Corso di Laurea Magistrale in
Scienze Riabilitative delle Professioni Sanitarie

Modulo: Ricerca Operativa
Corso: Management Sanitario

Prof. Renato Bruni

bruni@dis.uniroma1.it

Dipartimento di Ingegneria Informatica, Automatica e Gestionale (DIAG)
Università di Roma "Sapienza"

Modelli e Algoritmi di soluzione

- Dopo aver scritto il modello di ottimizzazione di un problema reale, bisogna ovviamente **risolvere** il modello, cioè trovare una **soluzione ottima**
- Per risolvere, come detto, anziché inventare un algoritmo di soluzione sul momento, utilizziamo **algoritmi di soluzione** già noti e quindi già **perfezionati** ed **efficienti**
- In particolare, per la **Programmazione Lineare** si usa spesso l'algoritmo detto **Simplesso**
- Per la **Programmazione Lineare Intera** si usa spesso l'algoritmo detto **Branch & Bound**
- Esistono molti altri algoritmi, alcuni generici altri specifici per particolari tipi di problemi, però in questo corso non possiamo scendere nei dettagli per motivi di tempo

Implementazione dell'Algoritmo

- L'algoritmo di soluzione deve ovviamente girare su un computer, e quindi deve essere **implementato**, cioè scritto in un linguaggio di programmazione (esempio c++, Java, Python, etc.)
- Anziché dover scrivere la nostra implementazione, è molto più **conveniente** utilizzarne una **già fatta**, se disponibile
- Esistono infatti dei software detti **solutori**, che contengono uno o più algoritmi di soluzione già implementati
- Di nuovo, l'implementazione di un solutore di solito è di una qualità molto migliore di quella che potremmo fare noi in qualche giorno di lavoro
- Inoltre sono stati inventati dei modi standard per fornire i **modelli** in input ai **solutori**, ad esempio attraverso **linguaggi di modellazione**

Linguaggio di Modellazione

- Un linguaggio di modellazione serve ad esprimere un **problema di ottimizzazione** in una forma che sia comprensibile dal **solutore**
- Ne esistono vari, vedremo **AMPL** (A Mathematical Programming Language) che è uno dei più diffusi e ha ispirato vari altri successori
- **AMPL** non è quindi un solutore, ma può chiamare vari solutori, ad esempio **Cplex**, che implementa simplesso, branch-and-bound, branch-and-cut e altro
- È a pagamento, ma esiste la versione **student** limitata ma **gratuita**
- È un file zippato che basta decomprimere e contiene già alcuni solutori
- Scaricabile da https://ampl.com/try-ampl/download-a-free-demo/ o cercando su internet AMPL free demo
- Il libro di AMPL è scaricabile da https://ampl.com/resources/the-ampl-book/

Esprimere un modello in AMPL

- Come in altri linguaggi di programmazione, tutto viene espresso attraverso normali **file di testo**
- Si usa un **file di modello** (*.mod) che contiene la struttura logica del modello ma non i dati (indipendenza dai dati)
- I dati possono essere in un **file di dati** (*.dat) o letti in altri modi
- Al livello base, per esprimere un modello si possono usare **insiemi** (set, insiemi su cui sono organizzati i dati), **parametri** (param, per esprimere i dati), **variabili** (var)
- Occorre dichiarare ciò che serve (insiemi, parametri, variabili, …), e poi usarle il tutto per scrivere la funzione obiettivo e i vincoli nel file .mod
- Per una guida introduttiva alla sintassi vedere

  http://www.dis.uniroma1.it/~bruni/files/AMPLsyntax.pdf

Primo Esempio Introduttivo

- Una azienda produce due prodotti: **standard** e **deluxe**
- Per farlo usa tre risorse: **materia prima**, **levigatura**, **pulitura**, secondo questa tabella

	1kg standard	1kg deluxe
Materia prima	4Kg	4Kg
Levigatura	4 ore	2 ore
Pulitura	2 ore	5 ore
Prezzo vendita	10EUR/Kg	15EUR/Kg

