2
Introduzione ad AMPL

Come ampiamente discusso nel capitolo precedente, l'*approccio modellistico* rappresenta un potente "strumento" per la soluzione di un problema di decisione. In particolare, rappresentare un problema attraverso un modello di Programmazione Matematica permette di utilizzare i numerosi solutori disponibili che sono in grado di risolvere efficientemente problemi anche di dimensioni molto elevate. Tuttavia pur disponendo di efficienti algoritmi di soluzione, sarà necessario trasferire al solutore il problema rappresentato dal modello di Programmazione Matematica. Ovvero è necessario implementare il modello in una qualche forma riconoscibile dai vari solutori. Gli strumenti che svolgono questa funzione vengono chiamati *generatori algebrici di modelli* e, ormai da diversi anni, ne sono stati messi a punto alcuni che hanno avuto una vasta diffusione. Uno di questi è AMPL (acronimo di *A Mathematical Programming Language*) che è, ad oggi, uno dei più largamente utilizzati per la formulazione di problemi di Programmazione Lineare, Programmazione Lineare Intera, Programmazione Non Lineare e mista. Esso ha la funzione di interfaccia tra il modello algebrico ed il solutore numerico e rappresenta un potente linguaggio di modellazione algebrica, ovvero un linguaggio che contiene diverse primitive per esprimere le notazioni normalmente utilizzate per formulare un problema di Programmazione Matematica: sommatorie, funzioni matematiche elementari, operazioni tra insiemi, etc. Utilizza un'architettura molto avanzata, fornendo un'ottima flessibilità d'uso; infatti, la naturale notazione algebrica utilizzata permette di implementare modelli anche complessi e di dimensioni molto elevate in una forma assai concisa e facilmente comprensibile. È di fatto un linguaggio ad alto livello e non fa uso di particolari strutture dati perché utilizza solamente file di testo sia per l'implementazione del modello sia

per la memorizzazione dei dati. In aggiunta, per affermare l'esigenza già ricordata di rendere un modello indipendente dai dati, permette di tenere distinta la struttura del modello dai dati numerici che sono memorizzati separatamente. Inoltre, permette di richiamare al suo interno diversi tra i più efficienti solutori per problemi di Programmazione Matematica di cui attualmente si dispone. È disponibile per i sistemi operativi Windows, Unix/Linux e Mac OS X e tutte le informazioni sono reperibili sul sito `http://www.ampl.com`. Sono disponibili la AMPL IDE version e la AMPL Command Line version. Nel seguito, si farà esplicito riferimento a sistemi Windows e alla versione "command line". Gli studenti possono scaricare la Demo version. Tale versione è limitata nel numero delle variabili e dei vincoli. In particolare, per problemi lineari fino a 500 variabili e 500 vincoli; per problemi non lineari fino a 300 variabili e 300 vincoli. È anche disponibile una versione di prova completa senza limitazioni valida 30 giorni (AMPL Free 30-day trials).

Il testo di riferimento è:

> Robert Fourer, David M. Gay, Brian W. Kernighan. AMPL *A Modeling Language For Mathematical Programming*, Second Edition, Duxbury Thomson, 2003

disponibile sul sito di AMPL al link `http://ampl.com/resources/the-ampl-book`.

Allo stesso link si possono trovare i file di tutti gli esempi riportati nel libro. Sul sito è inoltre disponibile ulteriore materiale riguardante AMPL e i solutori supportati.

2.1 INSTALLAZIONE E AVVIO DI AMPL

Per quanto riguarda la versione Windows a riga di comando, andrà scaricato il file `ampl-demo-mswin.zip` che è un archivio contenente tutti i file necessari. Una volta estratto l'archivio in una directory, saranno create due directory (MODELS e TABLES) che contengono numerosi utili esempi e dei file eseguibili. In particolare il file `ampl.exe` è l'eseguibile di AMPL che chiamato dal prompt di comandi avvierà il programma. Gli altri file eseguibili `cplex.exe`, `gurobi.exe`, `minos.exe` e `lpsolve.exe` sono i solutori disponibili, rispettivamente CPLEX 12.6, Gurobi 6.0, MINOS 5.51 e LP_SOLVE 4.0. Si tratta di una versione a linea di comando e quindi deve essere utilizzata dalla finestra del *prompt di comandi* (finestra nera). Quindi aprire tale finestra, portarsi nella directory dove è stato estratto il file `ampl-demo-mswin.zip` (oppure aggiungere questa directory nel `PATH` di Windows) ed avviare il programma digitando il comando `ampl`. Apparirà così il prompt dei comandi di AMPL:

```
ampl:
```

A questo punto si è nell'ambiente AMPL, ovvero possono essere digitati i comandi di AMPL. Alternativamente, per avviare AMPL si può cliccare due volte su sw.exe (*scrolling-window utility*), digitando poi `ampl` sul prompt che appare.

2.2 UN PRIMO ESEMPIO

In linea generale (anche se assai poco pratico) un modello potrebbe essere inserito dal prompt di comandi di AMPL, ovvero digitando, ad esempio, sulla riga di comando i seguenti comandi in successione:

```
ampl: var x1;
ampl: var x2;
ampl: maximize funzione_obiettivo: x1 + x2;
ampl: subject to vincolo1: x1 + x2 <= 1;
ampl: subject to vincolo2: x1 - x2 <= 2;
ampl: subject to x1_non_neg: x1 >= 0;
ampl: subject to x2_non_neg: x2 >= 0;
```

Vedremo più avanti come sia possibile scrivere tali comandi in un file, ma per il momento soffermiamoci sull'analisi dei comandi AMPL utilizzati nell'esempio e sulle parole chiave del linguaggio. Inanzitutto osserviamo che

- ogni comando termina con un "`;`"
- sono presenti due variabili (x_1 e x_2) e queste sono dichiarate con i comandi `var x1;` e `var x2;`
- la funzione obiettivo è introdotta dalla parola chiave `maximize` in quanto trattasi di un problema di massimizzazione (per problemi di minimizzazione la parola chiave sarà `minimize`). Tale parola chiave è seguita da una etichetta proposta dall'utente (nel caso dell'esempio è `funzione_obiettivo`) seguita da "`:` che introduco all'espressione della funzione obiettivo
- i vincoli sono elencati di seguito: ciascun vincolo è introdotto dalla parola chiave `subject to` (che può anche esser abbreviata in `s.t.`), seguita dall'etichetta che si vuole dare al vincolo e da "`:` dopo il quale è riportata l'espressione del vincolo.

