Manuale essenziale di AMPL

Documento preparato da M. Bruglieri, R. Cordone, L. Liberti e C. Iuliano

Dipartimento di Elettronica e Informazione, Politecnico di Milano,
Piazza Leonardo da Vinci 32, 20133 Milano

Milano, Febbraio 2010

Indice

Capitolo 1

I generatori algebrici di modelli

Fra gli anni '50 e gli anni '70, la programmazione matematica compì progressi sostanziali, cui, tuttavia, non corrispose un'applicazione altrettanto diffusa a problemi reali. Un forte ostacolo fu la difficoltà pratica di stendere i modelli, raccogliere e organizzare i dati, programmare gli algoritmi risolutivi e analizzare i risultati ottenuti.

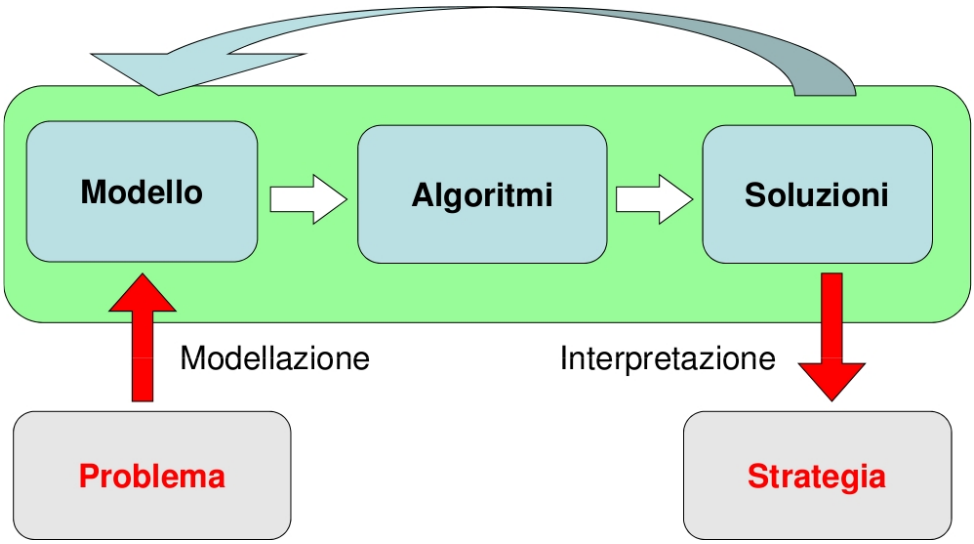


Figura 1.1: Schema dell'approccio modellistico per passare da un problema concreto a una strategia operativa

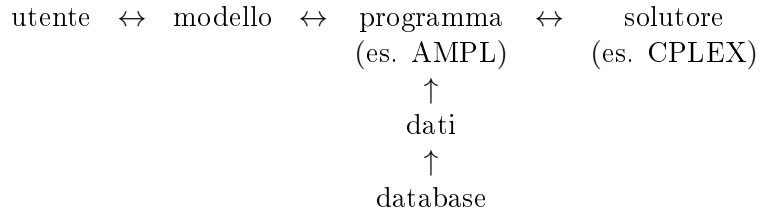
L'approccio modellistico, infatti, ha indubbi vantaggi, ma in esso, come ricorda la Figura 1.1, il passaggio dal problema alla strategia risolutiva è tutt'altro che immediato: si tratta di formulare prima un modello, poi di risolverlo, e infine di tradurre la soluzione in azioni concrete. Il passaggio dal problema al modello e quello dalla soluzione alla strategia operativa non sono banali e richiedono anzi la dose maggiore di creatività. Anche il passaggio intermedio (quello dal modello alla soluzione), però, è solo apparentemente astratto e matematico. Infatti, se si fa uso (come in genere avviene) di uno strumento informatico, questo passaggio non si riduce all'applicazione di un algoritmo, ma comprende:

1. la traduzione del modello (pensato o scritto in linguaggio matematico) e dei dati (disponibili su carta o in un database informatico) in strutture dati accessibili a un risolutore
2. la realizzazione di un risolutore, che trasformi i dati in una soluzione opportunamente codificata
3. la traduzione della soluzione in un formato accessibile all'utente (tabelle, grafici o altro).

