Assignment 1 - Linear Programming

Guglielmo Muoio matr. 826029

Contents

The problem

A construction materials company is looking for a way to maximize profit per transportation of their goods. The company has a train available with 4 wagons.

When stocking the wagons they can choose among 3 types of cargo, each with its own specifications. How much of each cargo type should be loaded on which wagon in order to maximize profit?

More data

TRAIN WAGON j	WEIGHT CAPACITY (TONNE) w_j	VOLUME CAPACITY (m^2) s_j
(wag) 1	10	5000
(wag) 2	8	4000
(wag) 3	12	8000
(wag) 4	6	2500

CARGO TYPE i	AVAILABLE (TONNE) a_i	VOLUME (m^2/t) v_i	PROFIT (PER TONNE) p_i
(cg) 1	20	500	3500
(cg) 2	10	300	2500
(cg) 3	18	400	2000

The decision variables

Le variabili decisionali sono definite dalla coppia cargo-vagone e quindi corrispondono a tutte le possibili combinazioni fra la tipologia di cargo (i) e il numero del vagone (j). In termini analitici:

x_{ij} indica la quantità di cargo i allocata nel vagone j dove $i \in \{1, 2, 3\}$ e $j \in \{1, 2, 3, 4\}$

The objective function

Il problema di programmazione lineare è impostato in termini di massimizzazione del profitto P. Il profitto per tonnellata dipende dalla tipologia di cargo allocata e quindi il coefficiente associato alle variabili decisionali dipende esclusivamente dal pedice i; in particolare $\forall j$:

- a x_{1j} si associa un coefficiente $p_1 = 3500$
- a x_{2j} si associa un coefficiente $p_2 = 2500$
- a x_{3j} si associa un coefficiente $p_3 = 2000$

La funzione obiettivo estesa risulta essere

$P = 3500x_{11}+2500x_{21}+2000x_{31}+3500x_{12}+2500x_{22}+2000x_{32}+3500x_{13}+2500x_{23}+2000x_{33}+3500x_{14}+2500x_{24}+2000x_{34}$

oppure semplificata

$$P(x) = \sum_{i=1}^{3} \sum_{j=1}^{4} p_i x_{ij}$$

The constraints

Le specifiche tecniche dei cargo e dei vagoni portano ad avere una serie di vincoli di cui si possono identificare 3 categorie:

- vincoli di peso. Il peso complessivo (in tonnellate) di cargo allocato in ciascun vagone non deve superare la capacità di peso del vagone stesso

$$\sum_{i=1}^{3} x_{ij} \leq w_j \quad \forall j \in \{1, 2, 3, 4\}$$

- vincoli di volume. Il volume complessivo di cargo allocato in ciascun vagone non deve superare la capacità di volume del vagone stesso

$$\sum_{i=1}^{3} v_i x_{ij} \leq s_j \quad \forall j \in \{1, 2, 3, 4\}$$

- vincoli di disponibilità. Per ogni tipologia di cargo, la quantità complessiva allocata sul treno non deve superare la quantità disponibile

$$\sum_{j=1}^{4} x_{ij} \leq a_i \quad \forall i \in \{1,2,3\}$$

Oltre a questi 11 vincoli è importante ricordarsi che ogni variabile decisionale deve essere non negativa

$$x_{ij} \geq 0 \quad \forall i \in \{1,2,3\}; \forall j \in \{1,2,3,4\}$$

Building the model

Definite le variabili decisionali, la funzione obiettivo e i vincoli possiamo formulare il problema di programmazione lineare come segue

$$\max P(x) = \sum_{i=1}^{3}\sum_{j=1}^{4} p_i x_{ij}$$

$$t.c. \sum_{i=1}^{3} x_{ij} \leq w_j \quad \forall j \in \{1,2,3,4\}$$

$$\sum_{i=1}^{3} v_i x_{ij} \leq s_j \quad \forall j \in \{1,2,3,4\}$$

$$\sum_{j=1}^{4} x_{ij} \leq a_i \quad \forall i \in \{1,2,3\}$$

$$x_{ij} \geq 0 \quad \forall i \in \{1,2,3\}; \forall j \in \{1,2,3,4\}$$

Per risolvere il problema si fa ricorso ad un software computazionale, in questo caso R. Innanzitutto si caricano le librerie necessarie e si crea un modello di programmazione lineare a 12 variabili

```
library(lpSolveAPI)
model <- make.lp(0,12)
model
## Model name:
##   a linear program with 12 decision variables and 0 constraints
```