Questo semplice esempio ci ha permesso di introdurre la struttura tipica di un modello nel linguaggio AMPL, struttura che ricalca fedelmente quella standard di un modello di Programmazione Matematica come riportato nella Tabella 1.3.1.

Ovvero, dopo aver introdotto le *variabili di decisione*, deve essere formalizzata la *funzione obiettivo* e i *vincoli*.
Ora che abbiamo il modello implementato in AMPL, naturalmente viene spontaneo domandarsi come può essere determinata la soluzione del problema di ottimizzazione rappresentato dal modello (che nel caso dell'esempio è un problema di Programmazione Lineare). Come già detto da AMPL sono disponibili alcuni solutori. Nella directory dove risiede AMPL oltre al file `ampl.exe` sono presenti altri file eseguibili che rappresentano i solutori; essi sono CPLEX,GUROBI, MINOS e LPSOLVE. Ciascuno di essi risolve alcune classe di problemi di Programmazione Matematica. Nel seguito riporteremo nel dettaglio le tipologie di problemi risolte da ciascun solutore. Ora vogliamo introdurre il comando per selezionare il solutore da utilizzare, ovvero

`option solver` *nome_solutore*;

Quindi digitando:

```
ampl: option solver cplex;
```

stiamo comunicando ad AMPL di voler utilizzare il solutore CPLEX. Quest'ultimo è un solutore per problemi di Programmazione Lineare (e non solo) e può quindi essere utilizzato per risolvere il problema riportato nell'esempio.
A questo punto è sufficiente dare il comando per risolvere il problema che è

`solve;`

Digitando questo comando al promtp dei comandi, ovvero

```
ampl: solve;
```

si ottiene il seguente messaggio:

```
CPLEX 12.6.1.0: optimal solution; objective 1;
0 dual simplex iterations (0 in phase I)
```

con il quale AMPL ci comunica che il solutore ha determinato una soluzione ottima con valore della funzione obiettivo pari a 1. Per vedere il valore delle variabili all'ottimo è necessario un altro comando, ovvero

`display` *nome_variabile*;

Quindi digitando, ad esempio,

```
ampl: display x1;
```

otteniamo il valore di x_1^*. Analogamente per l'altra variable x_2^*. Abbiamo così ottenuto il punto di massimo che stavamo cercando.

Scrivere il modello utilizzando la linea di comando, ovviamente, è piuttosto scomodo, soprattutto perché in questo modo, una volta usciti da AMPL il modello

è completamente perso. Comviene, pertanto, scrivere il modello in un file testo che deve avere estensione `.mod`. Quindi utilizzando un qualsiasi editor di testo possiamo riscrivere il modello nel seguente modo:

esempio.mod

```
var x1;
var x2;

maximize funzione-obiettivo: x1 + x2;

subject to vincolo1: x1 + x2 <= 1;
subject to vincolo2: x1 - x2 <= 2;
subject to x1_non_neg: x1 >= 0;
subject to x2_non_neg: x2 >= 0;
```

A questo punto, dal prompt di AMPL è sufficiente scrivere il seguente comando per leggere il file del modello:

`model` *<PATH>\nome_file.mod*

Quindi, assumendo che la directory dove risiede il file del modello `esempio.mod` sia *C:\MODELLI*, digitando,

```
ampl: model C:\MODELLI\esempio.mod;
```

carichiamo il modello. A questo punto, si procede come descritto in precedenza per risolvere il problema e stampare i risultati ottenuti.

Si può inoltre creare un file contenente i comandi da dare (in modo da non doverli digitare ogni volta). Tale file deve avere estensione `.run`. Un esempio di questo file relativamente all'esempio fino ad ora esaminato è il seguente:

esempio.run

```
reset;
model C:\MODELLI\esempio.mod;
option solver cplex;
solve;
display x1;
display x2;
display funzione_obiettivo;
```

Si noti che nel file è stato aggiunto il comando `reset;` che ha la funzione di eliminare da AMPL dati relativi ad un modello precedentemente risolto. È conveniente introdurlo all'inizio di ogni file `.run`.
Per lanciare il file `.run` *nell'ambiente* AMPL si utilizza il comando

`include` <*PATH*> \ *nome_file.run*

Alternativamente, *fuori dell'ambiente* AMPL è sufficiente avere tale file nella directory che contiene AMPL (oppure in un'altra directory se si è introdotto nel *PATH* il percorso dove risiede AMPL), aprire un terminale del prompt dei comandi in tale directory e dare il comando `ampl esempio.run`.

Infine, per uscire dall'ambiente AMPL è sufficiente il comando `quit;`.

2.3 I SOLUTORI

Abbiamo già menzionato il fatto che alcuni solutori sono disponibili nell'istallazione Demo versiondi AMPL. Essi sono:

- CPLEX
- GUROBI
- LPSOLVE
- MINOS.

CPLEX.
Risolve problemi di Programmazione Lineare e problemi di Programmazione Quadratica convessi utilizzando il metodo del simplesso e metodi a punti interni. Inoltre risolve anche problemi di Programmazione Lineare e problemi di Programmazione Quadratica convessi con variabili intere utilizzando procedure di tipo Branch-and-Bound. La versione distribuita con la Demo version di AMPL è limitata a 500 variabili e 500 vincoli.

GUROBI.
Risolve problemi di Programmazione Lineare con il metodo del simplesso e con metodi a punti interni. Risolve inoltre problemi di Programmazione Lineare Misti Interi utilizzanzo procedure di tipo Branch-and-Bound. Risolve inoltre anche problemi di Programmazione Programmazione Quadratica e problemi di Programmazione Quadratica Misti Interi. Utilizzando la "student version di AMPL, GUROBI limita la dimensione dei problemi a 500 variabili e 500 vincoli.