Grandi successi in campo matematico e un'enorme potenza di calcolo possono dar luogo a risolutori molto potenti, ma questi sono poco utili a livello applicativo se l'utente non dispone di un'interfaccia comoda verso il risolutore, ovvero di un software che gestisca modelli e dati appartenenti al mondo reale e interroghi il risolutore. I *generatori algebrici di modelli*, fra cui AMPL, costituiscono quest'interfaccia, cioè si occupano del primo e del terzo passo, lasciando il secondo al risolutore. Le caratteristiche principali dei generatori algebrici di modelli sono:

- fornire un linguaggio semplice per descrivere modelli complessi, un linguaggio che sia contemporaneamente
  - *ad alto livello*, cioè comprensibile a un essere umano
  - *formalmente strutturato*, cioè accessibile a un risolutore
- permettere all'utente di comunicare con il risolutore attraverso file di testo anziché attraverso strutture dati, in modo da non richiedergli conoscenze informatiche approfondite e da consentirgli di formulare il modello con un semplice editor, qualunque sia la piattaforma su cui viene scritto e quella su cui viene risolto
- permettere all'utente di comunicare con diversi risolutori, in modo da poter sfruttare i più potenti sul mercato, ovvero quelli disponibili
- tenere distinti la *struttura logica* del modello (variabili di decisioni, obiettivi, vincoli e le loro relazioni) dal valore dei *dati numerici*.

Si può descrivere questa situazione come segue:



Utilizzando un generatore di modelli l'utente non si occupa più direttamente di interrogare il risolutore, ma può concentrarsi sulla stesura del modello, usando un linguaggio di alto livello che gli renda più semplice descrivere problemi reali anche molto complessi. L'utente può anche affidare la gestione dei dati a un database esterno cui accedere quando necessario e può sostituire il risolutore senza dover riscrivere il modello. Infine la separazione tra la struttura logica e i dati evita che piccole modifiche nella struttura del modello o nei dati comportino di riscrivere tutto: diventa infatti semplice usare lo stesso modello su dati differenti o applicare modelli diversi allo stesso problema.

Capitolo 2

Il linguaggio AMPL

AMPL è un linguaggio ad alto livello per descrivere modelli di Programmazione Matematica. In questo capitolo ne descriviamo la sintassi, a partire da alcune sue caratteristiche generali, per proseguire con i diversi elementi che costituiscono un tipico modello AMPL.

2.1 Caratteristiche generali

Un'istruzione di AMPL può essere dichiarativa o esecutiva. Le istruzioni esecutive servono a scrivere programmi in AMPL, mentre le istruzioni dichiarative servono a scrivere i modelli veri e propri.

2.1.1 Grammatica del file di modello

Lo schema sintattico generale delle istruzioni dichiarative è:

entità nome [*quantificatori*] [*specificazioni*] [: *espressione-matematica* | := *definizione*] ;

Le entità di un modello sono quelle specificate anche in Sezione 2.2: insiemi, parametri, variabili, funzione obiettivo, vincoli. L'elemento sintattico entità è una parola chiave che specifica l'entità di cui si parla nella dichiarazione, secondo la tabella sotto.

Entità	Parola chiave
insieme	**set**
parametro	**param**
variabile	**var**
funzione obiettivo	**minimize** o **maximize**
vincolo	**subject to**

Ogni entità, in AMPL, deve avere un nome. Il nome è una stringa di caratteri alfanumerici (che non può cominciare con una cifra né un segno di punteggiatura) che identifica univocamente

l'entità: quindi non ci possono essere due entità con lo stesso nome, anche se si tratta di entità di tipo diverso; ad esempio, un vincolo non può avere lo stesso nome di una variabile o della funzione obiettivo.

I quantificatori descrivono l'estensione (dimensionalità e cardinalità) dell'entità: si possono avere vettori, matrici e tensori di entità, come anche entità indicizzate secondo insiemi definiti dall'utente. Il quantificatore non è obbligatorio: se si sta definendo un unico vincolo (non una classe di vincoli), ad esempio, i quantificatori non sono necessari. Un quantificatore di solito segue lo schema sintattico:

{ [*indice1* **in**] *insieme1* , . . . , [*indiceN* **in**] *insiemeN* [: *condizioni*] }

dove *condizioni* indica una successione di equazioni o disequazioni. Verranno allora create le entità specificate, una per ogni indice specificato dal quantificatore, soltanto se le condizioni sono vere. Ad esempio, si può creare un insieme di variabili x_{ij} con $i < j$ e $i, j \in I$ con il comando:

```
var x{i in I, j in I : i < j};
```

