Padova, Ottobre 2000

Appunti sul linguaggio MPL

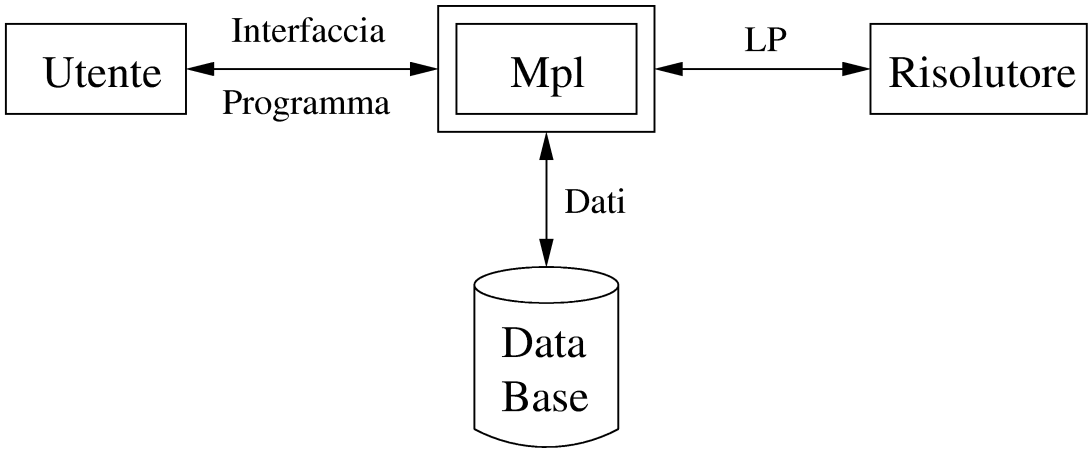
Ivan Luzzi

Dipartimento di Matematica Pura e Applicata,
Università di Padova,
Via Belzoni 7, 35131 Padova – ITALY,
e-mail: iluzzi@libero.it

Generatori Algebrici di Modelli

I **generatori algebrici di modelli** perseguono i seguenti obiettivi:

- fornire un linguaggio di programmazione ad alto livello che permetta di descrivere in modo semplice modelli reali anche molto complessi;
- creare modelli indipendenti dal risolutore utilizzato, in modo da poter sfruttare sempre i risolutori più potenti presenti sul mercato;
- descrivere modelli nei quali la struttura logica del problema e i dati utilizzati possano essere considerati come entità diverse.



Caratteristiche fondamentali

Le principali caratteristiche di questi generatori algebrici di modelli sono le seguenti:

- il linguaggio di programmazione utilizzato è semplice da imparare (è composto da poche parole chiave);
- i codici che si realizzano sono essi stessi una rappresentazione concisa ed elegante del modello;
- è possibile inserire commenti o "definizioni linguistiche" che rendono sia l'input che l'output comprensibili anche da "non-programmatori";
- la maggior parte dei programmi creati sta in un unico documento il che facilita il controllo e la creazione dei modelli;
- nell'output vengono rappresentate, in modo comprensibile, tutte le operazioni svolte dal solutore;
- i programmi vengono scritti con un semplice editor di testo e sono indipendenti dalla piattaforma sulla quale vengono eseguiti sia il programma, sia il solutore (anche se Windows è la piattaforma su cui è più usato);
- i modelli sviluppati sono indipendenti dal risolutore utilizzato, e non hanno bisogno di alcun aggiornamento alle nuove versioni. Eventuali nuovi *feature* introdotte nel risolutore saranno integrate e sfruttate direttamente dal linguaggio senza bisogno di eseguire alcun *tuning*.

Model Development Environment

MPL è l'acronimo di *Model Development Environment*, cioè Ambiente di Sviluppo di Modelli; si tratta di un linguaggio di programmazione ad alto livello che permette di descrivere in modo semplice e intuitivo dei modelli di Programmazione Lineare.

MPL ha circa 5 anni possiede una piccola libreria di esempi di modelli che comprende sia problemi lineari, che problemi non lineari, e un tutorial.

Caratterisctiche generali

Lunghezza delle righe

Ogni riga non può superare i 255 caratteri. I rimanenti verranno ignorati da **MPL** .