LPSOLVE.
Risolve problemi di Programmazione Lineare e Programmazione Lineare Intera di dimensioni moderate.

MINOS.
Risolve problemi di Programmazione Lineare attraverso il metodo del simplesso e problemi di Programmazione Non Lineare utilizzando metodi di tipo gradiente ridotto.

Esistono altri solutori dei quali è stato previsto l'uso con AMPL, ma non tutti sono disponibili gratuitamente.

2.4 ALCUNI ESEMPI DI MODELLI DI PROGRAMMAZIONE LINEARE

Esaminiamo ora alcuni semplici modelli di Programmazione Lineare costruendo prima la formulazione algebrica e poi realizzandone un'implementazione in AMPL.

Esempio 2.4.1 *Un'industria chimica fabbrica 4 tipi di fertilizzanti,* **Tipo 1**, **Tipo 2**, **Tipo 3**, **Tipo 4**, *la cui lavorazione è affidata a due reparti dell'industria: il reparto produzione e il reparto confezionamento. Per ottenere fertilizzante pronto per la vendita è necessaria naturalmente la lavorazione in entrambi i reparti. La tabella che segue riporta, per ciascun tipo di fertilizzante i tempi (in ore) necessari di lavorazione in ciascuno dei reparti per avere una tonnellata di fertilizzante pronto per la vendita.*

	Tipo 1	**Tipo 2**	**Tipo 3**	**Tipo 4**
Reparto produzione	2	1.5	0.5	2.5
Reparto confezionamento	0.5	0.25	0.25	1

Dopo aver dedotto il costo del materiale grezzo, ciascuna tonnellata di fertilizzante dà i seguenti profitti (prezzi espressi in Euro per tonnellata)

	Tipo 1	**Tipo 2**	**Tipo 3**	**Tipo 4**
profitti netti	250	230	110	350

Determinare le quantità che si devono produrre settimanalmente di ciascun tipo di fertilizzante in modo da massimizzare il profitto complessivo, sapendo che ogni settimana, il reparto produzione e il reparto confezionamento hanno una capacità lavorativa massima rispettivamente di 100 e 50 ore.

Si tratta di un problema di pianificazione della produzione industriale in cui le incognite, che saranno le variabili del problema, sono le quantità di fertilizzante di ciascun tipo che si devono produrre. Costruiamo un modello di Programmazione Matematica rappresentante il problema in analisi supponendo di voler pianificare la produzione settimanale.

– *Variabili di decisione.* È naturale introdurre le variabili reali x_1, x_2, x_3, x_4 rappresentanti rispettivamente le quantità di prodotto del **Tipo 1**, **Tipo 2**, **Tipo 3**, **Tipo 4** da fabbricare in una settimana.

– *Funzione Obiettivo.* Ciascuna tonnellata di fertilizzante contribuisce al profitto totale secondo la tabella data. Quindi il profitto totale sarà

$$250x_1 + 230x_2 + 110x_3 + 350x_4. \tag{2.4.1}$$

L'obiettivo dell'industria sarà quello di scegliere le variabili x_1, x_2, x_3, x_4 in modo che l'espressione (2.4.1) del profitto sia massimizzata. La (2.4.1) rappresenta la funzione obiettivo.

– *Vincoli.* Ovviamente la capacità produttiva della fabbrica limita i valori che possono assumere le variabili x_j, $j = 1, \ldots, 4$; infatti si ha una capacità massima lavorativa in ore settimanali di ciascun reparto. In particolare per il reparto produzione si hanno a disposizione al piú 100 ore settimanali e poiché ogni tonnellata di fertilizzante di **Tipo 1** utilizza il reparto produzione per 2 ore, ogni tonnellata di fertilizzante di **Tipo 2** utilizza il reparto produzione per 1.5 ore e così via per gli altri tipi di fertilizzanti si dovrà avere

$$2x_1 + 1.5x_2 + 0.5x_3 + 2.5x_4 \leq 100. \tag{2.4.2}$$

Ragionando in modo analogo per il reparto confezionamento si ottiene

$$0.5x_1 + 0.25x_2 + 0.25x_3 + x_4 \leq 50. \tag{2.4.3}$$

Le espressioni (2.4.2), (2.4.3) costituiscono i vincoli del modello. Si devono inoltre esplicitare vincoli dovuti al fatto che le variabili x_j, $j = 1, \ldots 4$ rappresentando quantità di prodotto non possono essere negative e quindi vanno aggiunti i vincoli di non negatività

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0, \quad x_4 \geq 0.$$

La formulazione finale quindi può essere scritta in questa forma

$$\begin{cases} \max\left(250x_1 + 230x_2 + 110x_3 + 350x_4\right) \\ 2x_1 + 1.5x_2 + 0.5x_3 + 2.5x_4 \leq 100 \\ 0.5x_1 + 0.25x_2 + 0.25x_3 + x_4 \leq 50 \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0. \end{cases}$$

Si riporta di seguito il file `.mod` che implementa questo modello:

fertilizzanti.mod

```
var x1;
var x2;
var x3;
var x4;

maximize profitto: 250*x1+230*x2+110*x3+350*x4;

subject to vincolo1: 2*x1+1.5*x2+0.5*x3+2.5*x4 <= 100;
s.t. vincolo2: 0.5*x1+0.25*x2+0.25*x3+x4<= 50;
s.t. x1_nonneg: x1 >=0;
s.t. x2_nonneg: x2 >=0;
s.t. x3_nonneg: x3 >=0;
s.t. x4_nonneg: x4 >=0;
```

