2 - Modellatori e risolutori

Mauro Passacantando

Dipartimento di Scienze Economico-Aziendali e Diritto per l'Economia
Università degli Studi di Milano-Bicocca
mauro.passacantando@unimib.it

Corso di Dinamica dei Sistemi Aziendali
Laurea Magistrale in Scienze Economico-Aziendali
Università degli Studi di Milano-Bicocca

Modellatori e risolutori

Un risolutore (*solver*) è un software che riceve in input una descrizione di un problema di ottimizzazione e fornisce in output la soluzione ottima del problema stesso.

È importante:

- scrivere il modello in un linguaggio semplice (ad alto livello) per l'utente
- fornire separatamente modello e dati del problema in modo da poter risolvere più istanze dello stesso problema

si utilizzano i generatori algebrici di modelli o modellatori.

Modellatori: AMPL, GAMS, OPL, ...
Risolutori: GUROBI, CPLEX, MINOS, SNOPT, BARON ...

AMPL

AMPL (A Mathematical Programming Language):

- riceve e restituisce file di testo
- permette di importare i dati da file $\longrightarrow$ un solo modello per diverse istanze
- si può scaricare dalla pagina web del corso
- manuale completo: *R. Fourer, D.M. Gay, B.W. Kernighan, AMPL: A Modeling Language for Mathematical Programming, Brooks/Cole Publishing Company / Cengage Learning, 2002.* (disponibile nella pagina web del corso)
- articolo: R. Fourer and D.M. Gay, B.W. Kernighan, A Modeling Language for Mathematical Programming. Management Science 36 (1990) 519-554, (disponibile alla pagina: `http://www.ampl.com/REFS/amplmod.pdf`)
- per maggiori informazioni: `http://www.ampl.com`

Modello e dati

Modello descrive la struttura matematica di un problema di ottimizzazione.
Dati caratterizzano la singola istanza del problema.

Esempio 2

Il problema di programmazione lineare da risolvere è:

$$\begin{cases} \max\ 3000x_1 + 5000x_2 + 4000x_3 + 4500x_4 \\ x_1 + 2x_4 \leq 4 \\ 2x_2 + x_3 \leq 12 \\ 3x_1 + 2x_2 + 4x_3 + x_4 \leq 18 \\ x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0 \end{cases}$$

Modello:

$$\begin{cases} \max \sum_{j=1}^{n} p_j x_j \\ \sum_{j=1}^{n} t_{ij} x_j \leq D_i \qquad \forall\ i = 1, \ldots, m \\ x_j \geq 0 \qquad \forall\ j = 1, \ldots, n \end{cases}$$

in cui definiamo le variabili (incognite) x_j ed i parametri (noti) n (numero di prodotti), m (numero di stabilimenti), p_j (profitto del prodotto j), t_{ij} (tempo di produzione del prodotto j nello stabilimento i), D_i (tempo di produzione disponibile nello stabilimento i).

Modello e dati

Nel problema che vogliamo risolvere i parametri assumono i seguenti valori:
Dati: $m = 3$, $n = 4$,

$$p_j = \boxed{\begin{array}{cccc} 3000 & 5000 & 4000 & 4500 \end{array}}$$

$$t_{ij} = \boxed{\begin{array}{cccc} 1 & 0 & 0 & 2 \\ 0 & 2 & 1 & 0 \\ 3 & 2 & 4 & 3 \end{array}} \qquad D_i = \boxed{\begin{array}{c} 4 \\ 12 \\ 18 \end{array}}$$

Modello e dati

In AMPL due file (di testo) separati:

- modello $\rightarrow$ file.mod
- dati $\rightarrow$ file.dat

In file.mod si dichiarano:

- dati (insiemi, parametri, vettori e matrici di dati)
- variabili
- funzione obiettivo
- vincoli

In file.dat si assegna un valore numerico ai dati (si definiscono i dati).

