AMPL Hjälp

1 Att komma igång

Ni kan hämta filer under /users/mdstud/ouo00/data. De filer som är av intresse är farm.mod, farm.dat farm2.mod samt farm2.dat. farm innehåller det deterministiska problemet, farm2 innehåller ett problem med tre scenarier.

Exemplet i detta papper finns även det, och filerna heter nisse.mod och nisse.dat. I detta papper är alla exempel lösta med LP_SOLVE. Ni kommer att använda MINOS eller CPLEX

2 Nisse-exemplet

Allt i detta papper relaterar till ett exempel vilket handlar om Nisse Pärlemo som blandar drinkar. Exemplet består av två filer, nisse.mod vilken innehåller den matematiska modellen och nisse.dat vilken innehåller data till modellen.

Det första ni skall göra när ni har startat ampl (genom att ge kommandot ampl) är att skriva `options solver ampl_cplex`. Detta kommer att byta ut den ordinarie lösaren mot CPLEX, vilken är mer stabil, samt klarar heltalsproblem. Om ni inte gör detta kommer ni att köra en testversion av MINOS som borde klara de flesta exempel.

För att prova nisse-exemplet skriver ni
`ampl: model nisse.mod;`

Kommandot laddar in modellfilen nisse.mod i AMPL. Skriv nu kommandot
`ampl: show;`

AMPL listar nu alla saker som finns i minnet. Utskriften borde vara

```
parameters:
juice_innehåll      tillgång            önskad_volym
pris                önskad_juiceandel

sets:   INGREDIENSER   JUICER   PRODUKTER

variable:   innehåll

constraints:   Andel_Juice   Erfoderlig_Volym   Lager

objective:   Kostnad
```

Detta är alla komponenter som finns i modellen. Skriver vi nu
`ampl: show juice_innehåll;`

så får vi utskriften

```
param juice_innehåll{INGREDIENSER, JUICER};
```

vilket är definitionen av vår parameter juice_innehåll. Mer om detta senare.

Om vi vill se värdet av juice_innehåll skriver vi istället
`ampl: display juice_inehåll;`

Nu borde vi få svaret

```
Error executing "display" command:

        No values for juice_innehåll.
```

Detta beror på att vi än så länge bara har specificerat att det existerar en parameter som heter juice_innehåll, men ej fyllt den med innehåll. Det gör vi genom att ladda in parametrarna till problemet med kommandot
`ampl: data nisse.dat;`

Ett nytt försök med `display juice_innehåll` ger oss nu

```
juice_innehåll [*,*]
:      Apelsin Grape   Kiwi 'Vinbär'    :=
A        0.4     0.4   0.05   0
B        0.05    0.1   0.1    0.2
C        1       0     0      0
D        0       1     0      0
E        0       0     0.6    0
Vatten   0       0     0      0
;
```

vilket är en tabell över hur mycket av vilka juicer våra ingredienser innehåller.

De enda variabler vi har i modellen är matrisen innehåll. Skriver vi kommandot
`ampl: display innehåll ;`

så får vi

```
innehåll :=
Hawaii    A        0
Hawaii    B        0
.
.
Taj_Mahal E        0
Taj_Mahal Vatten   0
;
```

vilket är hur mycket varje drink innehåller av de olika ingredienserna. Alla variabler är noll då vi ännu inte har löst problemet. Vi löser genom kommandot
`ampl: solve;`
vilket ger utskriften

```
LP_SOLVE 2.2: optimal, objective 4278.75
7 simplex iterations
LP_SOLVE 2.2: optimal, objective 4278.75
7 simplex iterations
```

Ni kan själva skriva kommandofiler för att automatisera vissa uppgifter (se nedan).

Nu kan vi titta på variabeln innehåll varvid vi får

```
ampl: display innehåll ;
innehåll :=
Hawaii    A          0
Hawaii    B        100
Hawaii    C         75
Hawaii    D         30
Hawaii    E          0
Hawaii    Vatten   195
Taj_Mahal A         37.5
Taj_Mahal B          0
Taj_Mahal C          0
Taj_Mahal D          0
Taj_Mahal E         30.2083
Taj_Mahal Vatten    32.2917
;
```

Kontrollerar man så ser man att denna blandning precis uppfyller kraven på juice-innehåll.