Si definisce la funzione obiettivo e si imposta la massimizzazione di questa

```
objective <- rep(c(3500,2500,2000),4)
set.objfn(model, obj = objective)
lp.control(model,sense='max')
```

Si inseriscono i vincoli. Prima quelli di peso

```
add.constraint(model,
               xt=c(1,1,1),
               type="<=",rhs=10,
               indices=c(1:3))  # vincolo 1
add.constraint(model,
               xt=c(1,1,1),
```

```
                type="<=",rhs=8,
                indices=c(4:6))  # vincolo 2
add.constraint(model,
                xt=c(1,1,1),
                type="<=",rhs=12,
                indices=c(7:9))  # vincolo 3
add.constraint(model,
                xt=c(1,1,1),
                type="<=",rhs=6,
                indices=c(10:12))  # vincolo 4
```

Poi quelli di volume

```
add.constraint(model,
                xt=c(500,300,400),
                type="<=",rhs=5000,
                indices=c(1:3))  # vincolo 5
add.constraint(model,
                xt=c(500,300,400),
                type="<=",rhs=4000,
                indices=c(4:6))  # vincolo 6
add.constraint(model,
                xt=c(500,300,400),
                type="<=",rhs=8000,
                indices=c(7:9))  # vincolo 7
add.constraint(model,
                xt=c(500,300,400),
                type="<=",rhs=2500,
                indices=c(10:12))  # vincolo 8
```

Infine quelli di disponibilità

```
add.constraint(model,
                xt=c(1,1,1,1),
                type="<=",rhs=20,
                indices=c(1,4,7,10)) # vincolo 9
add.constraint(model,
                xt=c(1,1,1,1),
                type="<=",rhs=10,
                indices=c(2,5,8,11)) # vincolo 10
add.constraint(model,
                xt=c(1,1,1,1),
                type="<=",rhs=18,
                indices=c(3,6,9,12)) # vincolo 11
```

Si inserisce anche la non negatività di tutte le variabili decisionali

```
set.bounds(model,lower=rep(0,12))
```

Per visualizzare il modello con i rispettivi vincoli è necessario utilizzare il seguente codice al fine di creare un file all'interno della stessa working directory. Questo si può successivamente aprire con qualsiasi editor di testo per visualizzare il codice commentato sottostante

```
write.lp(model,'model.lp')
# /* Objective function */
# max: +3500 C1 +2500 C2 +2000 C3 +3500 C4 +2500 C5 +2000 C6 +3500 C7 +2500 C8
# +2000 C9 +3500 C10 +2500 C11 +2000 C12;
#
# /* Constraints */
# +C1 +C2 +C3 <= 10;
# +C4 +C5 +C6 <= 8;
# +C7 +C8 +C9 <= 12;
# +C10 +C11 +C12 <= 6;
# +500 C1 +300 C2 +400 C3 <= 5000;
# +500 C4 +300 C5 +400 C6 <= 4000;
# +500 C7 +300 C8 +400 C9 <= 8000;
# +500 C10 +300 C11 +400 C12 <= 2500;
```

Si risolve quindi il modello

```
solution <- solve(model)
solution #se uguale a 0, si è trovata una soluzione ottima
## [1] 0
```