Esempio 2.4.2 *Un'industria manifatturiera possiede due impianti di produzione e fabbrica due tipi di prodotti* $\mathbf{P_1}$ *e* $\mathbf{P_2}$ *utilizzando due macchine utensili: una per la levigatura e una per la pulitura. Per avere un prodotto finito è necessaria l'utilizzazione di entrambe le macchine. Il primo impianto ha una disponibilità massima settimanale di 80 ore della macchina per la levigatura e di 60 ore della macchina per la pulitura. Le disponibilità orarie delle due macchine nel secondo impianto sono rispettivamente di 60 e 70 ore settimanali. La tabella che segue riporta, per ciascun prodotti, il numero di ore di lavorazione necessarie su ciascuna macchina per ottenere un prodotto finito (poiché le macchine possedute dal secondo impianto sono piú vecchie, i tempi di utilizzo sono maggiori)*

	Impianto 1		Impianto 2	
	$\mathbf{P_1}$	$\mathbf{P_2}$	$\mathbf{P_1}$	$\mathbf{P_2}$
levigatura	*4*	*2*	*5*	*3*
pulitura	*2*	*5*	*5*	*6*

Inoltre ciascuna unità di prodotto utilizza 4 Kg di materiale grezzo. Il profitto netto (in Euro) ottenuto dalla vendita di una unità di Prodotti $\mathbf{P_1}$ *e* $\mathbf{P_2}$ *è rispettivamente di 10 e 15.*

(a) *Costruire un modello lineare che permetta di massimizzare il profitto complessivo ottenuto dalla vendita dei prodotti in ciascun impianto sapendo che settimanalmente l'industria decide di assegnare 75 Kg di materiale grezzo al primo impianto e 45 Kg di materiale grezzo al secondo impianto.*

(b) *Costruire un modello lineare che permetta di massimizzare il profitto complessivo ottenuto dalla vendita dei prodotti supponendo che l'industria non allochi a priori 75 Kg di materiale grezzo nel primo impianto e di 45 Kg di materiale grezzo nel secondo impianto, ma lasci al modello la decisione di come ripartire tra i due impianti 120 Kg complessivi disponibili di questo materiale grezzo.*

Si riportano sinteticamente le formulazioni nei due casi.

Formulazione del caso (a)
Questo caso, nella pratica, corrisponde a costruire due modelli indipendenti: uno riferito al primo impianto, uno riferito al secondo impianto. Una "risorsa" (il materiale grezzo) è già allocata a priori.

Impianto 1: La formulazione relativa al primo impianto è:

$$\begin{cases} \max(10x_1 + 15x_2) \\ 4x_1 + 4x_2 \leq 75 \\ 4x_1 + 2x_2 \leq 80 \\ 2x_1 + 5x_2 \leq 60 \\ x_1 \geq 0, x_2 \geq 0 \end{cases}$$

A questo punto, prima di passare all'impianto 2, vediamo come fare per scrivere in AMPL il precedente problema di PL. Per fare questo è sufficiente creare il seguente file di modello (un semplice file di testo ma con estensione `.mod`) ad esempio `impianto1.mod`.

impianto1.mod

```
var x1;
var x2;

maximize profitto: 10*x1 + 15*x2;

subject to   m_grezzo: 4*x1 + 4*x2 <= 75;
subject to levigatura: 4*x1 + 2*x2 <= 80;
s.t.         pulitura: 2*x1 + 5*x2 <= 60;
s.t.         x1_non_neg: x1 >= 0;
s.t.         x2_non_neg: x2 >= 0;
```

Risolvendo il problema, si ottiene che all'ottimo l'impianto 1 ha un profitto di 225 Euro, ottenuto fabbricando $11,25$ unità di $\mathbf{P_1}$ e $7,5$ di $\mathbf{P_2}$. In particolare si è utilizzato tutto il materiale grezzo e sono state sfruttate tutte le ore di pulitura a disposizione. Le macchine per la levigatura, invece, sono state sottoutilizzate essendo avanzate 20 ore di tempo macchina.

Impianto 2: La formulazione relativa al secondo impianto è:

$$\begin{cases} \max(10x_3 + 15x_4) \\ 4x_3 + 4x_4 \leq 45 \\ 5x_3 + 3x_4 \leq 60 \\ 5x_3 + 6x_4 \leq 75 \\ x_3 \geq 0, x_4 \geq 0 \end{cases}$$

Il modello in AMPL per l'impianto 2 è:

impianto2.mod

```
var x3;
var x4;

maximize profitto: 10*x3 + 15*x4;
```

```
subject to   m_grezzo: 4*x3 + 4*x4 <= 45;
subject to levigatura: 5*x3 + 3*x4 <= 60;
s.t.         pulitura: 5*x3 + 6*x4 <= 75;
s.t.         x3_non_neg: x3 >= 0;
s.t.         x4_non_neg: x4 >= 0;
```

Risolvendo il problema si ha che all'ottimo l'impianto 2 ha un profitto di $168,75$ Euro ottenuto fabbricando $11,25$ unità di $\mathbf{P_2}$ e nessuna unità di $\mathbf{P_1}$. In particolare è stato utilizzato tutto il materiale grezzo ma, sia le macchine per la levigatura che quelle per la pulitura, sono state sottoutilizzate essendo avanzate rispettivamente $26,25$ e $7,5$ ore di tempo macchina.

Formulazione del caso (b)
Questo caso corrisponde a costruire un unico modello comprendente entrambi gli impianti. L'allocazione della "risorsa" data dal materiale grezzo è lasciata al modello stesso.