Om vi försöker ladda in problemet en gång till så får vi följande problem

```
nisse.dat, line 2 (offset 51):
        PRODUKTER is already defined
context:  set  >>> PRODUKTER: <<< = Hawaii Taj_Mahal;

nisse.dat, line 3 (offset 85):
        INGREDIENSER is already defined
context:  set  >>> INGREDIENSER: <<< = A B C D E Vatten;
.
.
```

Detta beror på att AMPL redan har komponenter i minnet. När vi försöker ladda in problemet igen tror AMPL att vi försöker specificera nya komponenter med samma namn vilket ger krockar. Om man vill ladda in en ny version av problemet måste man först skriva
`ampl: reset;`
Vill man bara radera parametrarnas värde så skriver man
`ampl: reset data;`

Vill man bara ändra en enskild parameter kan man göra detta med kommandot let.
`ampl: let önskad_volym["Hawaii"]:=300;`
ändrar den önskade volymen av Hawaii till 300 liter. Efter ändringar så måste naturligtvis problemet lösas om för att ändringarna skall ta effekt. Kommandot `let` kan även användas på variabler.
`ampl: let innehåll["Hawaii","A"]:=10;`
sätter innehållet av A i Hawaii till 10 liter. Om vi vill att det nya värdet på variabeln skall bevaras även när vi löser om problemet så måste vi fixera värdet. Detta görs med kommandot fix.
`ampl: fix innehåll["Hawaii","A"];`
fixerar det nya värdet av mängden A i Hawaii.

Löser vi problemet igen får vi

```
LP_SOLVE 2.2: optimal, objective 3576.25
7 simplex iteration
```

```
LP_SOLVE 2.2: optimal, objective 3576.25
7 simplex iterations
```

Priset har sjunkit då vi bara tillverkar 300 liter Hawaii. Tittar vi på det nya värdet av innehåll med `display` så får vi

```
ampl: display innehåll;
innehåll :=
Hawaii    A         10
Hawaii    B         75
Hawaii    C         52.25
Hawaii    D         18.5
Hawaii    E          0
Hawaii    Vatten   144.25
Taj_Mahal A         37.5
Taj_Mahal B          0
Taj_Mahal C          0
Taj_Mahal D          0
Taj_Mahal E         30.2083
Taj_Mahal Vatten    32.2917
;
```

3 AMPL-språkets viktigaste komponenter.

När man specificerar en modell i AMPL-språket så skriver man två olika filer. En modellfil, som innehåller själva modellen, och en datafil vilken innehåller data för ett specifikt problem. Om vi tittar på Nisse-exemplet så säger vi i modellfilen att vi vill blanda ett visst antal drinkar av ett antal ingredienser, samt att vi vill minimera kostnaden för detta under vissa bivillkor. I data-filen skriver vi vilka drinkar vi vill blanda av vilka ingredienser, vad varje ingrediens kostar och vad de innehåller. I datafilen kan vi inte införa några nya mängder eller definiera parametrar som inte redan finns beskrivna i modellfilen. Datafilen fyller alltså modellens skal med innehåll.

De kommandon som är nödvändiga då man specificerar sin modell är:

set	:	deklarerar en mängd
param	:	deklarerar en parameter
var	:	deklarerar en variabel
minimize	:	deklarerar en målfunktion
subject to	:	deklarerar ett bivillkor.

3.1 set

I nisse-exemplet står det i modellfilen
`set PRODUKTER;`

Det betyder att vi deklarerar en mängd produkter vi vill tillverka. I datafilen skriver vi
`set PRODUKTER:= Hawaii Taj_Mahal;`

Då fyller vi setet med innehåll, vi säger att vi producerar två olika produkter och att dessa heter Hawaii och Taj_Mahal.

Om vi hade försökt att deklarera detta set endast i datafilen så hade AMPL vägrat. I datafilen får vi endast fylla de set som tidigare deklarerats i modellfilen.