È possibile omettere gli indici (e specificare solo gli insiemi) nel caso in cui gli indici non vengano più utilizzati fino alla fine della dichiarazione (segnalata dal punto e virgola). Ad esempio, se volessimo dichiarare un vettore di variabili y_i per $i \in I$, potremmo usare il costrutto

```
var y{i in I};
```

ma anche

```
var y{I};
```

dato che l'indice "i" sintatticamente non appare più fino al carattere ";". Le *specificazioni* sono una successione (separata da virgole) di condizioni alle quali l'entità deve soddisfare, come per esempio l'integralità o la non negatività di una variabile. Di solito le *specificazioni* sono usate per definire i limiti di una variabile, ma possono essere anche usate per definire l'intervallo di variazione di un parametro in modo che i dati possano essere verificati prima di essere usati. Per dichiarare una variabile z binaria si usa:

```
var z binary;
```

Per dichiarare un vettore di variabili $0 \leq x_j \leq 10, x_j \in \mathbb{Z}$, con $j \in J$, si usa:

```
var x{J} >= 0, <= 10, integer;
```

L'*espressione-matematica* serve a definire la funzione obiettivo e i vincoli. Nel caso della funzione obiettivo, è un'espressione matematica funzione delle variabili, parametri e indici validi

al momento della dichiarazione. Nel caso dei vincoli, è un vincolo in forma di equazione (=) o di disequazione (<= o >=). Entrambi i membri dell'equazione / disequazione sono delle espressioni matematiche funzioni delle variabili, parametri e indici validi al momento della dichiarazione. AMPL non restringe l'utilizzo della sintassi alle sole espressioni lineari ma un limite in questo senso può venire dal solutore numerico impiegato. Per esempio, i vincoli di conservazione di flusso

$$\sum_{(h,j)\in A} x_{hj} = \sum_{(i,h)\in A} x_{ih} \quad h \in V : h \neq s, h \neq t$$

sono dichiarati come:

```
subject to conservazione{h in V: h <> s and h <> t}:
sum{(h,j) in A} x[h,j] = sum{(i,h) in A} x[i,h];
```

La *definizione* serve a definire il valore di un insieme o di un parametro al momento della dichiarazione. Si preferisce utilizzare il file di dati per provvedere a questa necessità, tuttavia, quando si tratta di definizione numerica di insiemi o parametri.

2.1.2 Sensibilità alle maiuscole

AMPL è *case sensitive*, cioè distingue le lettere maiuscole dalla minuscole. Così le dichiarazioni

```
var x;
var X;
```

introducono due variabili distinte, denominate x e X. È buona norma non sfruttare questa possibilità.

2.1.3 Commenti

È possibile aggiungere alle istruzioni del programma commenti che le rendano più chiare. I commenti sono introdotti dal simbolo `#`, che indica al software di ignorare il seguito della riga.

2.1.4 Separatori

Le diverse istruzioni del modello sono separate dal simbolo `;`

```
var x1;
var x2;
minimize obiettivo: x1 + x2;
```

2.2 Struttura di un programma AMPL

Sebbene sia lecito scrivere in un solo file AMPL sia il modello sia i dati, è concettualmente preferibile tenere separati questi due termini, costruendo:

- un *file di modello*, di estensione **.mod**, che descrive la struttura del modello
- un *file di dati*, di estensione **.dat**, che contiene i valori numerici del problema

Mantenendo fisicamente separato il modello dai dati, è possibile applicare lo stesso modello a dati diversi, o cambiare i dati senza dover modificare il modello, evitando il rischio di introdurvi errori.

Gli elementi principali di un programma AMPL sono:

- *insiemi*
- *dati*
- *variabili*
- *funzione obiettivo*
- *vincoli*

2.3 Insiemi

Gli insiemi "descrivono il mondo" in cui ci si muove, cioè definiscono il dominio del problema e la sua dimensione. Ciascun insieme del modello va:

- *dichiarato* nel file **.mod**, per indicare che un certo nome rappresenta un insieme; si impiega la parola chiave **set**, seguita dal nome dell'insieme e dal separatore ;

```
set I;
```

- *definito* nel file **.dat**, per assegnare all'insieme gli elementi che ne fanno parte; si impiega la parola chiave **set**, seguita dal nome dell'insieme, dal simbolo := , dagli elementi separati da spazi o "a capo"; il separatore ; chiude l'istruzione.

```
set I := Calorie Proteine Calcio Ferro Vitamine;
```

Si presti attenzione alla differenza tra "dichiarare" e "definire", che nel seguito verrà sottolineata meno fortemente.