- Le risorse sono limitate: per ogni settimana si può avere al più 75 Kg di materia prima, usare levigatura per al più 80 ore e pulitura per al più 60 ore
- Vogliamo pianificare la produzione in modo da massimizzare i ricavi

Formulazione

- Scelta variabili:

  x_1 = **quantità** in Kg di standard da produrre settimanalmente

  x_2 = **quantità** in Kg di deluxe da produrre settimanalmente

- Il modello è banalmente il seguente **PL**:

$$\max \; 10\,x_1 + 15\,x_2$$

$$4\,x_1 + 4\,x_2 \leq 75$$

$$4\,x_1 + 2\,x_2 \leq 80$$

$$2\,x_1 + 5\,x_2 \leq 60$$

$$x_1 \geq 0$$

$$x_2 \geq 0$$

File .mod (versione rozza)

```
var x1 >=0;
var x2 >=0;
maximize profit: 10*x1 + 15*x2;
s.t. vincolo1: 4*x1 + 4*x2 <= 75;
s.t. vincolo2: 4*x1 + 2*x2 <= 80;
s.t. vincolo3: 2*x1 + 5*x2 <= 60;
```

- Funziona ma è una implementazione di **scarsa qualità**
- Se il problema cresce di dimensione e/o complessità questo stile si rivela inadeguato. Se ho 1000 variabili e 1000 vincoli **non posso** scriverli così: ci metto tanto ed è illeggibile, quindi posso facilmente fare errori
- Vedremo uno stile **compatto** per esprimere modelli più grandi e complessi in modo più pulito e a prova di errore
- Per farlo vediamo alcuni richiami di **algebra lineare**

Moltiplicazione tra matrici

- Le matrici sono insiemi ordinati di numeri organizzate su m **righe** e n **colonne**
- Data una matrice, la sua dimensione è m x n
- Date due matrici di dimensioni m_1 x n_1 e m_2 x n_2 , possiamo moltiplicarle solo se $n_1 = m_2$
- Esempio:

$$A_{3 \times 2} = \begin{pmatrix} 4 & 4 \\ 0 & 2 \\ 2 & 1 \end{pmatrix} \qquad B_{2 \times 3} = \begin{pmatrix} 1 & 3 & 0 \\ 2 & -1 & -1 \end{pmatrix}$$

- Otteniamo una matrice m_1 x n_2 , quindi in questo caso 3 x 3, e ogni elemento di riga i colonna j è la somma dei prodotti tra tutti gli elementi della riga i della prima e della colonna j della seconda
- Esempio l'elemento di posizione 1 1 è 4x1 + 4x2 = 12. Otteniamo

$$C_{3 \times 3} = \begin{pmatrix} 12 & 8 & -4 \\ 4 & -2 & -2 \\ 4 & 5 & -1 \end{pmatrix}$$

Moltiplicazione tra vettori

- I vettori sono insiemi ordinati di numeri organizzati su m **righe** e 1 sola **colonna**
- Possiamo vederli come matrici ad una solo colonna, quindi possiamo moltiplicare matrice m_1 x n_1 per vettore m_2 x 1 , solo se $n_1 = m_2$
- Possiamo anche moltiplicare due vettori se il primo è trasposto e quindi è 1 x n_1 e il secondo m_2 x 1 ha stesso numero di elementi cioè $n_1 = m_2$
- Trasporre vuol dire scambiare righe e colonne, si indica con ^{T} o con ’
- Esempio: $A = \begin{pmatrix} 4 & 4 \\ 4 & 2 \\ 2 & 5 \end{pmatrix}$ $b = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$

otteniamo una matrice 3x1 (un vettore) con le regole viste $c = \begin{pmatrix} 12 \\ 8 \\ 12 \end{pmatrix}$

