Linguaggi di modellizzazione

Come visto, il primo passo per risolvere un problema di decisione consiste nel formularne il modello matematico.

Una volta definito il modello matematico lo dobbiamo passare a un risolutore che ne restituisce la soluzione.

Il passaggio dal modello matematico al risolutore non è però immediato. È necessario tradurre il modello nelle strutture dati che devono essere passate come input al programma risolutore.

I linguaggi di modelizzazione come AMPL cercano proprio di facilitare questo compito.

Continua

Invece di passare direttamente dal modello su carta all'input per il risolutore, si scrive un modello in linguaggio AMPL che viene poi passato a un traduttore che si occupa di trasformare il modello scritto in AMPL nell'input per il programma risolutore.

Il vantaggio di questo passaggio supplementare è che AMPL è concepito in modo che il modello scritto in AMPL sia molto simile al modello scritto su carta.

Inoltre, risolutori diversi possono richiedere input in formati molto diversi tra loro. Di tutto questo non si deve preoccupare l'utente AMPL, che utilizza la stessa sintassi indipendentemente dal risolutore che verrà utilizzato. L'utente si limiterà a specificare il risolutore tramite il comando

ampl: option solver *nome_risolutore*;

Per problemi di PL e PLI risolutori molto utilizzati sono *cplex*, *gurobi* e *lpsolve*.

Un esempio: problema dieta

Sia dato un insieme $PROD$ di prodotti ed un insieme $SOST$ di sostanze nutritive. Si sa che in un'unità del prodotto j si trova una quantità $quant\_unit_{ij}$ della sostanza nutritiva i. Inoltre si sa che il costo di un'unità di prodotto j è pari a $cost\_unit_j$. Tenendo conto che una dieta deve contenere una quantità minima $quant\_min_i$ di ciascuna sostanza nutritiva, si vuole determinare una dieta che abbia costo minimo.

Modello matematico

- **Variabili:** a ogni prodotto j si associa una *variabile* x_j indicante la quantità di prodotto da inserire nella dieta.
- **Vincoli:** dobbiamo avere almeno una quantità $quant\_min_i$ di ciascuna sostanza nutritiva, cioè

$$\sum_{j \in PROD} quant\_unit_{ij} * x_j \geq quant\_min_i \quad \forall\, i \in SOST.$$

  Dobbiamo avere una quantità non negativa, cioè

$$x_j \geq 0 \quad \forall\, j \in PROD.$$

- **Obiettivo:** minimizzare il costo complessivo della dieta, cioè

$$\min \sum_{j \in PROD} cost\_unit_j * x_j.$$

Quindi

$$\min \quad \sum_{j \in PROD} cost\_unit_j * x_j$$

$$\sum_{j \in PROD} quant\_unit_{ij} * x_j \geq quant\_min_i \quad \forall\, i \in SOST$$

$$x_j \geq 0 \quad \forall\, j \in PROD$$

Parti fondamentali di AMPL

- Insiemi di indici Nel caso del problema della dieta sono i due insiemi $PROD$ dei prodotti e $SOST$ delle sostanze nutritive.
- Parametri Sono tutte le quantità il cui valore è noto prima di risolvere il problema. Nel problema della dieta sono i costi unitari $cost\_unit_j$ dei prodotti, le quantità minime $quant\_min_i$ delle sostanze nutritive e le quantità $quant\_unit_{ij}$ di sostanza i in un'unità di prodotto j.
- Variabili Sono le quantità i cui valori devono essere stabiliti attraverso la soluzione del problema. Nell'esempio sono le variabili x_j indicanti le quantità di prodotto j (tutte non negative).

- Vincoli Limitano la scelta delle variabili. Nell'esempio abbiamo i vincoli sulle quantità minime di ciascuna sostanza.
- Obiettivo È la quantità il cui valore va massimizzato (o minimizzato) scegliendo opportunamente i valori delle variabili. Nell'esempio si deve minimizzare il costo della dieta.

Osservazione

Queste componenti fondamentali di AMPL coincidono con le componenti che abbiamo riconosciuto in un problema di decisione, con però un maggiore dettaglio: la componente dati è stata scomposta nelle due componenti insiemi di indici e parametri.