La formulazione relativa a questo caso è:

$$\begin{cases} \max & (10x_1 & + & 15x_2 & + & 10x_3 & + & 15x_4) & & \\ & 4x_1 & + & 4x_2 & + & 4x_3 & + & 4x_4 & \leq & 120 \\ & 4x_1 & + & 2x_2 & & & & & \leq & 80 \\ & 2x_1 & + & 5x_2 & & & & & \leq & 60 \\ & & & & & 5x_3 & + & 3x_4 & \leq & 60 \\ & & & & & 5x_3 & + & 6x_4 & \leq & 75 \\ & x_1 \geq 0, & & x_2 \geq 0, & & x_3 \geq 0, & & x_4 \geq 0 & & \end{cases}$$

in AMPL si avrà:

impianto1e2.mod

```
var x1;
var x2;
var x3;
var x4;

maximize profitto: 10*(x1+x3) + 15*(x2+x4);

s.t. m_grezzoTOT:   4*(x1+x2+x3+x4) <= 120;
s.t. levigatura1:   4*x1 + 2*x2 <= 80;
s.t. pulitura1:     2*x1 + 5*x2 <= 60;
```

```
s.t. levigatura2:   5*x3 + 3*x4 <= 60;
s.t. pulitura2:     5*x3 + 6*x4 <= 75;

s.t. x1_non_neg:        x1 >= 0;
s.t. x2_non_neg:        x2 >= 0;
s.t. x3_non_neg:        x3 >= 0;
s.t. x4_non_neg:        x4 >= 0;
```

Risolvere questo problema otteniamo un profitto complessivo di $404,17$ Euro ottenuto fabbricando nell'impianto 1: $9,17$ unità di $\mathbf{P_1}$ e $8,33$ unità di $\mathbf{P_2}$ e fabbricando, nell'impianto 2, solo $12,5$ unità di $\mathbf{P_2}$. È importante notare i seguenti punti:

- il profitto totale in questo caso ($404,17$ Euro) è superiore alla somma dei profitti ottenuti considerando separatamente i due modelli;
- l'impianto 1 ora usa 70Kg di materiale grezzo (contro i 75Kg del caso precedente) mentre l'impianto 2 ne usa 50Kg (contro i 45Kg del caso precedente).

3

Gli insiemi e i parametri in AMPL

AMPL permette di scrivere il modello in forma parametrica. Questo, nella sostanza, significa scrivere il modello nel file `.mod` senza specificare i dati che vengono invece scritti separatamente in un file `.dat`. Praticamente nel file del modello si effettuano le *dichiarazioni*, mentre nel file dei dati si effettuano le *assegnazioni*. Fatto questo, dal prompt di AMPL oltre al comando

`model` *<PATH>\nome_file.mod*

utilizzato per caricare il modello, sarà necessario un comando per caricare il file dei dati. Tale comando è:

`data` *<PATH>\nome_file.dat*

Quindi, assumendo che la directory dove risiedono i file del modello (`esempio.mod`) e dei dati (`esempio.dat`) sia *C:\MODELLI*, digitando anche

```
ampl: data C:\MODELLI\esempio.dat;
```

carichiamo anche i dati del modello. A questo punto, si procede come descritto in precedenza per risolvere il problema e stampare i risultati ottenuti.

Per introdurre l'uso di insiemi e parametri, consideriamo di nuovo i modelli presentati nell'Esempio 2.4.2 del capitolo precedente. Infatti, AMPL consente di scrivere il problema in modo diverso, utilizzando un concetto molto utilizzato in AMPL: gli *insiemi*. Grazie ad esso, è possibile tenere separati il modello dai dati. Il modello relativo all'impianto 1 può essere infatti riscritto come segue:

impianto.mod

```
set Prodotti;
set Risorse;

param q_max{Risorse} >=0;
param richiesta{Risorse,Prodotti} >=0;
param prezzo{Prodotti} >=0;

var x{Prodotti} >=0;

maximize profitto: sum{i in Prodotti} prezzo[i]*x[i];

s.t. vincolo_risorsa {j in Risorse}:
sum{i in Prodotti} richiesta[j,i]*x[i] <= q_max[j];
```

Ora analiziamo le istruzione del file `impianto.mod`. Anzitutto notiamo le istruzioni

```
set Prodotti;
set Risorse;
```

con le quali si dichiarano `Prodotti` e `Risorse` come insiemi di un numero imprecisato di elementi (prodotti e risorse). Subito dopo abbiamo tre istruzioni che ci servono per definire altrettanti parametri del modello. La prima di queste

```
param q_max{Risorse}>=0;
```

definisce un vettore di parametri con tante componenti quanti sono gli elementi dell'insieme `Risorse` e definisce le quantità massime disponibili di ciascuna risorsa.
Il $>= 0$ specifica che i valori immessi devono essere non negativi, AMPL eseguirà automaticamente il controllo sui dati.
L'istruzione successiva ovvero

```
param richiesta{Risorse,Prodotti}>=0;
```

definisce invece una matrice di parametri con tante righe quanti sono le risorse e tante colonne quante sono i prodotti, specificando per ogni prodotto la quantità di risorsa necessaria alla sua realizzazione.

L' istruzione

```
param prezzo{Prodotti} >=0;
```

definisce un vettore di parametri con tante componenti quanti sono gli elementi dell'insieme `Prodotti` specificando il prezzo di ogni prodotto.

La funzione obiettivo `profitto` è definita dalla istruzione

```
maximize profitto: sum{i in Prodotti} prezzo[i]*x[i];
```

I vincoli sono definiti atttraverso l'istruzione che segue:

```
s.t. vincolo_risorsa {j in Risorse}:
     sum{i in Prodotti} richiesta[j,i]*x[i] <= q_max[j];
```

Con essa si definiscono tanti vincoli quanti sono gli elementi dell'insieme `Risorse`.

A questo punto, completato il file della definizione del modello (`.mod`) è necessario definire un file con estensione `.dat` dove vengono specificati i valori per gli insiemi e i parametri del modello. Per quanto riguarda l'impianto 1 si avrà:

impianto1.dat

```
set Prodotti := P1 P2;
set Risorse  := m_grezzo levigatura pulitura;

param         q_max :=
m_grezzo        75
levigatura      80
pulitura        60;

param  richiesta: P1   P2 :=
m_grezzo          4     4
levigatura        4     2
pulitura          2     5;

param  prezzo :=
P1       10
P2       15;
```

Facciamo notare che in questo modo, avendo cioè a disposizione il file `impianto.mod` contenente il modello separato dai dati del problema (contenuti invece nel file `impianto1.dat`), possiamo risolvere differenti problemi di massimizzazione del profitto semplicemente cambiando i dati contenuti nel file con estensione `.dat`.