Commenti

le parentesi grafe { ...} racchiudono commenti in blocco, cioè che possono estendersi anche su più righe

```
{
  Commento...
}
```

il punto esclamativo "!" delimita invece l'inizio di un commento che termina a fine riga

```
x[foods]   !  dollars of food to be purchased daily
```

Separatori

I diversi statement all'interno di una sezione (indici, vincoli, etc...) devono essere separati dal carattere ";"

```
i = 1..5 ;
j = 1..12 ;
```

Gli elementi di una lista di valori devono essere separati dalla virgola "," o da un "a capo" (questo permette di introdurre dei commenti esplicativi accanto ad ogni valore).

```
nutrients  =  (Calorie,Protein,Calcium,Iron,Vitamin);

Required[nutrients]  =  (  3       ! Calories
                          70       ! Protein
                           0.8     ! Calcium
                          12       ! Iron
                          75 ) ;   ! Vitamin
```

Range di un insieme

Si può esprimere un insieme di valori con l'uso di caratteri speciali:

```
[indice = min .. max]
[indice > val]
```

oppure selezionare un singolo indice di valore specificato

```
["val"]
```

Formule

Si possono scrivere espressioni che comprendono le normali operazioni aritmetiche +, -, *, / nonché diverse altre funzioni matematiche: *sqr, sqrt, sin, cos, log, exp, power ....*

Esiste inoltre la funzione sommatoria che richiede come primo parametro il nome dell'indice su cui iterare e come secondo l'espressione da valutare con l'indice specifico:

```
Cost = SUM(foods: x);
```

Struttura di un programma MPL

Definizione

TITLE	- Il nome del modello
INDEX	- Gli indici del problema
DATA	- I dati (scalari, vettori e matrici multidimensionali)
DECISION	- Le variabili del problema
MACRO	- Macro riutilizzabili nelle varie espressioni

Modello

MODEL	- Descrizione del problema
MAX o MIN	- Funzione obiettivo
SUBJECT TO	- Vincoli
BOUNDS	- Estremi superiori ed inferiori delle variabili
FREE	- Variabili libere
INTEGER	- Variabili intere
BINARY	- Variabili binarie (0/1)
END	- Fine del modello

TITLE

Non è obbligatiorio, ma è buona norma usarlo. Terminare la riga con il carattere ";"

INDEX

Gli indici definiscono il dominio del problema e la sua dimensione.

indici numerici

specificare i due valori estremi separati dal carattere ".."

```
month := 1..12;
```

indici alfabetici

lista di nomi separati dalla virgola e racchiusi tra parentesi tonde. (preferibili perché semplificano la lettura di vincoli ed altre espressioni)

```
nutrients := (Calorie,Protein,Calcium,Iron,Vitamin);
```

Indici alias

In **MPL** è possibile assegnare un nuovo nome ad uno stesso indice senza dover ripeterne la definizione:

```
i := 1..4;
j := i;
```

Indici circolari

Per indici temporali si usa spesso un offset sull'indice: *Stock[month-1]*. Se l'indice esce dal range assegnato **MPL** ignora quel particolare valore. Si può invece forzarlo a ripartire dalla parte opposta del range rendendo la lista circolare. Basta aggiungere alla dichiarazione dell'indice la parola riservata CIRCULAR.

```
day    := (mon, tue, wed, thu, fri, sat, sun) CIRCULAR;
month  := 1..12 CIRCULAR;
```

Sottoinsiemi di indici

Se si vuole dichiarare un nuovo indice composto da un sottoinsieme di valori di un altro basta usare la seguente sintassi:

```
holiday[day] := (sat, sun);
summer[month] := (7..9);
```

dove tra parentesi quadre viene specificato l'insieme di origine.

Si possono anche eseguire le normali operazioni sugli insiemi:

- Differenza con il simbolo: -
- Negazione con il simbolo: NOT
- Unione con uno dei simboli: +, OR, UNION
- Intersezione con uno dei simboli: AND, INTERSECTION

```
INDEX
   plants := (NewYork, Chicago, London, Paris);
   OpenPlants[plants]    := (NewYork, London);
   EuropePlants[plants]  := (London, Paris);

! Differenza
   ClosedPlants[plants]  := plants - OpenPlants;
                         ! := (Chicago, Paris)
! Negazione
   USPlants[plants]      := NOT EuropePlants;
                         ! := (NewYork, Chicago)
! Unione
   OpenOrEurope[plants]  := OpenPlants OR EuropePlants;
                         ! := (NewYork, London, Paris)
! Intersezione
   OpenAndEurope[plants] := OpenPlants AND EuropePlants;
                         ! := (London)
```

Indici da file esterno

La lista di valori di un indice può anche essere letta da un file esterno in cui i dati sono separati dalla virgola o da uno spazio. Se poi oltre all'indice il file contiene anche altri dati, si può specificare quale colonna contiene i valori dell'indice.

```
product := INDEXFILE("product.dat", 1);
```

NB: Se i dati sono su un foglio elettronico o un database esterno si può accedere anche direttamente ai dati con opportuni comandi.