- Esempio (prodotto interno tra vettori): a = (2, 5, -1, 2) b =(10, 0, 6, 0)^T

otteniamo un singolo numero: 10x2 + 5x0 + -1x6 + 2x0 = 14

Formulazione compatta

- Il modello visto può essere scritto in forma matriciale (o compatta) in questo modo:

$$\max \; c^T x$$

$$Ax \leq b$$

$$x_i \geq 0 \quad i \in \{1,2\}$$

Dove c = (10, 15)^T b =(75, 80, 60)^T $A = \begin{pmatrix} 4 & 4 \\ 4 & 2 \\ 2 & 5 \end{pmatrix}$

- Allora il file in AMPL scritto in modo compatto è nella prossima slide

File .mod

```
set PROD;      # insieme dei prodotti

set RISORSE;   # insieme delle risorse

param b{RISORSE};

param a{RISORSE,PROD};

param c{PROD};

var x{j in PROD} >=0;

maximize profit: sum{j in PROD} c[j]*x[j];

s.t. vincolo{i in RISORSE}: sum{j in PROD} a[i,j]*x[j] <= b[i];
```

Il vettore di vincoli per esteso sarebbe

```
s.t. v1: sum{j in PROD} a[MATERIA,j]*x[j] <= b[MATERIA];
s.t. v2: sum{j in PROD} a[LEVIGATURA,j]*x[j] <= b[LEVIGATURA];
s.t. v3: sum{j in PROD} a[PULITURA,j]*x[j] <= b[PULITURA];
```

File .dat

```
set PROD:=STANDARD, DELUXE;
set RISORSE:=MATERIA, LEVIGATURA, PULITURA;

param:          b:=
MATERIA         75
LEVIGATURA      80
PULITURA        60;

param a:  STANDARD   DELUXE:=
MATERIA          4           4
LEVIGATURA       4           2
PULITURA         2           5;

param:          c:=
STANDARD        10
DELUXE          15;
```

Esecuzione

Eseguire ampl.exe , (fornisce una finestra a riga di comando)

Dare i comandi (notare il punto e virgola alla fine di ogni comando):

option solver cplex; (per scegliere il solutore)

model (path)pian.mod; (per scegliere il file di modello)

data (path)pian.dat; (per scegliere il file di dati)

solve; (per risolvere)

Se i file di modello e di dati si trovano nella stessa cartella non serve il path, altrimenti si

Risultati

In caso di errori, resettare usando il comando reset;

Per correggere errori, cominciare dalla prima segnalazione di errore (viene indicata la riga) risolvere il problema e ripetere il processo.

Da notare che esistono anche errori di cui AMPL non può accorgersi (ad esempio errati valori numerici dei parametri, etc.).

Per vedere infine la soluzione, o ogni altro oggetto di interesse, usare il comando display. Per esempio:

display x; (per vedere il vettore x)

Se tutto è corretto, la soluzione del modello in esame è:

```
x [*] :=
 DELUXE   7.5
 STANDARD  11.25
```

Esempio Gestione Turni

- Un problema che nella pratica capita molto frequentemente è la gestione dei turni di lavoro, detto anche scheduling o timetabling
- Vedremo una versione molto semplificata ma possibile. Abbiamo:
- m = 4 **persone**
- n = 12 **turni** (ad esempio 2 turni al giorno per 6 giorni di una settimana)

Vogliamo coprire tutti i turni **dividendo equamente** il carico e rispettando per quanto possibile le **preferenze** dei lavoratori data nel seguente modo:

s_{ij} = (scontento) misura quanto il lavoratore i vuole evitare il turno j, con valore reale compreso tra 0 e 1

- Dobbiamo decidere come assegnare turni a persone, usiamo **variabili binarie** con **2 indici**

$$x_{ij} = \begin{cases} 1 & \text{se lavoratore } i \text{ svolge turno } j, \text{ con } i \in \{1,..,4\} \text{ e } j \in \{1,..,12\} \\ 0 & \text{altrimenti} \end{cases}$$