Sfruttando questo fatto è banale risolvere il problema relativo all'impianto 2: si utilizza lo stesso file `.mod` e un diverso file `.dat`, che è il seguente:

impianto2.dat

```
set Prodotti := P1 P2;
set Risorse  := m_grezzo levigatura pulitura;

param         q_max :=
m_grezzo        45
levigatura      60
pulitura        75;


param  richiesta: P1   P2 :=
m_grezzo          4     4
levigatura        5     3
pulitura          5     6;


param  prezzo :=
P1       10
P2       15;
```

Nello stesso modo, cioè specificando il solo file dei dati, è possibile risolvere il problema completo:

impianto.dat

```
set Prodotti :=
       P1_impianto1
       P2_impianto1
       P1_impianto2
       P2_impianto2;

set Risorse :=
       m_grezzo
       levigatura1
       pulitura1
       levigatura2
       pulitura2;
```

```
param         q_max :=
      m_grezzo         120
      levigatura1      80
      pulitura1        60
      levigatura2      60
      pulitura2        75;


param  richiesta: P1_impianto1   P2_impianto1  P1_impianto2  P2_impianto2:=
 m_grezzo           4                4             4               4
 levigatura1        4                2             0               0
 pulitura1          2                5             0               0
 levigatura2        0                0             5               3
 pulitura2          0                0             5               6;


param          prezzo :=
        P1_impianto1       10
        P2_impianto1       15
        P1_impianto2       10
        P2_impianto2       15;
```

Si può ovviamente modificare il file `.run` scritto in precedenza aggiungendo l'istruzione necessaria per caricare il file dei dati.

È importante ribadire che nel file `.mod` vengono effettuate le *dichiarazioni* di insiemi e parametri, mentre nel file `.dat` vengono effettuate le *assegnazioni*.

Esaminiamo nel dettaglio nel paragrafi che seguono gli elementi base della sintassi di AMPL riguardante *insiemi*, *parametri*, *variabili*, *funzione obiettivo* e *vincoli*.

3.1 GLI INSIEMI

Gli insiemi definiscono gli elementi di base con i quali si possono indicizzare variabili, parametri e vincoli del modello.

La dichiarazione di un *insieme generico* (nel file `.mod`) si effettua come segue:

```
set NomeInsieme;
```

L'assegnazione di un *insieme generico* (nel file `.dat`) si effettua come segue:

```
set NomeInsieme := e1 e2 e3 e4 e5 ;
```

Questa istruzione specifica che l'insieme `NomeInsieme` è composto dagli elementi $\{e_1, e_2, e_3, e_4, e_5\}$. Gli insiemi così definiti corrispondono agli insiemi generici. Esiste la possibilità di definire altri tipi di insiemi:

- *insiemi ordinati*

  ```
  set NomeInsieme ordered;
  ```

- *insiemi numerici*

  ```
  set NomeInsieme := 1 .. N;
  set NomeInsieme := 1 .. N by p;
  ```

- *insiemi ordinati e ciclici*

  ```
  set NomeInsieme circular;
  ```

Per quanto riguarda gli insiemi numerici la notazione `NomeInsieme := 1 .. N` definisce tutti i numeri interi tra 1 e N, mentre `NomeInsieme := 1 .. N by p` definisice tutti i numeri interi tra 1 e N distanti fra di loro di p numeri. Si osservi, quindi, che per quanto riguarda gli insiemi numerici la dichiarazione di fatto è un'assegnazione e quindi l'assegnazione di un insieme numerico *non* deve essere ripetuta nel file `.dat`.

Riportiamo ora gli operatori e le funzioni più comuni tra due **insiemi generici**:

Operatore/Funzione	**Significato**
`A union B`	insieme di elementi che stanno in A o B
`A inter B`	insieme di elementi che stanno sia in A che in B
`A within B`	A sottoinsieme di B
`A diff B`	insieme di elementi che stanno in A ma non in B
`A symdiff B`	insieme di elementi che stanno in A o in B ma non in entrambi
`card(A)`	numero di elementi che stanno in A

Di seguito gli operatori più comuni utilizzati in un **insieme ordinato** A:

Funzione	Significato
`first(A)`	primo elemento di A
`last(A)`	ultimo elemento di A
`next(a,A)`	elemento di A dopo a
`prev(a,A)`	elemento di A prima di a
`next(a,A,k)`	k-esimo elemento di A dopo a
`prev(a,A,k)`	k-esimo elemento di A prima di a
`ord(a,A)`	posizione di a in A
`ord0(a,A)`	come `ord(a,A)` ma restituisce 0 se a non è in A
`member(k,A)`	elemento di A in k-esima posizione

3.1.1 Gli insiemi multidimensionali

In AMPL è possibile definire insiemi a più dimensioni. Tale dimensione deve essere specificata nella dichiarazione di insieme (file `.mod`). La dicharazione

```
set INSIEME dimension p;
```

si usa per indicare che l'insieme `INSIEME` è costituito da $p-$uple *ordinate*, ovvero i suoi elementi sono della forma $(a_1, a_2, \ldots, a_p)$. Un esempio potrebbe essere:

```
set TERNE dimension 3;
```

che indica che l'insieme `TERNE` è formato da tutte terne ordinate di valori che verranno poi specificate nel file `.dat`, ad esempio, nel seguente modo:

```
set TERNE := (a,b,c) (d,e,f) (g,h,i) (l,m,n);
```

oppure

```
set TERNE :=
a b c
d e f
g h i
l m n;
```

Un modo alternativo di ottenere insiemi multidimensionali di dimesnione p è l'utilizzazione del prodotto cartesiano di p insiemi. Il prodotto cartesiano in AMPL si indica con `cross`. Quindi se, ad esempio, A, B e C sono stati già dichiarati come insiemi, l'istruzione

```
set TERNE := A cross B cross C;
```

indica l'insieme delle terne ordinate (a, b, c) con $a \in A$, $b \in B$ e $c \in C$.