Oltre che simbolici, gli insiemi possono essere numerici. In tal caso, si possono definire come intervalli, indicandone solo gli estremi.

```
set Mesi := 1..12;
```

La parola chiave **by** consente di specificare la distanza fra due elementi successivi dell'insieme:

```
set MesiDispari := 1..12 by 2;
```

equivale a

```
set MesiDispari := 1 3 5 7 9 11;
```

Ovviamente, `1..12 by 1` equivale a `1..12`.

Le definizioni precedenti come intervalli avvengono direttamente nel file **.mod**. È possibile prevedere una dichiarazione di un intervallo dipendente da un parametro nel file **.mod**. nel seguente modo

```
param m;
set I := 1..m;
```

mentre il valore numerico del parametro viene definito nel file **.dat**.

2.3.1 Sottoinsiemi

Si può dichiarare un insieme come sottoinsieme di un altro attraverso la parola chiave **within**.

```
set Mesi;
set MesiEstivi within Mesi;
```

La definizione di **Mesi** e **MesiEstivi** dovrà essere coerente con questa dichiarazione, altrimenti AMPL segnalerà un errore.

2.3.2 Operazioni su insiemi

Dati due insiemi **A** e **B**, è possibile compiere su di loro diverse operazioni per definire insiemi derivati. Supponiamo che sia

```
set A := 1 3 5 7 9 11;
set B := 9 11 13 15 17;
```

Operazione	Risultato	Significato
A union B	1 3 5 7 9 11 13 15 17	elementi di **A** o di **B**
A inter B	9 11	elementi di **A** e di **B**
A diff B	1 3 5 7	elementi di **A** e non di **B**
A symdiff B	1 3 5 7 13 15 17	elementi di **A** o di **B**, ma non di entrambi
card(A)	6	numero degli elementi di **A**

2.3.3 Insiemi ordinati

In generale, gli insiemi sono *non ordinati*. È possibile dichiararli ordinati facendo seguire al nome dell'insieme la parola chiave **ordered**.

```
set A ordered;
```

Sugli insiemi ordinati si possono compiere ulteriori operazioni rispetto agli insiemi ordinati.

Operazione	Risultato	Significato
first(A)	1	primo elemento di **A**
last(A)	11	ultimo elemento di **A**
next(3,A,2)	7	secondo elemento di **A** dopo **3**
prev(5,A,2)	1	secondo elemento di **A** prima di **5**
next(3,A)	5	primo elemento di **A** dopo **3**
prev(5,A)	3	primo elemento di **A** prima di **3**
ord(5,A)	2	posizione di **5** in **A**
ord0(4,A)	0	come sopra, ma rende **0** se **4** non è in **A**
member(3,A)	5	terzo elemento di **A**

Dato un insieme **A** ordinato e un insieme **B** generico (ordinato o no), la loro differenza `A diff B` è ordinata, mentre gli altri insiemi derivati non lo sono.

Negli insiemi ordinati *circolari*, all'ultimo elemento segue il primo.

```
set A circular;
```

2.3.4 Collezioni di insiemi

Si può dichiarare un insieme di insiemi con la notazione

```
set Indice;
set Collezione{Indice};
```

che dichiara tanti insiemi quanti sono gli elementi dell'insieme `Indice`.

2.3.5 Insiemi pluridimensionali

Spesso è utile poter lavorare con insiemi pluridimensionali, vale a dire insiemi i cui elementi sono n-uple (si tratta sempre di n-uple *ordinate*). Gli insiemi pluridimensionali si dichiarano facendo seguire al nome dell'insieme la parola chiave **dimen** e il numero di termini della n-upla.

```
set TRIPLET dimen 3;
```

Si definiscono come gli insiemi monodimensionali, elencandone gli elementi. Questi vengono racchiusi fra parentesi tonde e i loro termini sono separati da virgole.

```
set TRIPLET := (1,7,1) (Giugno,Luglio,Agosto) (1,3,5) (3,1,5);
```

Si badi che gli elementi `(1,3,5)` e `(3,1,5)` possono coesistere in `TRIPLET` perché l'ordine delle loro componenti è significativo.

Quando un insieme pluridimensionale è il prodotto cartesiano di più insiemi monodimensionali, si può definire direttamente come tale, con la parola chiave **cross**

```
set X := 0 1 2 3;
set Y := 0 1 2;
set Z := 1 2 3;
set Punto3D := X cross Y cross Z;
```

L'ultima dichiarazione è equivalente a

```
set Punto3D := {X,Y,Z};
```

Questo equivale a definire `Punto3D` come $(0,0,1)(0,0,2)(0,0,3)(1,0,1)\ldots(3,2,3)$.