Si tratta ora di vedere come queste componenti vengono dichiarate e definite in linguaggio AMPL.

Insiemi

Un insieme T si dichiara semplicemente attraverso la seguente sintassi

set T **;**

Si noti che questa è soltanto la dichiarazione dell'insieme. Da una qualche altra parte, come vedremo, andrà specificato il contenuto dell'insieme, ovvero verrà data la definizione dell'insieme.

Nel nostro esempio avremo le seguenti dichiarazioni

set $PROD$ **;**
set $SOST$ **;**

Parametri

Un parametro a si dichiara nel modo seguente

param a **;**

Qualora sia noto che il parametro è sempre positivo conviene indicarlo esplicitamente nella dichiarazione nel modo seguente

param $a > 0$ **;**

In modo analogo si può specificare che un parametro è non negativo ($>= 0$), negativo (< 0) o non positivo ($<= 0$).

Si può anche specificare che un parametro deve avere un valore intero nel modo seguente

param $a > 0$ **integer ;**

In questo caso il parametro a deve essere un intero positivo.

Continua

È possibile anche dichiarare vettori di parametri con indici in un insieme T attraverso la seguente dichiarazione

param $a\{T\}$ **;**

Nel caso gli indici siano gli interi da 1 a n si può scrivere

param $a\{1..n\}$ **;**

Si possono infine anche dichiarare array bidimensionali con insiemi di indici T_1 e T_2 nel modo seguente

param $a\{T_1, T_2\}$ **;**

Nell'esempio

Nell'esempio si avranno le seguenti dichiarazioni

param $cost\_unit\{PROD\} > 0$ **;**
param $quant\_unit\{SOST,PROD\} >= 0$ **;**
param $quant\_min\{SOST\} > 0$**;**

Si noti che si richiede che i costi unitari e le quantità minime siano strettamente positive, mentre le quantità di sostanza in un'unità di prodotto devono essere non negative.

Variabili

La dichiarazione delle variabili è del tutto analoga a quella dei parametri. La sola differenza è che la parola chiave **param** viene sostituita dalla parola chiave **var**.

Nel nostro esempio abbiamo un vettore di variabili x con indici in $PROD$ che sarà definito in questo modo

var $x\{PROD\} >= 0$ **;**

Si noti che nella dichiarazione delle variabili sono già compresi i vincoli di non negatività delle stesse.

Continua

Nel caso una variabile sia vincolata ad assumere solo valori interi è sufficiente aggiungere la parola chiave **integer** nella sua dichiarazione, esattamente come si è fatto per i parametri. Si noti che questa aggiunta è la sola cosa che distingue un modello di AMPL per la PL da uno per la PLI.

Se sulle variabili si hanno, oltre ai vincoli di non negatività, anche limiti superiori sui valori delle stesse, possiamo indicare tali limiti sulla stessa riga.

Per esempio, se avessimo anche un parametro

param $max\_prod\{PROD\} > 0$ **;**

che definisce dei limiti superiori sui valori assunti dalle variabili, potremmo modificare la dichiarazione delle variabili come segue

var $x\{i$ **in** $PROD\} >= 0$, $<= max\_prod[i]$ **;**

Si noti come in questo caso abbiamo dovuto introdurre l'indice i appartenente (**in**) all'insieme $PROD$ per potersi riferire alle componenti del parametro max_prod (si noti anche l'uso delle parentesi quadre per richiamare tali componenti).

Vincoli

Un singolo vincolo viene dichiarato nel seguente modo

subject to *nome_vincolo* **:** *formula_vincolo* **;**

Una collezione di vincoli indicizzata su un insieme di indici I viene dichiarata in questo modo

subject to *nome_insieme_vincoli* {i **in** I} **:** *formula_vincoli* **;**

Nell'esempio

subject to min_sostanza {i **in** $SOST$} **:** **sum** {j **in** $PROD$}
quant_unit[i,j]*x[j] $>$= quant_min[i] **;**

Notare l'uso di **sum** {j **in** J} per la definizione di una sommatoria con indice J.