Partendo da un insieme multidimensionale è possibile ottenere gli insiemi componenti l'insieme multidimensionale effettuando un procedimento inverso al precedente. Con riferimento all'esempio precedente, dall'insieme `TERNE` si possono ottenere i tre insiemi di partenza nel seguente modo con l'uso di `setof`:

```
set A := setof{(i,j,k) in TERNE} i;
set B := setof{(i,j,k) in TERNE} j;
set C := setof{(i,j,k) in TERNE} k;
```

È inoltre possibile definire insiemi in modo implicito, ovvero senza specificarne il nome, ma solo indicando la loro espressione. Ad esempio, dati A e B due insiemi,

```
{A,B}
{a in A, b in B}
```

sono due espressioni equivalenti per definire il prodotto cartesiano $A \times B$, ovvero l'insieme di tutte le coppie ordinate (a, b) con $a \in A$ e $b \in B$. Un altro esempio è dato dall'espressione

```
{a in A : prezzo[a]>=5}
```

che sta ad indicare tutti gli elementi a dell'insieme A tali che il `prezzo` di a sia maggiore o uguale a 5, dove, come vedremo più avanti, `prezzo` è un parametro indicizzato sull'insieme A.

3.2 I PARAMETRI

I parametri rappresentano i dati di un problema. Una volta che essi vengono assegnati (nel file `.dat`), essi non vengono in nessun caso modificati dal solutore. Un parametro deve essere *dichiarato* nel file `.mod` e *assegnato* nel file `.dat`. L'istruzione

```
param t;
```

utilizzata nel file `.mod` effettua la dichiarazione del parametro t. È possibile anche dichiarare vettori e matrici di parametri. Quindi se `PROD` e `ZONA` sono due insiemi le istruzioni

```
set PROD;
set ZONA;
param T;
param costi{PROD};
param prezzo{PROD,ZONA};
param domanda{PROD,ZONA,1..T};
```

oltre che dichiarare gli insiemi, definiscono:

- il parametro `T`;
- il parametro `costi` indicizzato dall'insieme `PROD`, ovvero `costi` è un vettore che ha tante componenti quanti sono gli elementi di `PROD`;
- il parametro a due dimensioni `prezzo`, indicizzato dagli insiemi `PROD` e `ZONA`;
- il parametro a tre dimensioni `domanda`, indicizzato dagli `PROD`, `ZONA` e dall'insieme dei numeri interi che vanno da 1 a T.

L'assegnazione dei parametri avviene nel file `.dat`. Un esempio di assegnazione dei parametri prima introdotti è il seguente:

```
set PROD := p1 p2;
set ZONA := z1 z2;
param T := 2;

param costi :=
   p1 5
   p2 4;

param prezzo: z1    z2  :=
   p1          2     7
   p2          5     9;

param domanda:=
  [*,*,1] :   z1    z2:=
     p1       10    15
     p2       13    22

  [*,*,2] :   z1    z2:=
     p1       32    25
     p2       18    15;
```

Si osservi innanzitutto, che la scelta di scrivere i dati incolonnati ha il solo scopo di rendere il file più leggibile: nessuna formattazione particolare è richiesta da AMPL. Inoltre è opportuno soffermarci sull'uso dei ":"; infatti i ":" sono obbligatori se si assegnano valori a due o più vettori di parametri monodimensionali indicizzati sullo stesso insieme, come nel caso di `prezzo` e `domanda`.

Il parametro `domanda` può essere alternativamente assegnato nel seguente modo:

```
param : domanda :=
   p1  z1  1  10
   p1  z1  2  32
```

```
p1  z2  1  15
p1  z2  2  25
p2  z1  1  13
p2  z1  2  18
p2  z2  1  22
p2  z2  2  15;
```

Nella dichiarazione dei parametri (file `.mod`) si possono anche includere controlli o restrizioni sui parametri stessi. Ad esempio, le istruzioni

```
param T > 0;
param N integer, <= T;
```

controllano che il parametro T sia positivo e che il parametro N sia un numero intero minore o uguale a T. Esiste anche l'istruzione `check` per effettuare controlli contenenti espressioni logiche. Un esempio potrebbe essere il seguente:

```
set PROD;
param offertatot >0;
param offerta{PROD} >0;
check: sum{p in PROD} offerta[p]=offertatot;
```

Esiste, infine, la possibilità di dichiarare parametri "calcolati come nel seguente esempio:

```
set PROD;
param offerta{PROD};
param offertatot:=sum{p in PROD} offerta[p];
```

3.3 LE VARIABILI

Le variabili rappresentano le incognite del problema e il loro valore è calcolato dal solutore. Una volta che la soluzione è stata determinata, il valore delle variabili all'ottimo rappresenta la soluzione del problema Si osservi, quindi, la differenza con i parametri: questi ultimi vengono assegnati dall'utente e rimangono costanti; il valore delle variabili cambia durante le iterazioni del solutore. Alle variabili l'utente può eventualmemte (ma è del tutto opzionale) assegnare dei valori iniziali che sono poi modificati dal solutore.

La dichiarazione delle variabili è obbligatoria. Per default, una variabile è considerata *reale*, oppure può essere specificata *intera* o *binaria* (a valori in $\{0, 1\}$). L'esempio che segue riporta la dichiarazione di variabili con la specifica del tipo di variabili:

```
var x;
var n integer;
var d binary;
```

Analogamente a quanto avviene per i parametri, anche le variabili possono essere indicizzate da insiemi come riportato nel seguente esempio:

```
set PROD;
set OPERAI;
param dom{PROD};
var num{PROD} integer;
var assegnamento{PROD,OPERAI} binary;
```

Anche nel caso delle variabili si possono introdurre dei controlli contestualmente alla loro dichiarazione come riportato nell'esempio seguente:

```
set PROD;
param dom{PROD};
var x >=0;
var quantita{p in PROD} >=0, <= dom[p];
```