På mängder (set) kan man använda olika operatorer. Den mest användbara är union. Om nisse t.ex hade haft två olika varugrupper, nämligen

```
set ALKOHOLFRITT;
set OVRIGT;
```

så hade vi kunnat bilda mängden av alla drinkar genom att (i modellfilen) skriva
`set ALLA_DRINKAR := ALKOHOLFRITT union OVRIGT;`
Nu kan vi definiera bivilkor (se nedan) vilka skall gälla på setet ALLA_DRINKAR men vi kan även ha bivillkor vilka bara gäller setet ALKOHOLFRITT.

På samma sätt kan man bilda skärningen av set genom kommandot `inter`. Fler operationer på set finns i det utdelade utdraget ur AMPL-boken.

3.2 param

Parametrar deklareras genom ordet param. Om vi till exempel ville att nisse skulle räkna med moms så skulle vi kunna definiera en parameter för detta genom att i modellfilen skriva
`param moms;`
vilket gör att begreppet moms finns i modellen.

I data-filen skriver vi
`param moms := 0.25;`
för att tala om den faktiska moms-satsen.

Vi kan även indexera våra parametrar över mängder. I nisse-exemplet har vi till exempel deklarationen
`param önskad_volym{PRODUKTER};`
Detta betyder att vi för varje element i mängden PRODUKTER anger hur mycket kunden vill ha.

Parametern fylls sedan med innehåll i data-filen genom kommandot

```
param önskad_volym:=
Hawaii    400
Taj_Mahal 100;
```

Det går även att ha två eller flera dimensioner på sina parametrar vilket nisse-exemplet visar med t.ex
`param juice_innehåll{INGREDIENSER,JUICER};`
vilken innehåller andelen av varje juice i varje ingrediens.

Fler exempel på parameterdeklarationer finns i det utdelade utdraget ur AMPL-boken.

3.3 var

Variabler definieras även de i modellfilen. Variabler används aldrig i datafilen. I nisseexemplet har vi deklarationen
`var innehåll{PRODUKTER,INGREDIENSER} >= 0 ;`

Detta deklarerar en variabel för varje kombination av produkt och ingrediens (hur mycket av varje ingrediens i varje produkt). Vidare så säger denna ingrediens att innehållet av alla ingredienser skall vara större eller lika med 0.

Om vi deklarerar variabeln x som
`var x <= 5, >= 3;`
så kan vi ange både en övre och nedre gräns.

Om vi vill ange en övre gräns för en matris av variabler samtidigt blir det lite mer komplicerat. Då får vi skriva som följer (i modellfilen). Först definierar vi två set

```
set ETTSET;
set ANNATSET;
```

Sedan definierar vi en övre gräns för variabeln. (Parametern måste fortfarande fyllas med innehåll i data-filen).

```
param max_för_y{ETTSET,ANNATSET};
```

Sedan måste vi tilldela gränsen för varje enskild medlem av vektorn x vilket görs genom

```
var y{i in ETTSET,j in ANNATSET} >= 0, <= max_för_y[i,j];
```

Vill vi ange att en variablel skall vara heltalig görs detta med hjälp av nyckelordet `integer` och binära variabler markeras med `binary`. Vill vi till exempel ha en heltalig variabel mella 3 och 5 så anges detta med

`var x integer <= 5, >= 3;`
och en binär variabel deklareras som

```
var z binary;
```

3.4 minimize

Vi måste ha en målfunktion för att kunna ha något att optimera. En målfunktion deklareras med orden `minimize` resp `maximize`. I nisse-exemplet har vi

```
minimize Kostnad : sum{i in PRODUKTER, j in INGREDIENSER}
                                (innehåll[i,j]*pris[j]);
```

Här säger vi att vi skall minimera en Kostnad som är en summa över samtliga produkters ingredienser multiplicerat med priset för sagda ingrediens. Notera att man kan göra radbrytningar nästan över allt (dock ej inne i ord). AMPL fortsätter att läsa och tolkar allt fram till ";" som en sats. Notera vidare att vi summerar över två index samtidigt. Man kan summera över fler index genom att bara addera dem med "," emellan.