Obiettivo

L'obiettivo si dichiara, in generale, nel modo seguente

maximize *nome_obiettivo* **:** *formula_obiettivo* **;**

(nel caso si debba minimizzare si usa la parola chiave **minimize** al posto di **maximize**).

Nel nostro esempio avremo

minimize total_cost **: sum** {j **in** $PROD$} cost_unit[j]*x[j] **;**

DIETA.MOD

Le scritte comprese tra ### sono commenti.

INSIEMI

set $PROD$;

set $SOST$;

PARAMETRI

param $cost\_unit\{PROD\} > 0$;

param $quant\_unit\{SOST,PROD\} >= 0$;

param $quant\_min\{SOST\} > 0$;

VARIABILI

var $x\{PROD\} >= 0$ **;**

VINCOLI

subject to min_sostanza {i **in** $SOST$} **: sum** {j **in** $PROD$} quant_unit[i,j]*x[j] >= quant_min[i] **;**

OBIETTIVO

minimize total_cost **: sum** {j **in** $PROD$} cost_unit[j]*x[j] **;**

Dati (input) del problema

Una volta costruito il modello bisognerà inserire in un altro file i valori di insiemi e parametri, ovvero i dati di input dell'istanza del nostro problema.

Mentre il modello viene inserito in un file con estensione .MOD, i valori vengono inseriti in un file con estensione .DAT.

Il file DIETA.DAT dovrà contenere le definizioni degli insiemi $PROD$ e $SOST$ e i valori assegnati ai diversi parametri.

Istanza per l'esempio

Supponiamo di avere a disposizione i seguenti dati. L'insieme $PROD$ contiene *pasta, verdura, carne*; l'insieme $SOST$ contiene *vitamine, proteine*; un'unità di pasta, verdura e carne costano rispettivamente 3,2 e 5; le quantità minime di vitamine e proteine sono rispettivamente 8 e 6; le quantità di vitamine in un'unità di pasta, verdura o carne sono rispettivamente 0.3, 0.5 e 0.4; le quantità di proteine in un'unità di pasta, verdura o carne sono rispettivamente 0.5, 0.2 e 0.7.

Definizione insiemi

Per la definizione di un insieme T contenente gli oggetti $t_1, t_2, \ldots, t_n$ si usa la seguente sintassi

set T **:=** $t_1\ t_2\ \ldots\ t_n$ **;**

Nel nostro esempio avremo

set $PROD$ **:=** pasta verdura carne **;**
set $SOST$ **:=** vitamine proteine **;**

Definizione parametri

Per quanto riguarda i parametri si usa la seguente sintassi

param a **:=** *valore_parametro* **;**

Per parametri vettore con insieme indice $T = \{t_1, \ldots, t_n\}$ si usa la seguente sintassi

param a **:=**
t_1 *valore*$_1$
$\vdots$
t_n *valore*$_n$ **;**

Per parametri che sono array bidimensionali con primo insieme di indici $T = \{t_1, \ldots, t_n\}$ e secondo insieme di indici $S = \{s_1, \ldots, s_m\}$ si usa la seguente sintassi

param a **:**

$$\begin{array}{cccc} & s_1 & \cdots & s_m \\ t_1 & val(t_1, s_1) & \cdots & val(t_1, s_m) \\ \vdots & \vdots & \vdots & \vdots \\ t_n & val(t_n, s_1) & \cdots & val(t_n, s_m) \end{array} \quad \textbf{:=} \quad ;$$

Nell'esempio

param cost_unit **:=**
pasta 3
verdura 2
carne 5 ;

param quant_min **:=**
vitamine 8
proteine 6 ;

param quant_unit **:**

	pasta	verdura	carne	:=
vitamine	0.3	0.5	0.4	
proteine	0.5	0.2	0.7	;

DIETA.DAT

INSIEMI

set $PROD$:= pasta verdura carne ;
set $SOST$:= vitamine proteine ;

PARAMETRI

param cost_unit :=
pasta 3
verdura 2
carne 5 ;
param quant_min :=
vitamine 8
proteine 6 ;

param quant_unit :

	pasta	verdura	carne	:=
vitamine	0.3	0.5	0.4	
proteine	0.5	0.2	0.7	;

Osservazione

Qui abbiamo inserito certi dati ma può capitare che lo stesso tipo di problema debba essere risolto con altri dati (ad esempio il costo di certi prodotti può cambiare o tra le sostanze se ne possono aggiungere altre come i carboidrati).