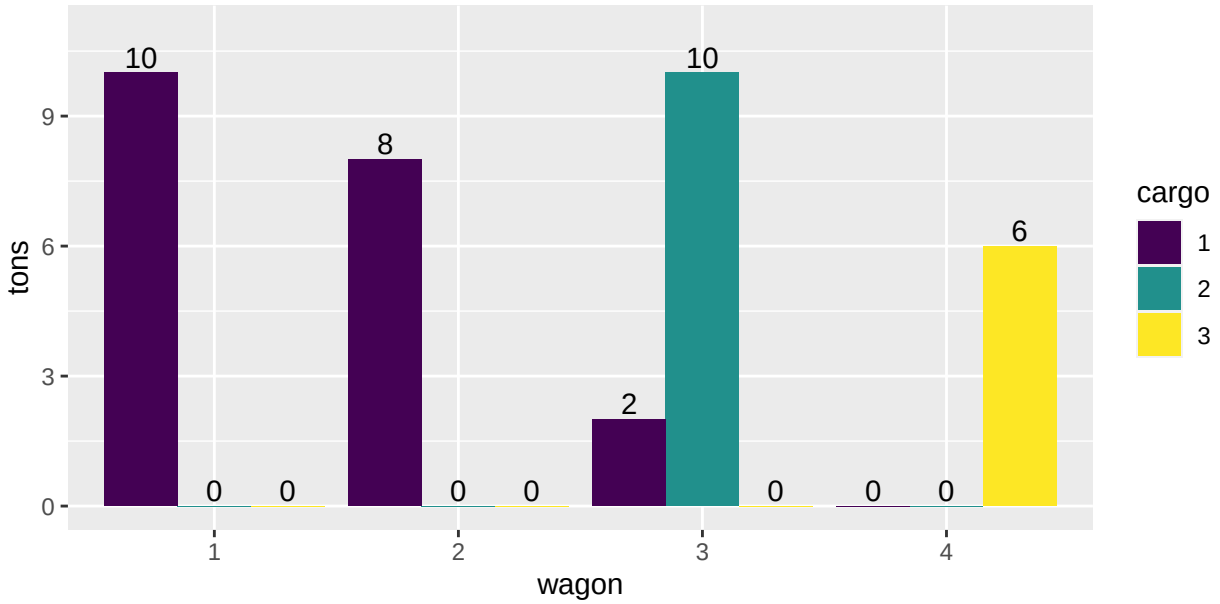
Per la soluzione ottima, il profitto è pari a

```
get.objective(model)
## [1] 107000
```

Le variabili decisionali assumono i seguenti valori

```
get.variables(model)
##  [1] 10  0  0  8  0  0  2 10  0  0  0  6
```

```
##  [1] 10  0  0  8  0  0  2 10  0  0  0  6
```



Si verificano i vincoli

```
get.constraints(model)
##  [1]   10    8   12    6 5000 4000 4000 2400   20   10    6
```

Si identificano i vincoli 7, 8 e 11 come non attivi in quanto non saturi. Infatti nonostante i valori RHS di questi vincoli siano rispettivamente 8000, 2500 e 18, i valori raggiunti dalla soluzione ottima risultano essere 4000, 2400 e 6. Questo si può verificare anche dalla base ottima.

```
get.basis(model,nonbasic = F)
```

```
##  [1] -13 -16 -19 -23 -12 -15  -7  -8 -18 -22 -11
```

-7, -8, -11 corrispondono ai vincoli non attivi

Sensitivity analysis

Si opera un'analisi di sensitività sulla funzione obiettivo e sui valori RHS per valutare la sensibilità della soluzione ottima alla variazione di parametri e/o coefficienti (si utilizzano funzioni personalizzate).

```
printSensitivityObj(model)
```

```
##    Objs                    Sensitivity
## 1    C1 3500 <= C1 <= 4833.33333333333
## 2    C2              -inf <= C2 <= 2500
## 3    C3              -inf <= C3 <= 2000
## 4    C4 3500 <= C4 <= 4833.33333333333
## 5    C5              -inf <= C5 <= 2500
## 6    C6              -inf <= C6 <= 2000
## 7    C7              3500 <= C7 <= 3500
## 8    C8              2500 <= C8 <= 2500
## 9    C9              -inf <= C9 <= 2000
## 10  C10             -inf <= C10 <= 3500
## 11  C11             -inf <= C11 <= 2500
## 12  C12             2000 <= C12 <= 2500
```

```
printSensitivityRHS(model)
```

```
##    Rhs         Sensitivity
## 1   B1      10 <= B1 <= 10
## 2   B2       8 <= B2 <= 8
## 3   B3      6 <= B3 <= 12
## 4   B4    0 <= B4 <= 6.25
## 5   B5 3000 <= B5 <= 5000
## 6   B6 2400 <= B6 <= 4000
## 7   B7  -inf <= B7 <= inf
## 8   B8  -inf <= B8 <= inf
## 9   B9     20 <= B9 <= 26
## 10 B10    10 <= B10 <= 16
## 11 B11 -inf <= B11 <= inf
```

```
## 12 B12 -inf <= B12 <= inf
## 13 B13 -inf <= B13 <= inf
## 14 B14      0 <= B14 <= 0
## 15 B15 -inf <= B15 <= inf
## 16 B16 -inf <= B16 <= inf
## 17 B17      0 <= B17 <= 0
## 18 B18 -inf <= B18 <= inf
## 19 B19 -inf <= B19 <= inf
## 20 B20      0 <= B20 <= 6
## 21 B21    -10 <= B21 <= 0
## 22 B22 -inf <= B22 <= inf
## 23 B23 -inf <= B23 <= inf
```