Viceversa, le parole chiave **setof** e **in** consentono di definire un insieme monodimensionale come la *proiezione* su un indice di un insieme pluridimensionale.

```
set TRIPLET dimen 3;
set X := setof{(i,j,h) in TRIPLET} i;
```

2.3.6 Espressioni di indicizzazione

È possibile definire insiemi non solo esplicitamente, assegnando loro gli elementi o costruendoli a partire da altri insiemi, ma anche implicitamente, attraverso le cosiddette *espressioni di indicizzazione*. In genere queste espressioni si usano per definire un insieme senza attribuirgli un nome.

```
{I}                         # elementi di A
{I,J}                       # coppie ordinate di elementi,
                            # tratti rispettivamente da I e da J
{i in I,j in J}             # coppie ordinate di elementi,
                            # tratti rispettivamente da I e da J
{i in I: c[i] >= 10}        # elementi di I tali che il valore
                            # associato nel vettore c è >= 10
```

L'insieme viene poi usato per fare da indice in una sommatoria o nella definizione di un vettore o una matrice di parametri, variabili, vincoli. In un'espressione di indicizzazione possono comparire, oltre agli insiemi componenti, anche gli indici che li scorrono. Questi sono necessari se l'espressione viene usata in un contesto nel quale è necessario poter individuare i singoli elementi dell'insieme (ad esempio, in una sommatoria, i cui termini dipendono dall'indice stesso).

```
sum{i in I: c[i] >= 10} x[i];
```

L'istruzione somma i valori del vettore `x` corrispondenti agli indici nell'insieme `I` i cui corrispondenti valori nel vettore `c` sono non inferiori a 10.

2.4 Parametri

I parametri sono i valori numerici che definiscono in dettaglio il problema che si vuole affrontare, una volta precisata la sua struttura. Essi sono assegnati una volta per tutte, al contrario delle variabili, che vengono modificate dal risolutore in modo da determinare la soluzione ottima al problema. Ovviamente, è possibile modificare i parametri generando problemi diversi, ma ciò avviene sempre a monte del processo risolutivo: il risolutore non può alterare i parametri.

2.4.1 Dichiarazione

Come gli insiemi, i parametri vengono dichiarati nel file del modello **.mod**. Per dichiararli, si impiega la parola chiave **param**, seguita dal nome del dato.

```
param n;
```

È possibile dichiarare vettori o matrici di parametri, facendo seguire al nome un'espressione di indicizzazione che stabilisca l'insieme cui corrispondono gli elementi del vettore o della matrice.

```
set I;
set J;

param c{J};
param a{I,J};
```

L'espressione di indicizzazione può esplicitare gli indici

```
param c{j in J};
param a{i in I,j in J};
```

ma non è necessario che lo faccia. Diventa invece necessario nel definire i vincoli, dato che in tal caso gli indici sono di solito ripresi nel corpo della definizione.

Per far riferimento al singolo dato, si usa la notazione con le parentesi quadre: `c[2]` è il valore associato al secondo elemento del vettore `c`, ovvero è il costo del secondo elemento dell'insieme `J`, mentre `a[i,j]` è il valore associato all'i-esimo nutriente e allo j-esimo cibo nella matrice **a**.

2.4.2 Definizione

Come gli insiemi, i parametri vengono definiti nel file dei dati **.dat**.

```
param n := 10;
```

Vettori e matrici si definiscono elencando le coppie indice-valore, separate da spazi o "a capo" (che spesso aumentano la leggibilità).

```
set J := Pane Latte Carne Verdure;

param c := Pane    10
           Latte    5
           Carne   20
           Verdure  8;
```

È possibile definire simultaneamente diversi vettori

```
param : Costi    Disponibilita :=
Pane       10          4
Latte       5          5
Carne      20          2
Verdure     8          3  ;
```

Si noti il simbolo `:` dopo la parola chiave **param**: serve a indicare che ciascuna delle parole seguenti è il nome di un vettore di parametri.