DATA

Lettura dei valori dei dati da usare nel modello $\Longrightarrow$ assegnazione di una particolare istanza al problema.

I dati possono essere dichiarati direttamente o essere caricati da un file esterno, da un foglio elettronico o da un database, con il comando DATAFILE.

I dati possono essere scalari, vettori o matrici multidimensionali.

Nel caso scalare il valore assegnato risulta una costante per tutta la risoluzione del modello.

Con il carattere speciale "?" in run-time si chiede conferma all'utente del valore predefinito

```
NumberOfMonths  =  12;
NumberOfYears  =  4?;
```

Per vettori e matrici:

- indici racchiusi tra parentesi quadre e separati dalla virgola
- valori racchiusi tra parentesi tonde e separati dalla virgola o a capo.

```
Required[nutrients]  =  ( 3, 70, 0.8, 12, 5, 2.7,  18, 75 );
A[foods,nutrients] = DATAFILE(nutri.dat)
       !  Nutritive values of foods per dollar expenditure.
```

Nel caso di una matrice, la lista corrisponde alla scansione per riga della matrice stessa. Lo stesso concetto si estende al caso di matrici multidimensionali.

Dati semplici o il risultato di espressioni matematiche.

```
A[i] := (2, -4+3, 2*SQR(3)+2, 1/(2+1), last(i))
```

Per vettori e matrici sparse, si possono inserire solo i valori non nulli; basta sostituire alle normali parentesi tonde quelle quadrate e far precedere ai singoli valori i relativi indici.

```
A[i] := [2: 4.0, 5: 3.0, 6: -4.0] ;

ProdCost[plant, machine, product] := [
          p1,  m11,  A1,  73.30,
          p1,  m11,  A2,  52.90,
          p1,  m12,  A3,  65.40,
          p1,  m13,  A3,  47.60,

          p2,  m21,  A1,  79.00,
          p2,  m21,  A3,  66.80,
          p2,  m22,  A2,  52.00,

          p3,  m31,  A1,  75.80,
          p3,  m31,  A3,  50.90,
          p3,  m32,  A1,  79.90,
          p3,  m32,  A2,  52.10,

          p4,  m41,  A1,  82.70,
          p4,  m41,  A2,  63.30,
          p4,  m41,  A3,  53.80];
```

DECISION

Si usa indifferentemente:
DECISION VARIABLES
DECISION
VARIABLES

Dichiarazione delle variabili:

- scalari
- vettori
- matrici

Nel caso di vettori o matrici, i relativi indici vanno racchiusi tra parentesi quadre e separati dalla virgola.

```
Production[product,month]
```

Condizioni sulle variabili

Per limitare le variabili al verificarsi di determinate condizioni basta aggiungere la clausola WHERE seguita dalla condizione

```
Production[product,month] WHERE (Demand[product,month] > 0);
```

MACROS

Definizione di espressioni a cui assegnare un nome specifico da usare poi nella definizione della funzione obiettivo e dei vincoli (semplificano il modello rendendolo piú leggibile).

nomemacro := *espressione*

Non si possono dichiarare macro indicizzate, ma si possono usare vettori e formule nel calcolo della sua espressione.

```
TotalRevenue  := SUM(product,month: price * Sales) ;
```

MODEL

Inizio della descrizione del modello vero e proprio. La funzione obiettivo richiede:

- il senso di ottimizzazione: MAXIMIZE o MINIMIZE (MAX o MIN)
- il nome (facoltativo ma consigliato)
- il segno di uguaglianza "="
- l'espressione da valutare

Se non viene indicato alcun nome **MPL** assegna per definizione il nome "z" alla funzione obietttivo.

```
MAX  3x1 + 5x2 ;

MIN  Cost = SUM(products,months : InventoryCost) ;
```

SUBJECT TO

Descrizione dei vincoli del problema.
È buona abitudine assegnare un nome ad ogni vincolo, ed anteporlo alla sua definizione. Il nome non può contenere spazi vuoti ed altri caratteri riservati e deve essere seguito dal simbolo ":". (nome automatico per il vincolo i-esimo: "c*i*").