Si controllano i prezzi ombra. Il prezzo ombra di un vincolo viene definito come la quantità di cui il valore della funzione obiettivo cambia dato un aumento unitario del valore RHS del vincolo, assumendo che tutti gli altri coefficienti rimangano costanti.

```
get.dual.solution(model)
```

```
##  [1]    1 2000 2000 2000 2000    0    0    0    0 1500  500    0    0    0    0
## [16]    0    0    0    0    0    0    0    0    0
```

Dalla sensitivity analysis di RHS, il valore per il vincolo 9 può variare da 20 a 26 senza modificare la soluzione ottima. Sulla base del prezzo ombra ottenuto ($= +1500$), si verifica l'invarianza della base ottima. Si modifica il valore rhs del vincolo 9 da 20 a 24 e si ricalcola la soluzione.

```
set.constr.value(model, constraints = 9, rhs = 24)
solve(model)
```

```
## [1] 0
```

Tecnicamente, la funzione obiettivo dovrebbe aumentare di $prezzo\ ombra \cdot \Delta unità = 1500 \cdot 4 = 6000$

```
get.objective(model)
## [1] 113000
```

che è uguale a $107000 + 6000$

I valori delle variabili decisionali cambiano

```
get.variables(model)
##  [1] 10  0  0  8  0  0  6  6  0  0  4  2
```

E anche quelli dei vincoli

```
get.constraints(model)
##  [1]   10    8   12    6 5000 4000 4800 2000   24   10    2
```

Infine, come da aspettative, la base ottima non è cambiata

```
get.basis(model,nonbasic = F)
```

```
##  [1] -13 -16 -19 -23 -12 -15  -7  -8 -18 -22 -11
```

Questions about LP

1. Can an LP model have more than one optimal solution. Is it possible for an LP model to have exactly two optimal solutions? Why or why not?

In un caso particolare, denominato *Alternate Optimal Solution*, un modello di programmazione lineare può avere più di una soluzione ottima. Questo succede quando la curva di livello della funzione obiettivo è parallela al vincolo attivo e diventano quindi ottimali tutti i punti appartenenti al segmento compreso tra i due vertici ottimali (soluzioni di base). Tuttavia, risulta impossibile per un problema LP avere esattamente due soluzioni ottime.

2. Are the following objective functions for an LP model equivalent? That is, if they are both used, one at a time, to solve a problem with exactly the same constraints, will the optimal values for x_1, x_2 and x_3 be the same in both cases? Why or why not?

$$\max 2x_1 + 3x_2 - x_3$$

$$\min -2x_1 - 3x_2 + x_3$$

Considerando gli stessi vincoli, la soluzione ottima per le funzioni obiettivo soprastanti è la medesima. Questo perché le due funzioni obiettivo, nonostante sembrino apparentemente diverse, coincidono. Infatti, massimizzare una funzione è equivalente a minimizzare la stessa funzione con tutti i termini cambiati di segno.

$$max[f(x)] = min[-f(x)]$$

3. Which of the following constraints are not linear or cannot be included as a constraint in a linear programming problem?

a. $2x_1 + x_2 - 3x_3 \geq 50 \rightarrow lineare$

b. $2x_1 + \sqrt{x_2} \geq 60 \rightarrow non\ lineare$

c. $4x_1 - \frac{1}{2}x_2 = 75 \rightarrow lineare$

d. $\frac{3x_1+2x_2x_1-3x_3}{x_1+x_2+x_3} \leq 0.9 \rightarrow non\ lineare$

e. $3x_1^2 + 7x_2 \leq 45 \rightarrow non\ lineare$