AMPL è concepito in modo tale che queste modifiche possano essere fatte andando a modificare il solo file .DAT mentre nessuna modifica deve essere fatta nel file .MOD.

Risoluzione del problema

È sufficiente inserire i seguenti comandi in corrispondenza del prompt ampl:

```
ampl: reset;
ampl: option solver gurobi;
ampl: model DIETA.MOD;
ampl: data DIETA.DAT;
ampl: solve;
```

(Si notino i ";" al termine di ogni comando).

Nota bene-I

Il primo comando di reset non è sempre necessario se non si sono verificati cambiamenti (o, nel caso questi riguardino solo i dati, ci si può limitare a un comando "reset data;") ma, dal momento che vengono sempre tenuti in memoria l'ultimo modello e gli ultimi dati caricati, conviene usare il reset per evitare che modello e dati attuali siano "sporcati" da informazioni precedenti.

Nota bene-II

La seconda riga specifica che si richiede gurobi (risolutore di problemi di PL e PLI) come risolutore. L'esecuzione di gurobi resta invisibile all'utente, ma vale la pena citare il fatto che l'utente ha la possibilità di scegliere tra alcune opzioni corrispondenti a diversi modi di funzionamento per il risolutore. Se nulla viene specificato, il risolutore funziona nella modalità di default.

Nota bene - III

La terza riga specifica che il modello deve essere letto dal file DIETA.MOD.

La quarta riga specifica che i dati devono essere letti dal file DIETA.DAT.

Infine, la quinta riga comunica ad AMPL di prendere modello e dati caricati, tradurli nell'input del risolutore e quindi risolvere il problema.

A questo punto apparirà automaticamente il valore ottimo del problema (se, come in questo caso, esiste).

optimal solution; objective 45.263.....

Soluzione primale

Per visualizzare la soluzione primale si deve dare il comando

ampl: display x;

Apparirà la seguente risposta

x[*] :=
carne 0
pasta 7.36842
verdura 11.5789
;

Soluzione duale

Per visualizzare la soluzione duale è sufficiente mandare il comando "display" con il nome dei vincoli primali corrispondenti.

```
ampl: display min_sostanza;
```

Apparirà la seguente risposta

```
min_sostanza [*] :=
proteine 4.73684
vitamine 2.10526
;
```

Coefficienti di costo ridotto

Per visualizzarli basta usare il comando

ampl: display x.rc;

(l'estensione rc sta per *r*educed *c*ost).

Range ottimalità termini noti

Per i termini noti dei vincoli, il comando

ampl: display min_sostanza.down, min_sostanza.current,
min_sostanza.up;

restituisce per ciascun termine noto il limite inferiore del range in cui non cambia la base ottima attuale (down), il valore attuale del termine noto (current) e il limite superiore del range (up).

NB È necessario attivare una option di gurobi:

ampl: option gurobi_options 'sensitivity';

Range ottimalità coefficienti obiettivo

Il comando

```
ampl: display x.down, x.current, x.up;
```

restituisce per ciascun coefficiente nell'obiettivo il limite inferiore del range in cui non cambia la base ottima attuale (down), il valore attuale del coefficiente (current) e il limite superiore del range (up).

Il file .RUN

Invece di digitare uno alla volta i comandi visti, possiamo anche scrivere un file DIETA.RUN in cui specifichiamo tutti questi comandi più eventualmente molti altri con costrutti tipici dei linguaggi di programmazione (statement IF, cicli, eccetera), sui quali non ci soffermeremo qui.