Un'equazione è composta da due membri separati da uno dei seguenti segni di comparazione:

minore o uguale	$<$	$<=$
uguale	$=$	
maggiore o uguale	$>$	$>=$

Sia le variabili che le costanti possono essere scritte in entrambi i lati della relazione. Le variabili possono anche essere ripetute nella stessa formula o nei due lati dello stesso vincolo.

Ogni vincolo può estendersi anche su più righe e deve essere terminato con il simbolo ";" per separarlo dal successivo.

Vincoli semplici

Vincoli senza indici: formati dalla combinazione di variabili semplici, vettori costanti ed eventualmente sommatorie su vettori di variabili (purchè tutte le variabili del vettore siano utilizzate).

```
3 (Sb1 + Co1 + So1)  =  2 (Sb2 + Co2 + So2) ;
Overtime   :  Over  <  50\% * 170 Workforce ;
Production :  SUM(shifts: Prod[shifts])  <  3750 ;
```

Vincoli vettoriali

Vincoli indicizzati: **MPL** li analizza e li riscrive cambiando il valore dell'indice ogni volta.

Si dichiarano aggiungendo il nome degli indici racchiusi tra parentesi quadre subito dopo il nome del vincolo e prima del simbolo ":".

Non occorre dichiarare l'indice dopo ogni variabile, purchè questo sia compatibile con l'indice principale del vincolo.

```
InventoryBalance[product,period]:
        Inventory  =  Inventory[period-1] + Production - Sales ;

ProdCap[machine,time] :
        SUM(tire: ProdRates * Prod)  <  ProdHours ;
```

Limitare i vincoli vettoriali

Direttamente nella definizione stessa del vicnolo

```
InventoryBalance[month=Jan..Nov] :  ...
InventoryBalance[month<=Jun] :  ...
InventoryBalance[month=Dec] :  ...
```

oppure al verificarsi di determinate condizioni

```
InventoryBalance[product,month]
        WHERE (month > Jan)
```

BOUNDS

Definizione dei limiti inferiori e superiori delle variabili.

Aumentano l'efficienza dei risolutori rispetto ai vincoli espliciti

Il limite può andare sul lato sinistro o destro della relazione e si possono anche definire vincoli composti del tipo:

$$limiteinferiore < variabile < limitesuperiore$$

Il limite inferiore è per definizione 0.

```
x  <  4*12 ;
z_bounds :  2  <  z  <  8 ;
CloseInv :  Inventory[December]  =  20000 ;

MaxInv[month<=Nov] :  Inventory  <=  90000 ;
Inventory[month=Dec]  =  90000 ;
Sales  <=  Demand ;
```

FREE

Definizione del tipo di variabili. Possono essere poste in qualsiasi ordine purchè dopo la sezione dei vincoli (SUBJECT TO).

MPL come la maggior parte dei risolutori LP, assume che le variabili siano positive o nulle. Per permettere ad una variabile semplice o ad un vettore di variabili di assumere anche valori negativi bisogna dichiararli come liberi.

```
FREE
    Temperature ;
    Inventory[month] ;
```

INTEGER e BINARY

Similmente si possono forzare delle variabili ad assumere solamente valori interi:

```
INTEGER
    Production[month,product] ;
```

oppure solamente valori binari (0 o 1):

```
BINARY
    ShopOpen ;
```

N.B. Il risolutore deve supportare la progrmmazione lineare intera (Mixed Integer Programming).

Problema di assegnamento

Un'azienda deve eseguire N lavori diversi avendo a disposizione N persone. Ognuna impiega un determinato tempo a svolgere un certo incarico secondo la tabella riportata sotto. Sapendo che la ditta paga i suoi dipendenti ad un determinato prezzo orario, come devono essere assegnati gli incarichi alle diverse persone per minimizzare il costo complessivo?

Lavoro	Bianchi	Verdi	Rossi	Ferrari
lavoro A	12	15	9	5
lavoro B	13	16	11	6
lavoro C	6	8	5	3
lavoro D	5	7	4	3

Modello matematico

Poichè il costo è direttamente proporzionale al tempo impiegato, è sufficiente minimizzare il tempo totale.