Esempio Gestione Turni

- La funzione obiettivo potrebbe essere min $\sum$ scontenti
- Però così qualcuno potrebbe essere scontentato molto…
- Allora è meglio minimizzare il **massimo scontento** S per persona e anche il **massimo numero di turni** N per persona (altre 2 variabili)

$$\min \; S + N$$

- Servono 4 vincoli per dire che tutti i turni devono essere **coperti**

Esempio: per il turno 1, dico che $x_{1\,1} + x_{2\,1} + x_{3\,1} + x_{4\,1} \geq 1$

$$\sum_{i} x_{ij} \geq 1 \qquad \text{per } j \in \{1,..,12\}$$

Esempio Gestione Turni

- Poi servono vincoli per collegare il **massimo scontento** S e il **massimo numero di turni** N alle variabili x

- Esempio: per persona 1 ho $x_{1\,1} + x_{1\,2} + \ldots + x_{1\,12} \leq N$
- Complessivamente:

$$\sum_j x_{ij} \leq N \qquad \text{per } i \in \{1,..,4\}$$

- Esempio: per persona 1 ho $x_{1\,1}\, s_{1\,1} + x_{1\,2}\, s_{1\,2} + \ldots + x_{1\,12}\, s_{1\,12} \leq S$
- Complessivamente:

$$\sum_j s_{ij} x_{ij} \leq S \qquad \text{per } i \in \{1,..,4\}$$

.mod file

```
set P := 1..4;       # persone
set T := 1..12;      # turni

param scontento{P,T} ; # valori di non-preferenza

var x{i in P, j in T} binary; # assegnamento persona i a turno j
var s >= 0;    # massimo scontento di una persona
var n >= 0;    # massimo numero di turni di una persona

minimize carico: s+n;
s.t. copertura{j in T}:  sum{i in P} x[i,j] >= 1;
s.t. turni{i in P}: sum{j in T} x[i,j] <= n;
s.t. nonpreferenza{i in P}: sum{j in T} scontento[i,j] * x[i,j] <= s;
```

.dat file

```
param scontento:   1    2    3    4    5    6    7    8    9    10   11   12 :=
1                  0.3  0.1  0    0.1  0    0    0    0.5  0.1  0    0    0
2                  0    0    0    0.1  0.4  0.1  0.4  0    0    0.5  0.6  0.6
3                  0.5  0.5  0    0    0    0    0    0.5  0.1  0.5  0.8  0.8
4                  0    0    0    0    0    0    0    0    1    1    0.5  0;
```

Soluzione

- Se risolviamo con questi dati troviamo questa **soluzione**

```
x [*,*] (tr)
:   1   2   3   4   :=
1   0   1   0   0
2   0   1   0   0
3   0   0   0   1
4   0   0   1   0
5   1   0   0   0
6   0   0   1   0
7   0   0   1   0
8   0   0   0   1
9   0   1   0   0
10  1   0   0   0
11  1   0   0   0
12  0   0   0   1;

s = 0

n = 3
```

- Vuol dire che il numero **massimo di turni per persona** è **3** (in questo caso tutti ne fanno 3, ma in generale non posso garantire che siano uguali per tutti perché il numero di turni potrebbe non essere divisibile per il numero di persone)
- Il **massimo scontento** per persona è **0** (tutti sono accontentati con le preferenze date, anche qui potrebbe non essere possibile farlo, dipende dalle preferenze)

Esempio di Gestione Scorte

- Dobbiamo gestire un **magazzino**, riceviamo prodotti dalle **fabbriche** e li spediamo ai **negozi**
- Nei prossimi 5 mesi (il nostro **orizzonte temporale**), per ogni merce, abbiamo gli **ordoni** dai negozi. Focalizziamoci su una singola merce

	mese 1	mese 2	mese 3	mese 4	mese 5
ordine	12	4	18	17	7

- Per ogni mese, il **prezzo unitario** della merce dalla fabbrica è il seguente

	mese 1	mese 2	mese 3	mese 4	mese 5
prezzo	8	12	7	11	10

Esempio di Gestione Scorte

- Ogni volta che compriamo dalla fabbrica, dobbiamo anche pagare il seguente **costo di spedizione** (non dipende dalla quantità)