Con una sintassi analoga si possono definire le matrici colonna per colonna, anziché elencarne gli elementi uno per uno. Oltre all'indice di ogni colonna, bisogna riportare il nome della matrice.

```
param a: Calorie Proteine Calcio Ferro Vitamine :=
Pane       300       12      10     4      2
Latte      500       50     200    10     60
Carne      200      100      20    20     20
Verdure     50        3       5    15    100 ;
```

Se la matrice ha molte colonne e poche righe, per renderla più leggibile si può indicarne la trasposta, segnalandolo con la parola chiave **(tr)**

```
param a(tr): Pane Latte Carne Verdure :=
    Calorie   300   500   200      50
    Proteine   12    50   100       3
    Calcio     10   200    20       5
    Ferro       4    10    20      15
    Vitamine    2    60    20     100 ;
```

Per le matrici a più di due dimensioni, oltre che elencare gli elementi uno per uno, si possono riportare le "sezioni bidimensionali" ottenute fissando tutti gli indici tranne due. Ogni sezione è introdotta dall'indicazione degli indici fissati e di quelli rimasti liberi, racchiusa fra parentesi quadre. Nell'esempio che segue, una matrice tridimensionale viene sezionata rispetta alla seconda dimensione.

```
set DimX := I II III;
set DimY := 1 2;
set DimZ := a b c;
set Spazio := DimX cross DimY cross DimZ;

param Cubo :=
    [*,1,*] :   a   b   c :=
          I    10   4   3
          II    5   1  14
          III   3   2  20
    [*,2,*] :   a   b   c :=
          I    10   4   3
          II    5   1  14
          III   3   2  20 ;
```

Si notino gli asterischi che contraddistinguono gli indici non fissati e il fatto che il separatore **;** compare solo al termine e non fra una sezione e l'altra.

2.5 Variabili

Le variabili sono le grandezze che descrivono la soluzione del problema. Il loro valore deve essere determinato dal risolutore. Le variabili vengono dichiarate nel file del modello attraverso la parola chiave **var**, seguita dal nome della variabile ed, eventualmente, da restrizioni al suo valore. Queste ultime possono essere descritte da espressioni logiche oppure dalle parole chiave **integer** e **binary**

```
var x;
var y >= 0, <= SUP;
var z integer;
```

```
var w binary;
var v = 100;
```

dove `SUP` deve essere un dato dichiarato e definito. La variabile y deve rimanere nell'intervallo compreso fra 0 e `SUP`, la variabile z è intera e w può assumere solo i valori 0 e 1. Infine, alla variabile v viene assegnato un valore fisso: si tratta di un caso abbastanza degenere. Il linguaggio AMPL ammette quest'ultima possibilità per consentire esperimenti nei quali si fissa il valore di alcune variabili, lasciando le altre libere.

Anche le variabili si possono raccogliere in vettori o matrici grazie alle espressioni indicizzate.

```
set J;

param q{J};

var x{J} >=0;
var z{j in J} >= 0, <= q[j];
```

Queste istruzioni dichiarano due variabili vettoriali indicizzate sull'insieme `J`. Nella dichiarazione della prima variabile è equivalente indicare solo l'insieme di definizione della variabile (`x{J}`) o esplicitarne anche l'indice (`x{j in J}`). Nella seconda dichiarazione, invece, l'indice deve essere esplicito, perché compare anche in una restrizione e quest'ultima è diversa per ciascuna variabile nel vettore `q`.

Per far riferimento a una singola variabile in un vettore o matrice, si impiega la consueta notazione con le parentesi quadre.

```
x[i,j]
```

Benché il valore delle variabili sia sottratto al controllo dell'utente e affidato interamente al risolutore (entro i vincoli stabiliti dal modello) è consentito imporre alle variabili un valore iniziale. Questo è molto importante nei problemi non lineari, dato che influenza la soluzione finale che viene determinata.

```
var v := 100;
```

Si noti la differenza rispetto alla restrizione `var v = 100`, che obbliga la variabile v a rimanere pari a 100 sino alla fine.

2.6 Espressioni algebriche

La funzione obiettivo e i vincoli che la soluzione finale deve rispettare sono generalmente espressioni algebriche complesse, costruite a partire dai parametri e dalle variabili. Esse impiegano i seguenti operatori, posti in ordine di precedenza decrescente:

Operatore	Simbolo
Potenza	^ (oppure **)
Numero negativo	-
Somma	+
Sottrazione	-
Prodotto	*
Divisione	/
Divisione intera	div
Modulo	mod
Differenza non negativa (max(a-b,0))	less
Sommatoria	sum
Produttoria	prod
Minimo	min
Massimo	max
Unione di insiemi	union
Intersezione di insiemi	inter
Differenza di insiemi	diff
Differenza simmetrica di insiemi	symdiff
Prodotto cartesiano di insiemi	cross
Appartenenza a un insieme	in
Non appartenenza a un insieme	notin
Maggiore, Maggiore o uguale	>,>=
Minore, Minore o uguale	<,<=
Uguale	= (oppure ==)
Diverso	<> (oppure !=)
Negazione logica	not (oppure !)
And logico	and
Quantificatore esistenziale	exists
Quantificatore universale	forall
Or logico	or (oppure ||)
If then else	if then else