DIETA.RUN

```
reset;
option solver gurobi;
model DIETA.MOD;
data DIETA.DAT;
solve;
display x;
display min_sostanza;
display x.rc;
display min_sostanza.down, min_sostanza.current, min_sostanza.up;
display x.down, x.current, x.up;
```

Una volta scritto il file DIETA.RUN possiamo eseguire tutti i comandi in esso contenuti semplicemente con il comando

ampl: include DIETA.RUN;

Altro esempio

È dato un insieme $OGGETTI$ di oggetti a ciascuno dei quali è associato un *peso* e un *valore*. È inoltre dato uno zaino a cui è associata una *capacità*, ovvero un peso massimo che può essere trasportato all'interno dello zaino. Si vogliono inserire degli oggetti nello zaino in modo tale da massimizare il valore complessivo trasportato in esso, tenendo conto che il peso totale degli oggetti inseriti non può superare la capacità dello zaino.

Modello matematico

- **Varibabili** All'oggetto i associamo una variabile binaria x_i, ovvero una variabile che può assumere solamente i valori 0 e 1. Se $x_i = 0$ l'oggetto i non viene inserito nello zaino, se $x_i = 1$ l'oggetto viene inserito nello zaino.
- **Vincoli** L'unico è quello che gli oggetti inseriti nello zaino abbiano un peso complessivo che non superi la capacità dello zaino, ovvero

$$\sum_{i \in OGGETTI} peso_i * x_i \leq \textit{capacità}.$$

- **Obiettivo** È quello di massimizzare il valore degli oggetti inseriti nello zaino, quindi

$$\max \sum_{i \in OGGETTI} valore_i * x_i.$$

Quindi ...

$$\max \quad \sum_{i \in OGGETTI} valore_i * x_i$$

$$\sum_{i \in OGGETTI} peso_i * x_i \leq \textit{capacità}$$

$$x_i \in \{0, 1\} \qquad i \in OGGETTI$$

In AMPL

Abbiamo un solo insieme, l'insieme $OGGETTI$, che verrà dichiarato nel modo seguente

set $OGGETTI$ **;**

Come parametri abbiamo due vettori di parametri, *peso* e *valore*, con insieme indice $OGGETTI$ e il singolo parametro cap (*capacità* dello zaino). Tutti questi parametri sono positivi e verranno dichiarati nel modo seguente

param $peso\{OGGETTI\} > 0$ **;**
param $valore\{OGGETTI\} > 0$ **;**
param cap > 0 **;**

Continua

Vettore di variabili con insieme indice $OGGETTI$. Le variabili sono vincolate ad assumere i soli valori 0 e 1. Si può esprimere questo con la seguente dichiarazione delle variabili

var $x\{OGGETTI\}$ >= 0, <= 1, **integer ;**

In questo modo le variabili sono vincolate ad essere interi compresi tra 0 e 1 e quindi possono assumere i soli valori 0 e 1.

Tuttavia, visto il largo uso che si fa nella PLI di variabili binarie, AMPL prevede una dichiarazione speciale per esse:

var $x\{OGGETTI\}$ **binary ;**

Continua

Vincolo sulla capacità dello zaino:

subject to max_capac **:** sum{i **in** $OGGETTI$} $peso[i] * x[i]$ <= cap **;**

Obiettivo:

maximize tot_valore **:** sum{i **in** $OGGETTI$} $valore[i] * x[i]$ **;**

ZAINO.MOD

INSIEMI

set $OGGETTI$ **;**

PARAMETRI

param $peso\{OGGETTI\} > 0$ **;**

param $valore\{OGGETTI\} > 0$ **;**

param cap > 0 **;**

VARIABILI

var $x\{OGGETTI\}$ **binary ;**

VINCOLI

subject to max_capac **:** sum{i **in** $OGGETTI$} $peso[i] * x[i]$ <= cap **;**

OBIETTIVO

maximize tot_valore **:** sum{i **in** $OGGETTI$} $valore[i] * x[i]$ **;**

Un'istanza

Consideriamo un caso in cui gli oggetti sono un *libro*, una *radio*, un *telefono* e una *macchina fotografica*, con peso rispettivamente pari a 3, 7, 2 e 4 e valore 5, 8, 4 e 7. La capacità dello zaino è 11.

