rad k, fordi den etter dette vil være lik rad i.

- For hver $j \in J$, anta at $x_j = 1$. Da må $x_k = 0$, $\forall k \in T = (\{\cup_{i \in \beta j} \alpha_i\} - j)$. Det vil si at hvis vi velger kolonne j, så kan vi ikke velge noen av kolonnene som dekker noen av de samme radene som j. Hvis $U = \{i \mid \alpha_i \subseteq T, i \in I - \beta_j\} \neq \varnothing$ så er problemet uløsbart hvis $x_j = 1$ fordi rad $i \in U$ ikke kan dekkes. Derfor må $x_j = 0$, og vi har dermed ikke bruk for denne kolonnen. U er mengen rader som ikke dekkes av j, og som heller ikke dekkes av noen kolonner utenom mengden T.

6.2 Oscillering på tabu tenure

For å hindre at søket begynner å sirkulere i samme område ved å til stadighet finne igjen de samme løsningene, skal man undersøke hvilken påvirkning det har for søket at størrelsen på tabu tenure varierer underveis. Forhåpentligvis vil dette kunne hindre noe sirkulering og gi enda bedre løsninger på enkelte av problemene.

7. Resultater og testtilfeller

Heuristikken er blitt testet på set partitioning-problemer hentet fra J. Beasleys OR-library. Dette biblioteket har 55 set partitioning-problemer som er hentet fra luftfartsindustrien. Størrelsen på problemene varierer fra 197 kolonner og 17 rader til 1 053 137 kolonner og 145 rader. Andelen enere i matrisen A varierer fra 0.99% til 39.27%.

54 av problemene er hittil blitt testet. Med de hittil beste settingene er 9 løst til optimalitet, 10 løst med mindre enn 5% overskridelse av optimalverdi, ytterligere 4 løst med mindre enn 10% overskridelse, dessuten 20 til løst med mer enn 10% overskridelse. 11 problemer ble ikke løst med disse settingene. Tilsammen er 12 problemer løst til optimalitet.

Man håper fremdeles å forbedre disse tallene, spesielt ved også å løse preprosesserte utgaver av problemene.

Referanser

Beasley, J. E.: "OR-library": http://www.ms.ic.ac.uk/info.html

Chu, P. C. og Beasley, J. E.: "Constraint Handling in Genetic Algorithms: The Set Partitioning Problem", Journal of Heuristics, vol 4, no 4, Dec 1998, sidene 323-357

Glover, Fred og Kochenberger, Gary A: "Critical Event Tabu Search for Multidimensional Knapsack Problems", i I.H. Osman J.P. Kelly, red, "Metaheuristics: The Theory and Applications", sidene 407-427. Kluwer Academic Publishers, 1996

Hoffman, Karla L. og Padberg, Manfred: "Solving Airline Crew Scheduling Problems by Branch-and-Cut", "Management Science", vol 39, no 6, juni 1993, sidene 657-682

Løkketangen, Arne og Glover, Fred: "Probabilistic Move Selection in Tabu Search for Zero-One Mixed Integer Programming Problems", i "Metaheuristics: Theory and Applications", Kluwer, 1996

Løkketangen, Arne og Glover, Fred: "Surrogate Constraint Analysis – New Heuristics and Learning Schemes for Satisfiability Problems", DIMACS, vol 35, 1997, sidene 537-571

Løkketangen, Arne og Glover, Fred: "Solving zero-one mixed integer programming problems using tabu search", European Journal of Operational Research", vol 106, 1998, sidene 624-658

Wolsey, Laurence A.: "Integer Programming", Wiley-Interscience, 1998