	mese 1	mese 2	mese 3	mese 4	mese 5
Spediz	10	10	10	10	10

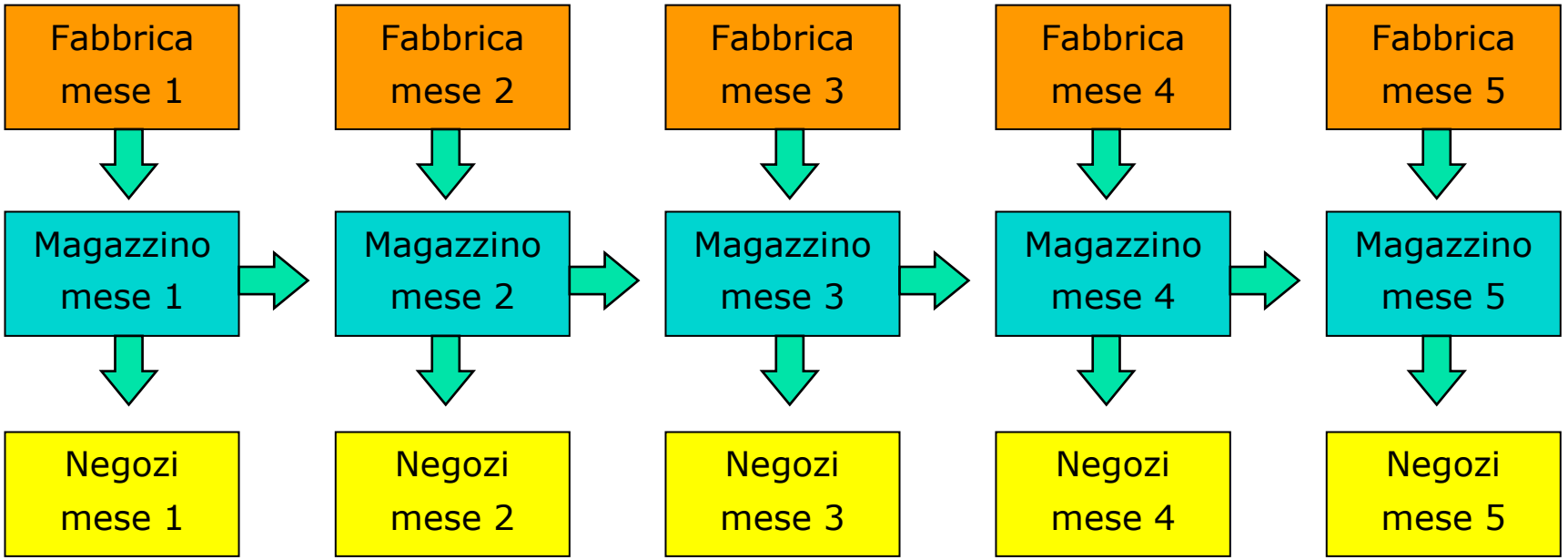
- Infine, tenere la merce in magazzino ha il seguente **costo di magazzino** per unità di merce al mese

	mese 1	mese 2	mese 3	mese 4	mese 5
Magaz	5	5	5	5	5

- Vogliamo soddisfare gli ordini dei negozi e **minimizzare le spese**
- Come possiamo fare?

Il flusso della merce

- Ad ogni mese, possiamo **comprare** dalla fabbrica o **usare** la merce che abbiamo lasciato in magazzino del mese precedente, **spediamo** ai negozi, e posiamo anche **lasciare** merce in magazzino per il futuro



- Il magazzino è **vuoto all'inizio**, e deve **rimanere vuoto alla fine** dell' orizzonte temporale