Possono inoltre impiegare le seguenti funzioni:

Operatore	Simbolo
Valore assoluto	abs(x)
Arco seno	asin(x)
Arco seno iperbolico	asinh(x)
Arco coseno	acos(x)
Arco coseno iperbolico	acosh(x)
Arco tangente iperbolica	atanh(x)
Seno	sin(x)
Seno iperbolico	sinh(x)
Coseno	cos(x)
Coseno iperbolico	cosh(x)
Tangente	tan(x)
Tangente iperbolica	tanh(x)
Approssimazione intera per difetto	floor(x)
Approssimazione intera per eccesso	ceil(x)
Logaritmo naturale	log(x)
Logaritmo decimale	log10(x)
Esponenziale	exp(x)
Radice quadrata	sqrt(x)
Minimo fra più numeri	min(x1,x2,...,xn)
Massimo fra più numeri	max(x1,x2,...,xn)

2.7 Funzione obiettivo

La funzione obiettivo specifica la grandezza del problema di cui si vuole trovare il valore ottimale. Viene introdotta dalla parola chiave **minimize** o **maximize**, seguita da un nome (obbligatorio), dal simbolo **:** e dall'espressione che definisce la funzione obiettivo in termini dei dati e delle variabili.

```
minimize CostoTotale : sum{j in Cibi} Costi[j] * Quantita[j];
```

È possibile definire più funzioni obiettivo (anche un vettore di funzioni obiettivo indicizzato su un insieme). In generale AMPL considera solo la prima e ignora le altre.

2.8 Vincoli

I vincoli distinguono le soluzioni ammissibili da quelle inammissibili. Vengono introdotti dalla parola chiave **subject to**, seguita da un nome (obbligatorio), dal simbolo **:**, da un'espressione e dal consueto separatore **;**. L'espressione consiste nel confronto di due espressioni algebriche attraverso un operatore di relazione (**<=**, **=** o **>=**)

```
subject to vincolo: x <= 1;
```

Anche i vincoli possono essere indicizzati

```
subject to fabbisogno{i in I}:
  sum{j in J} a[i,j] * x[j] >= b[i];
```

Questa istruzione dichiara un vincolo per ogni elemento dell'insieme `I`. Il vincolo impone per ogni fattore nutritivo nell'insieme `I` che la somma dei prodotti fra i coefficienti `a[i,j]` e le quantità di cibo `x[j]` soddisfi il fabbisogno del fattore stesso. Come dire:

```
subject to fabbisogno_calorie:
  sum{j in J} a['Calorie',j] * x[j] >= b['Calorie'];
subject to fabbisogno_proteine:
  sum{j in J} a['Proteine',j] * x[j] >= b['Proteine'];
subject to fabbisogno_calcio:
  sum{j in J} a['Calcio',j] * x[j] >= b['Calcio'];
subject to fabbisogno_ferro:
  sum{j in J} a['Ferro',j] * x[j] >= b['Ferro'];
subject to fabbisogno_vitamine:
  sum{j in J} a['Vitamine',j] * x[j] >= b['Vitamine'];
```

Sono evidenti i vantaggi della forma compatta.

Per comodità e leggibilità entrambi i termini della relazione possono contenere variabili e parametri. Le variabili possono essere ripetute nella stessa relazione o nei suoi due termini. Nei vincoli semplici (cioè senza indici) possono comparire variabili semplici, vettori di parametri, sommatorie su vettori o matrici di variabili purché estese a tutti gli indici. Nei vincoli vettoriali, invece, occorre che l'indice del vincolo sia compatibile con gli indici residui (non utilizzati) nel vincolo stesso.

Imporre una restrizione al valore di una variabile o definire un vincolo sono modi equivalenti di distinguere le soluzioni ammissibili da quelle inammissibili. In generale, il primo tipo di descrizione consente ai risolutori di programmazione lineare di affrontare il problema in modo più efficiente; essa però riduce l'informazione disponibile in uscita, dato che le variabili duali vengono riportate solo per i vincoli espliciti, e non per le restrizioni sulle variabili.