3.5 subject to

Till sist så har vi bivillkor. I Nisse-exemplet har vi bland annat bilvillkoren Lager.

```
subject to Lager {j in INGREDIENSER}:
           sum{i in PRODUKTER} innehåll[i,j] <= tillgång[j];
```

Om vi bryter ned detta så står det på första raden
`subject to Lager {j in INGREDIENSER}:`

Detta betyder att vi skapar ett villkor för varje j tillhörande setet INGREDIENSER. Sedan (efter :) kommer själva villkoret. Detta säger att summan över samtliga produkter av innehållet av j (index för ingrediens) skall vara mindre än tillgången av ingrediens j. På samma sätt som med summor så kan man indexera över flera set separerade med ",".

I nisse-exemplet har vi

```
subject to Andel_Juice { i in PRODUKTER, k in JUICER}:
           sum{j in INGREDIENSER}innehåll[i,j] * juice_innehåll[j,k]
                                   >= önskad_juiceandel[i,k] * önskad_volym[i];
```

vilket ger undre gränser för andelen av olika juicer.

Vill man ha ett enstaka villkor så struntar man bara i `{.....}` i definitionen.

4 Analys av lösningen

Genom att ange suffix på variabler och bivillkor kan man få ut extra information. Genom att skriva
`display Lager.slack;`
så får vi ut värdet på slackvariabeln tillhörande samtliga Lagerbivillkor.

```
Lager.slack [*] :=
     A       162.5
     B       300
     C        25
     D        20
     E       769.792
Vatten  Infinity
;
```

Genom kommandot
`display Lager.dual;`
får vi värdet på dualvariabeln tillhörande lager. Inte särskilt upphetsande.
`display Andel_Juice.dual;`
är betydligt intressantare.

En mer komplett beskrivning av samtliga suffix finns i övrigt utdelat material. Genom att skriva
`ampl: display Lager.dual , Lager.slack;`
så kan man få många olika uppgifter i en tabell. Man kan naturligtvis ange fler parametrar separerade med ",".

5 Mest sannolika missarna

Q: Jag skrev ett kommando men inget hände, när jag sedan skriver in nästa så får jag märkliga fel av typen

```
syntax error
context: »>.............
```

A: Du har nog glömt ";" efter förra kommandot. Om inget händer när ett kommando skrivs, titta på prompten. Står det
`ampl?`
så väntar ampl fortfarande på slutet på förra kommandot.

Q: Jag får ett felmeddelande med hänvisning till en viss rad när jag försöker ladda in min modell. När jag tittar på raden ser den jätterätt ut. Vad göra?

A: Åter igen ";". Titta på raden innan och se om den avslutas med ";". AMPL varnar på närmast efterföljande rad då det är där satsen avslutas på ett felaktigt sätt.

Q: När jag försöker få in min modell så måste jag skriva kommandona
`reset; model minmodell.mod; data minmodell.dat;` väldigt många gånger i följd. Kan man underlätta detta på något sätt?

A: Ja, skriv en kommandofil. Skapa helt enkelt en fil som tex heter minfil. Skriv in kommandona du vill upprepa många gånger i minfil. Kommandona körs genom att skriva `include minfil`.

Q: Jag få fel av typen
`invalid subscript innehåll[j,i]` trots att jag borde indexerat rätt.

A: Kontrollera ordningen på dina index.

6 Mer information

En uppsjö av färdiga exempel finns på
http://www.ampl.com/cm/cs/what/ampl//BOOK/EXAMPLES/index_files.html .

Mer info om ampl finns på http://www.ampl.com/cm/cs/what/ampl/.
Denna sida är närmast att betrakta som överkurs och innehåller mest info om scriptspråk och generiska sätt att komma åt samtliga bivillkorl etc.

En rapport om en äldre version av AMPL med div. exemplel och definition av satser finns på
http://slack.iems.nwu.edu/ampl/papers.html
där man även kan hitta ett kapitel ur boken om AMPL.