Variabili binarie x_{ij}:

$$x_{ij} = \begin{cases} 1 & \text{se il lavoro i-esimo viene assegnato alla j-esima persona} \\ 0 & \text{altrimenti} \end{cases}$$

Funzione obiettivo:

$$\min \sum_{i,j} t_{ij}\, x_{ij}$$

Vincoli:

- ogni persona può svolgere esattamente un lavoro:

$$\sum_{i} x_{ij} \;=\; 1 \qquad\qquad j = 1\,..\,N$$

- ogni lavoro può essere assegnato solamente ad una persona:

$$\sum_j x_{ij} \;=\; 1 \qquad\qquad i = 1\,..\,N$$

Programma MPL

```
TITLE
    Problema_di_Assegnamento;

INDEX
    persone = (Bianchi, Verdi, Rossi, Ferrari);
    incarichi = (lavoroA, lavoroB, lavoroC, lavoroD);

DATA
    ! tempo impiegato dalla persona j a svolgere il lavoro i
    tempi[persone, incarichi] = ( 12, 15,  9, 5,
                                  13, 16, 11, 6,
                                   6,  8,  5, 3,
                                   5,  7,  4, 3 );

DECISION
    x[incarichi, persone]   ! persona che svolge l'incarico

MODEL
    MIN  tempoTotale = Sum(persone, incarichi: tempi * x);

SUBJECT TO
    vincolo_incarichi[persone] :  Sum(incarichi: x) = 1;
    vincolo_persone[incarichi] :  Sum(persone: x) = 1;

BINARY
    x[persone, incarichi];
```

```
END
```

Soluzione

Se si risolve questo modello si avra il seguente assegnamento ottimo:

lavoro A ==> Rossi
lavoro B ==> Ferrari
lavoro C ==> Bianchi
lavoro D ==> Verdi

di tempo totale minimo = 28.

Problema di trasporto

Una ditta di trasporto deve trasferire container vuoti dai propri magazzini ai principali porti nazionali. Le disponibilità di container vuoti ai magazzini e le richieste ai porti sono le seguenti:

Magazzini	Disponitilitá	Porti	Richiesta
Verona	10	Genova	20
Perugia	12	Venezia	15
Roma	20	Ancona	25
Pescara	24	Napoli	33
Taranto	18	Bari	21
Lamezia	40		

I costi di trasporto sono proporzionali al numero di container ed ai chilometri percorsi dai camion, secondo la seguente tabella:

	Genova	Venezia	Ancona	Napoli	Bari
Verona	290	115	355	715	810
Perugia	380	340	165	380	610
Roma	505	530	285	220	450
Pescara	655	450	155	240	315
Taranto	1010	840	550	305	95
Lamezia	1072	1097	747	372	333

Si vuole determinare la politica di trasporto di costo complessivo minimo.

Modello matematico

Variabili intere:

$$x_{ij} = \text{ numero di container trasferiti dal magazzino} i \text{ al porto } j$$

Funzione obiettivo:

$$\min \sum_{i,j} distanze_{ij} * x_{ij} * costo\_unitario$$

Vincoli:

- ogni magazzino di partenza può spedire tanti container quant'è la sua disponibilità:

$$\sum_{j} x_{ij} \ \leq \ disp_i \qquad\qquad i = 1..6$$

- ogni porto di distinazione deve soddisfare la propria richiesta:

$$\sum_{i} x_{ij} \ \geq \ rich_j \qquad\qquad j = 1..5$$

Programma MPL

```
TITLE
    Problema_di_Trasporto;

INDEX
    magazzini = ( Verona, Perugia, Roma, Pescara, Taranto, Lamezia );
    porti = ( Genova, Venezia, Ancona, Napoli, Bari );

DATA
    disponibilita[magazzini] = ( 10, 12, 20, 24, 18, 40 );
    richiesta[porti] = ( 20, 15, 25, 33, 21 );

    distanze[magazzini, porti] = DATAFILE ("Distanze.dat");
```

```
    costo_unitario = 3000;          ! costo per chilometro

DECISION
    ! numero di containter spediti dal magazzino i al porto j
    x[magazzini, porti]

MODEL
    ! funzione obiettivo
    MIN  costoTotale = Sum(magazzini, porti: distanze*x*costo_unitario)

SUBJECT TO
    vincolo_disponibilita[magazzini] :  Sum(porti: x) <= disponibilita;
    vincolo_richiesta[porti] :  Sum(magazzini: x) >= richiesta;

INTEGER
    x[magazzini, porti];
END
```