Sintassi del file.mod

`#` indica un commento

`;` deve concludere ogni riga

insiemi vengono indicati con il comando `set`

dati vengono indicati da `param`

indici su insiemi: per indicare che un certo indice prende valori in un insieme si usa `in`

sottoinsieme di un insieme: per indicare che un insieme contiene alcuni elementi di un altro insieme si usa `within`

matrici e vettori: i dati possono essere raccolti in matrici e vettori: oltre a `param` bisogna indicare in quale insieme prendono valori gli indici

controllo sui dati: si possono imporre condizioni sui dati che vengono verificate quando vengono assegnati i valori numerici

variabili sono introdotte da `var`. Per definire il loro tipo:

- `binary` per variabili binarie
- `integer` per variabili intere
- `>=0` per variabili continue maggiori o uguali a 0

Sintassi del file.mod

La funzione obiettivo è introdotta da `minimize` o `maximize`. Alla funzione obiettivo è associato un nome. La struttura con cui si dichiara la funzione obiettivo è `minimize nome_della_funzione:`

I vincoli sono introdotti da `subject to` (oppure `s.t.`) e a ciascuno di essi è associato un nome. Bisogna inoltre definire quanti vincoli di un certo tipo sono presenti.

Sintassi del file.dat

singolo valore `param N := 10;`

insieme `set A := 1 3 4 6 8;`

insieme `set B := 1..4;`

vettore (definito su `A`) `param p :=`

```
1 10
3 14
4 18
6 21
8 15;
```

matrice (definita su `A` $\times$ `B`) `param c: 1 2 3 4 :=`

```
1 0 1 0 0
3 1 0 1 0
4 0 0 0 1
6 1 1 0 0
8 1 0 0 1;
```

Comandi AMPL

- `model file.mod;` carica il modello
- `data file.dat;` carica i dati
- `option solver gurobi;` sceglie GUROBI come solver
- `solve;` risolve il problema
- `display` accede ai valori della soluzione:
  - variabili o parametri: `display nomeVariabile;`
  - variabili duali: `display nomeVincolo;`
  - variabili di scarto: `display nomeVincolo.slack;`
- `:=` assegna un valore iniziale non modificabile a un parametro
- `default` assegna un valore iniziale modificabile a un parametro
- `let :=` assegna un valore a un parametro; può essere chiamato più volte per uno stesso parametro
- `for` esegue un blocco di istruzioni finché è verificata una condizione
- `printf` stampa (su video o su file)
- `reset;` cancella il modello e i dati
- `reset data;` cancella solo i dati
- `quit;` esce da AMPL

Tutti i comandi possono essere inseriti in un file.run

Esempio 2

Modello:

$$\begin{cases} \max \sum_{j=1}^{n} p_j x_j \\ \sum_{j=1}^{n} t_{ij} x_j \leq D_i \qquad \forall\ i = 1, \ldots, m \\ x_j \geq 0 \qquad \forall\ j = 1, \ldots, n \end{cases}$$

esempio2.mod

```
# Modello per il problema dell'esempio 2

param m >0;   # numero degli stabilimenti
param n >0;   # numero dei prodotti
param p{j in 1..n} >=0;   # profitto del prodotto j
param t{i in 1..m, j in 1..n} >=0;  # tempi di produzione
param D{i in 1..m} >=0;  # tempi di produzione disponibili

var x{j in 1..n} >=0; # x[j]= numero di lotti di prodotto j realizzati

maximize ProfittoTotale: sum{j in 1..n} p[j]*x[j];

s.t. tempi{i in 1..m}: sum{j in 1..n} t[i,j]*x[j] <= D[i];
```

Esempio 2

esempio2.dat

```
# Dati per il problema dell'esempio 2

param m := 3;   # numero degli stabilimenti
param n := 4;   # numero dei prodotti
param p :=
1 3000
2 5000
3 4000
4 4500; # profitti
param t: 1 2 3 4 :=
1 1 0 0 2
2 0 2 1 0
3 3 2 4 3; # tempi di produzione
param D :=
1 4
2 12
3 18; # tempi di produzione disponibili
```

Esempio 2

esempio2.run

```
# Script per il problema dell'esempio 2

reset;
reset;

model esempio2.mod;
data esempio2.dat;

option solver gurobi;
option gurobi_options 'timelim 60 outlev 1' ;

solve;

display x;
```

Esercizio

- Scrivere il file .dat relativo al problema dell'esempio 1
- Risolvere con AMPL il problema dell'esempio 1 utilizzando il file .dat scritto in precedenza ed il file .mod scritto per il problema dell'esempio 2