ZAINO.DAT

INSIEMI

set $OGGETTI$ **:=** libro radio telefono macchina_fotografica **;**

PARAMETRI

param peso **:=**

libro 3

radio 7

telefono 2

macchina_fotografica 4 **;**

param valore **:=**

libro 5

radio 8

telefono 4

macchina_fotografica 7 **;**

param cap := 11 **;**

Risoluzione

```
ampl: reset;
ampl: option solver gurobi;
ampl: model ZAINO.MOD;
ampl: data ZAINO.DAT;
ampl: solve;
```

Visualizzazione soluzione

Attraverso il comando

```
ampl: display x;
```

è possibile visualizzare la soluzione ottima.

Nell'esempio specifico essa è

```
x[*] :=
libro 1
macchina_fotografica 1
radio 0
telefono 1 ;
```

con valore ottimo tot_valore pari a 16.

Nota bene

Anche in questo caso potremmo raccogliere tutti i comandi in un apposito file ZAINO.RUN eseguito tramite

include ZAINO.RUN **;**

Costrutti sintattici

AMPL consente di scrivere programmi in modo del tutto analogo ai classici linguaggi di programmazione.

Costrutti sintattici tipici sono:

- assegnamenti di valori
- If
- cicli For
- cicli While

Assegnamenti di valori

Se a è un parametro o una variabile, possiamo assegnare un valore particolare ad a con la seguente sintassi:

let a **:=** *valore* **;**

Se a è un vettore di parametri o di variabili con insieme di indici I, possiamo assegnare a tutti un unico valore con la seguente sintassi

let $\{i$ **in** $I\}$ $a[i]$ **:=** *valore* **;**

In alternativa, se b è un altro vettore di parametri o di variabili con lo stesso insieme di indici I, possiamo ricopiare b in a con la seguente sintassi

let $\{i$ **in** $I\}$ $a[i]$ **:=** $b[i]$ **;**

Se vogliamo assegnare alle componenti di a dei valori casuali generati in modo uniforme nell'intervallo $[u_0, u_1]$, si userà la seguente sintassi

let $\{i$ **in** $I\}$ $a[i]$ **:=** Uniform(u_0, u_1) **;**

If

La sintassi è la seguente

if (*condizione* **)** **then** $\{\ldots\}$ **else** $\{\ldots\}$

La parte **else** è opzionale.

Cicli For

Dato un insieme di indici I, la sintassi di un ciclo for è la seguente

for $\{i$ **in** $I\}$ $\{\ldots\}$

In alternativa, se n è un parametro intero, si può scrivere

for $\{i$ **in** $1..n\}$ $\{\ldots\}$

Dato un parametro o una variabile a con insieme di indici I, si può usare la seguente sintassi

for $\{i$ **in** $I \ : \ a[i] \leq valore\}$ $\{\ldots\}$

per restringere il **for** ai soli indici che soddisfano la condizione specificata dopo **:** (in questo caso $a[i] \leq valore$).

Cicli While

La sintassi è la seguente

repeat $\{\ldots\}$ **while** (*condizione*) **;**

Scrittura su file

Se si vuole scrivere il valore di un parametro (o una variabile) a su un file, si usa sempre il **display**, ridirigendo l'output sul file

display $a >>$ *nomefile* **;**

Una volta terminata l'esecuzione del programma si deve provvedere a chiudere il file

close *nomefile* **;**

Programmi in AMPL

In questo modo è possibile scrivere in AMPL dei programmi nello stile dei linguaggi di programmazione.

Tali programmi possono essere scritti in un file .run ed eseguiti con la sintassi già vista

include nomefile.run **;**

Esempio: ZAINO.RUN

```
reset;
option solver gurobi;
model zaino.mod;
data zaino.dat;
solve;
display x >> zaino.out;
for {i in OGGETTI} {
if (peso[i]<=10) then { let peso[i]:=peso[i]-1; }
else { let peso[i]:=peso[i]+1; }
}
solve;
display x>>zaino.out;
close zaino.out;
```