Connessione tra le variabili

- Cosa possiamo controllare? Cosa vogliamo **decidere**?
- Usiamo due gruppi di variabili **Reali**:

c_i = merce comprata al mese i, con $i \in \{1,2,3,4,5\}$

s_i = merce lasciata in magazzino al mese i, con $i \in \{1,2,3,4,5\}$

Inoltre, ci servono **variabili binarie** per modellare il costo di spedizione

$$t_i = \begin{cases} 1 & \text{se paghiamo trasporto al mese } i, \text{ con } i \in \{1,2,3,4,5\} \\ 0 & \text{altrimenti} \end{cases}$$

- Le variabili sono collegate. Per il generico mese abbiamo:

$$s_i = s_{i-1} + c_i - \text{ordine}(i) \quad i \in \{2,3,4,5\}$$

E per il primo mese: $s_1 = c_1 - \text{ordine}(1)$

- Per modellare i costi di spediz usiamo: $c_i \leq M\, t_i \quad i \in \{1,2,3,4,5\}$

Funzione Obiettivo

- Vogliamo minimizzare la somma di tutte le spese
- Quanti **tipi** di spese?
- Costo della merce (per unità) : $Cost_i$
- Costo di Spedizione : $Trans_i$
- Costo di magazzino (per unità di merce al mese): $Storage_i$
- Quindi il **costo totale** è:

$$\min \sum_{i=1..5} Cost_i \; c_i \quad + \sum_{i=1..5} Trans_i \; t_i \quad + \sum_{i=1..5} Storage_i \; s_i$$

Il modello complessivo

$$
\begin{cases}
\min \sum_{i=1..5} Cost_i \; c_i \quad + \sum_{i=1..5} Trans_i \; t_i \quad + \sum_{i=1..5} Storage_i \; s_i \\
s_1 = c_1 \text{ - ordine}(1) \\
s_i = s_{i-1} + c_i \text{ - ordine}(i) \quad i \in \{2,3,4,5\} \\
c_i \leq M \, t_i \quad i \in \{1,2,3,4,5\} \\
c_i \geq 0 \quad i \in \{1,2,3,4,5\} \\
s_i \geq 0 \quad i \in \{1,2,3,4,5\} \\
t_i \in \{0,1\} \quad i \in \{1,2,3,4,5\}
\end{cases}
$$

- È un modell di **programmazione lineare mista intera**: alcune variabili sono reali e alcune sono intere (in particolare binarie)

.mod file

```
set O := 1..5;       # temporal horizon
param Order{O} ; # commodity needed
param Cost{O};    # unit cost of commodity
param Trans{O}; # transportation cost
param Storage{O}; # storage cost per unit
param M;             # big value for the fixed cost constraints
var c{i in O} >= 0; # commodity bought in month i
var s{i in O} >= 0; # commodity stocked at month i
var t{i in O} binary; # 1 if we must pay transportation , 0 otherwise
minimize total_cost: sum{i in O} Cost[i] * c[i] +
sum{i in O} Trans[i] * t[i] +
sum{i in O} Storage[i] * s[i];
s.t. month_1: s[1] = c[1] - Order[1];
s.t. month_i{i in O: i>1}: s[i] = s[i-1] + c[i] – Order[i];
s.t. transportation{i in O}: M * t[i] >= c[i];
```

.dat file

```
param Order:=
1       12
2       4
3       18
4       17
5       7;

param Cost :=
1       8
2       12
3       7
4       11
5       10;
```

```
param Trans :=
1       10
2       10
3       10
4       10
5       10;

param Storage :=
1       5
2       5
3       5
4       5
5       5;

param M := 60;
```

Soluzione

OBJECTIVES:
costi = 571

VARIABLES:
c [*] :=
1 16
2 0
3 18
4 17
5 7;

s [*] :=
1 4
2 0
3 0
4 0
5 0;
t [*] :=
1 1
2 0
3 1
4 1
5 1;

- Questo vuol dire che la soluzione ottima è **comprare** nei mesi 1, 3, 4, 5 e usare il **magazzino** per il mese 2. Ovviamente può cambiare quando cambiano i costi di spedizione e magazzino
- Possiamo vedere che succede variando i costi