È possibile *fissare* una variabile ad un valore mediante l'istruzione `fix` che può essere utilizzata nel file `.dat` o nel file `.run` e NON nel file `.mod`. Quindi data la variabile x, l'istruzione

```
fix x :=4;
```

assegna il valore 4 ad x. In questo caso il solutore *non cambierà il valore di tale variabile* che verrà considerata fissata al valore assegnato. Esiste poi il comando opposto `unfix` che sblocca una variabile precedentemente fissata. Nel caso dell'esempio si avrebbe

```
unfix x;
```

Infine, c'è la possibilità di inizializzare una variabile ad un determinato valore con il comando `let` da utilizzare nel file `.dat` o nel file `.run` e NON nel file `.mod`. Se scriviamo

```
let x:= 10;
```

stiamo inizializzando la variabile x al valore 10. Questo vuole dire che l'algoritmo risolutivo assegnerà 10 come valore iniziale della variabile x, valore che sarà cambiato dal solutore nel corso delle sue iterazioni. Si osservi, quindi, la differenza fondamentale tra il "fixing" di una variabile che non permette di modificare il valore assegnato a quella variabile e l'assegnazione di un valore iniziale ad una variabile.

3.4 LA FUNZIONE OBIETTIVO E I VINCOLI

La funzione obiettivo è sempre presente in un modello di Programmazione Matematica e rappresenta ciò che vogliamo massimizzare o minimizzare. Essa deve essere specificata nel file del modello (`.mod`). La parola chiave del linguaggo AMPL per introdurre la funzione obiettivo è `minimize` o `maximize` a seconda che ci trovi di fronte ad un problema di minimizzazione o massimizzazione. La sintassi è

```
maximize nome_funzione_obiettivo :  espressione_aritmetica ;
```

oppure

```
minimize nome_funzione_obiettivo :  espressione_aritmetica ;
```

Quindi un esempio potrebbe essere:

```
maximize profitto_totale : sum{i in PROD} prezzo[i]*quantita[i];
```

I vincoli sono parte integrante di un modello di Programmazione Matematica e sono specificati anch'essi nel file del modello (`.mod`). La parola chiave è `subject to` che può essere abbreviata in `s.t.`. La sintassi è

```
subject to nome_vincolo :  espressione_aritmetica e/o logica;
```

Se x è una variabile reale, la più semplice espressione di un vincolo potrebbe essere

```
subject to vincolo : x >=0;
```

In realtà, questo tipo di vincoli semplici viene spesso inserito come restrizione nella dichiarazione della variabile x. Anche i vincoli possono essere indicizzati e quindi invece di scrivere tanti vincoli, in una sola espressione si possono esplicitare più vincoli. Un esempio di ciò è il seguente:

```
set PROD;
set REPARTI;

param orelavoro{PROD,REPARTI};
param maxore{REPARTI};
param prezzo{PROD};

var x{PROD};

maximize ricavo : sum{in PROD} prezzo[i]*x[i];
```

```
subject to vincoli{j in REPARTI} sum{i in PROD}
                        orelavoro[i,j]*x[i] <= maxore[j];
```

In questo esempio, con un unico vincolo si sono scritti tanti vincoli quanti sono gli elementi dell'insieme `REPARTI` al posto dei vincoli

```
subject to vincolo_1 : sum{i in PROD} orelavoro[i,"R1"]*x[i]
                            <= maxore["R1"];
subject to vincolo_2 : sum{i in PROD} orelavoro[i,"R2"]*x[i]
                            <= maxore["R2"];
   .
   .
   .
   .
subject to vincolo_m : sum{i in PROD} orelavoro[i,"Rm"]*x[i]
                            <= maxore["Rm"];
```

avendo supposto che l'insieme `REPARTI` sia composto dagli elementi $\{R_1, R_2, \ldots, R_m\}$. Quest'ultima scrittura non darebbe luogo a messaggi di errore in AMPL, ma oltre che essere molto lunga presenta l'ovvio inconveniente di dover conoscere già nel file del modello (`.mod`) quali sono gli elementi dell'insieme `REPARTI`. Questo contravviene al fatto che un *modello deve essere indipendente dai dati*; infatti, non utilizzando la scrittura indicizzata dei vincoli, un cambio degli elementi dell'insieme `REPARTI` (che ricordiamo è assegnato nel file dei dati (`.dat`)) non permetterebbe più al modello implementato di essere corretto.

3.5 LE ESPRESSIONI

Nella costruzione della funzione obiettivo e dei vincoli, così come nell'imporre condizioni sui parametri e sulle variabili si utilizzano espressioni aritmetiche, funzioni e operatori di diverso tipo. Abbiamo già riportato in precedenza i principali operatori e funzioni sugli insiemi. Le principali funzioni e operatori aritmetici e logici sono riportati nelle tabelle che seguono (per una elenco completo si fa riferimento al testo [Fourer et al., 2003]):

Funzione	Significato
`abs(x)`	valore assoluto di x
`sin(x)`	$\sin(x)$
`cos(x)`	$\cos(x)$
`tan(x)`	$\tan(x)$
`asin(x)`	$\arcsin(x)$
`acos(x)`	$\arccos(x)$
`atan(x)`	$\arctan(x)$
`exp(x)`	$\exp(x)$
`sqrt(x)`	radice quadrata di x, $\sqrt{x}$
`log(x)`	logaritmo naturale di x, $\ln(x)$
`log10(x)`	logaritmo in base 10 di x, $\log(x)$
`ceil(x)`	parte intera superiore di x, $\lceil x \rceil$
`floor(x)`	parte intera inferiore di x, $\lfloor x \rfloor$

Operatori aritmetici	Significato
`^`	potenza
`+`	somma
`-`	sottrazione
`*`	prodotto
`/`	divisione
`div`	divisione intera
`mod`	modulo
`sum`	sommatoria
`prod`	produttoria
`min`	minimo
`max`	massimo
$>$, $>=$	maggiore, maggiore o uguale
$<$, $<=$	minore, minore o uguale
`=`	$=$
`<>`, $!=$	diverso

Operatori logici	Significato
`not`	negazione logica
`or`	"or logico
`and`	"and logico
`exists`	quantificatore esistenziale logico
`forall`	quantificatore universale logico
`if then else`	espressione condizionale