Capitolo 3

Ottenere e visualizzare la soluzione

Una volta dichiarata la struttura del modello nel file **.mod** si deve usare AMPL per ottenere e visualizzare la soluzione di una specifica istanza, legata ai valori nel file **.dat**.

3.1 Esecuzione di AMPL

I comandi per l'esecuzione di un modello tramite AMPL sono contenuti nel file **.run**.

```
model dieta.mod
data dieta.dat
option solver cplex;
solve;
```

Le istruzioni del precedente file prevedono:

1. il caricamento del file del modello, con l'istruzione **model** seguita dal nome del file di modello
2. il caricamento del file dei dati, con l'istruzione **data** seguita dal nome del file dei dati
3. la scelta del risolutore (in questo caso **cplex**) attraverso l'istruzione **option solver** seguita dal nome e dal punto e virgola
4. il lancio del risolutore, con l'istruzione **solve** seguita dal punto e virgola (il software risponde indicando il risolutore impiegato, alcuni dettagli sul procedimento e il valore della soluzione)

Si può costruire un problema caricando diversi file di modello e di dati con le istruzioni **model** e **data**, che vengono aggiunti via via al modello corrente.

Lanciato il risolutore, AMPL determina se i dati rispettano le istruzioni di controllo e segnala eventuali violazioni. Quindi, esegue una fase di *presolve*, nella quale individua le variabili fissate

da condizioni stringenti e rafforza le restrizioni imposte dall'utente combinandole fra loro. Il risultato è in generale un problema ridotto, oppure la dimostrazione che il problema non ha soluzioni ammissibili.

Si riportano di seguito alcuni possibili comandi, che devono precedere la risoluzione del problema

- **option presolve 0;**: per disabilitare la fase di *presolve*
- **option relax integrality 1;**: per rilassare l'integralità di tutte le variabili
- **option display_1col 0;**: per evitare una scrittura per righe dei risultati

3.2 Definizione di un problema

È possibile definire la struttura di un problema selezionando solo alcune tra le entità dichiarate con il comando **problem**, dopo la scelta del risolutore, come nel seguente esempio

```
problem pb1: x,costo,disponibilita;
```

dove il problema **pb1** sarà costituito dalle entità che seguono i **:**.

Per risolvere poi lo specifico problema è sufficiente usare il comando **solve** seguito dal nome.

```
solve pb1;
```

3.3 Visualizzare i risultati

Per accedere ai risultati e a tutti gli elementi del modello, si usa l'istruzione **display**, che dà loro un formato analogo a quello impiegato per definirli.

```
ampl: display J;
set J := Pane Latte Carne Verdure;
```

L'insieme che segue la parola chiave **display** può anche essere costruito con un'espressione di indicizzazione.

```
ampl: display {j in J: c[j] >= 10};
set {j in J: c[j] >= 10} := Pane Carne;
```

Lo stesso avviene per i parametri e per le variabili, i vincoli e la funzione obiettivo. Per visualizzare i dati è sufficiente che essi siano stati caricati, mentre per gli altri termini finché il problema non è risolto viene visualizzato solo il valore iniziale, che non è significativo.

3.4 Modificare un modello

La modifica di un modello permette di sperimentare per verificarne la correttezza o studiarne le proprietà. Si tratta essenzialmente di rilassare vincoli o fissare variabili.

Per rilassare un vincolo, si usa la parola chiave **drop** seguita dal nome del vincolo e dal punto e virgola.

```
drop vincolo;
drop fabbisogno['Calcio'];
```

È possibile rilassare interi vettori di vincoli oppure una selezione ottenuta con espressioni di indicizzazione.

```
drop fabbisogno;
drop fabbisogno{i in I: d[i] >= 2};
```

Per riattivare il vincolo, basta eseguire il comando **restore** con la stessa sintassi.

```
restore vincolo;
restore fabbisogno{i in I: d[i] >= 2};
```

Si può fissare una variabile al suo valore corrente attraverso la parola chiave **fix**, seguita dal nome della variabile oppure da un'espressione di indicizzazione che determina un sottoinsieme di variabili da fissare e dal loro nome.

```
fix x := 10;
fix z{j in J} := q[j];
```

Per rendere di nuovo libera la variabile, basta eseguire il comando **unfix**.

Bibliografia

[1] R. Fourer, D. M. Gay and B. W. Kernighan. *AMPL: A Modeling Language For Mathematical Programming.* Boyd & Fraser Publishing Company, Danvers, Massachussets, (1999)