Soluzione

Risolvendo questo modello con **MPL** si avrà la seguente configurazione di trasporto ottimo:

	Genova	Venezia	Ancona	Napoli	Bari
Verona	-	10	-	-	-
Perugia	7	5	-	-	-
Roma	13	-	1	6	-
Pescara	-	-	24	-	-
Taranto	-	-	-	-	18
Lamezia	-	-	-	27	3

di costo minimo = £ 90.459.000.

Problema della dieta

Una mensa deve pianificare gli acquisti di alimenti per la sua attività. Nella formulazione della dieta deve obbedire a requisiti nutrizionali minimi, nonché vincolare le porzioni massime di ogni elemento entro certi limiti.

Alimento	Costo unitario	Quantità massima
Pane	2	4
Latte	3	8
Uova	4	3
Carne	19	2
Dolce	20	2

Requisiti nutrizionali minimi

Nutrimento	requisito
Calorie	200 cal
Proteine	50g
Calcio	700 mg

Conoscendo i costi unitari dei vari alimenti, trovare la dieta ottima che minimizzi il costo complessivo rispettando i vincoli imposti.

Modello matematico

Variabili:

$$x_i = \text{ quantità di alimento j da inserire nella dieta}$$

Funzione obiettivo:

$$\min \sum_i costo_i * x_i$$

Vincoli:

- apporto nutrizionale:

$$\sum_{i \in alimenti} apporto_{ij} * x_i \geq richieste_j \qquad \forall j \in nutrimenti$$

- limite massimo di ogni alimento:

$$x_i \leq maxPorz_i \qquad \forall i \in alimenti$$

N.B. Questo limite può essere imposto introducendo dei nuovi vincoli oppure, in modo più efficiente, utilizzando degli upper bound.

Programma MPL

```
TITLE
    Dieta_ottima

INDEX
    nutrimenti  =  (Calorie,Proteine,Calcio) : 10
    alimenti    =  (Pane, Latte, Uova, Carne, Dolce) : 10

DATA
    richieste[nutrimenti]  = ( 2000    ! Calorie          []
                               50      ! Proteine         [grammi]
                               700) ;  ! Calcio           [milligrammi]
```

```
    costo[alimenti] = ( 2      ! Pane
                        3      ! Latte
                        4      ! Uova
                       19      ! Carne
                       20 ) ;  ! Dolce

    maxPorz[alimenti] = ( 4      ! Pane Massimo numero di porzioni
                          8      ! Latte tollerato giornalmente
                          3      ! Uova
                          2      ! Carne
                          2 ) ;  ! Dolce

    apporto[alimenti,nutrimenti] = DATAFILE(nutrimen.dat)
                         !  Contenuti di nutrimenti nei diversi aliment

DECISION
    x[alimenti]   ->  ""    !  porzioni di ogni alimento

MODEL
    MIN  Cost = SUM(alimenti: costo * x) ;

SUBJECT TO
    NutrBal[nutrimenti] : SUM(alimenti: apporto * x) > richieste;

BOUNDS
    MaxPorzioni[alimenti]  :  x < maxPorz ;

END
```

Soluzione

Risolvendo questo modello con **MPL** si avrà la seguente dieta ottima:

Alimento	quantitá
Pane	4
Latte	8
Uova	1.5556
Carne	0
Dolce	0

di costo minimo = 38.222.
Come si può notare la soluzione ottima prevede un valore non intero per l'alimento Uova.
Se vogliamo forzare la soluzione ad assumere solo valori interi basterà aggiungere il seguente vincolo di interezza al modello subito dopo gli altri vincoli:

```
INTEGER
    x;        ! variabili intere
```

La nuova soluzione intera risulta essere ora:

Alimento	quantitá
Pane	4
Latte	8
Uova	2
Carne	0
Dolce	0

di costo minimo = 40.
Il fatto che la soluzione intera sia semplicemente quella lineare con l'unica variabili non intera arrotondata per eccesso è puramente casuale. Spesso la soluzione intera e la sua versione rilassata sono assai diverse.