ROAR III

Ricerca Operativa Applicazioni Reali

Alessandro Gobbi Alice Raffaele Gabriella Colajanni Eugenia Taranto

IIS Antonietti, Iseo (BS)

5 novembre 2022

Introduzione

Chi siamo e i nostri contatti

Alessandro Gobbi (UniBS)
alessandro.gobbi@unibs.it

Alice Raffaele (UniVR)
alice.raffaele@univr.it

Gabriella Colajanni (UniCT)
gabriella.colajanni@unict.it

Eugenia Taranto (UniCT)
eugenia.taranto@unict.it

Parte 1:
Funzionalità più avanzate di Python e PuLP

Assegnazione di incarichi

Una compagnia finanziaria deve decidere chi assumere fra i tre candidati C1, C2 e C3. In base ai loro differenti curriculum, l'azienda sa che, in caso di assunzione, dovrà assicurare loro uno stipendio mensile fisso, rispettivamente di 1450, 1600 e 1300 euro. Inoltre, nel mese corrente, la compagnia ha necessità di portare a termine tre progetti (LAV1, LAV2, LAV3) che richiedono diverse abilità ed esperienza. Al progetto LAV1 dovranno essere assegnate almeno 2 persone, agli altri due progetti almeno 1 persona ciascuno. In base all'assegnazione dei lavori ai candidati, la compagnia finanziaria dovrà retribuire i dipendenti con uno o più bonus in busta paga. La stima di tale bonus (in €), riferito a ciascun candidato se fosse assegnato a ciascuno dei tre lavori, è riportata nella tabella seguente:

	LAV1	LAV2	LAV3
C1	150	230	110
C2	100	90	150
C3	350	410	210

Costruire un modello di Programmazione Lineare che decida quali persone assumere e come assegnare gli incarichi, minimizzando i costi che l'azienda dovrà sostenere nel mese corrente.

Variabili (I)

$$y_1 = \begin{cases} 1 \text{ se il candidato 1 viene assunto} \\ 0 \text{ se il candidato 1 non viene assunto} \end{cases}$$

$$y_2 = \begin{cases} 1 \text{ se il candidato 2 viene assunto} \\ 0 \text{ se il candidato 2 non viene assunto} \end{cases}$$

$$y_3 = \begin{cases} 1 \text{ se il candidato 3 viene assunto} \\ 0 \text{ se il candidato 3 non viene assunto} \end{cases}$$

Variabili (II)

$$x_{11} = \begin{cases} 1 \text{ se il candidato 1 svolge il lavoro 1} \\ 0 \text{ se il candidato 1 non svolge il lavoro 1} \end{cases}$$

$$x_{12} = \begin{cases} 1 \text{ se il candidato 1 svolge il lavoro 2} \\ 0 \text{ se il candidato 1 non svolge il lavoro 2} \end{cases}$$

$$\dots$$

$$x_{ij} = \begin{cases} 1 \text{ se il candidato } i \text{ svolge il lavoro } j \\ 0 \text{ se il candidato } i \text{ non svolge il lavoro } j \end{cases}$$

Quante variabili in totale sono necessarie?

Modello

$$\min \quad 1450y_1 + 1600y_2 + 1300y_3 +$$

$$+150x_{11}+230x_{12}+110x_{13}+100x_{21}+90x_{22}+150x_{23}+350x_{31}+410x_{32}+210x_{33}$$

$$x_{11} + x_{21} + x_{31} \geq 2$$

$$x_{12} + x_{22} + x_{32} \geq 1$$

$$x_{13} + x_{23} + x_{33} \geq 1$$

$$x_{11} + x_{12} + x_{13} \leq 3y_1$$

$$x_{21} + x_{22} + x_{23} \leq 3y_2$$

$$x_{31} + x_{32} + x_{33} \leq 3y_3$$

$$x_{ij} \in \{0,1\}, \forall i \in \{1,2,3\}, \forall j \in \{1,2,3\}$$

$$y_j \in \{0,1\}, \forall j \in \{1,2,3\}$$

Dizionari (I)

L'indicizzazione delle liste in Python è numerica e progressiva, dove il primo elemento è identificato dall'indice `0`. Per esempio, definendo la lista `V = [45, 89, 29, 101]`, `V[0]` corrisponde all'elemento in prima posizione (45) e `V[3]` all'elemento in quarta posizione (101).

Talvolta, è però utile creare un'*indicizzazione personalizzata* di una lista, associando a ogni elemento una *chiave univoca*:

```
D = {chiave1: elem1, chiave2: elem2, …, chiaveN: elemN}
```

- Questo tipo particolare di lista è detto **dizionario**.
- Ogni elemento di un dizionario è accessibile tramite la sua chiave e **non** tramite la classica indicizzazione numerica da `0` a `len(D)-1`.
- Le chiavi di un dizionario possono essere numeri o stringhe.

Facciamo qualche esempio!

Dizionari (II)

```
costanti = {"pigreco": 3.14, "nepero": 2.71, "accelerazione": 9.8}
```

In questo caso, abbiamo utilizzato le tre stringhe `pigreco`, `nepero` e `accelerazione` come chiavi. Per accedere agli elementi del dizionario `costanti`, ossia `3.14`, `2.71` e `9.8`, scriveremo, rispettivamente, `costanti["pigreco"]`, `costanti["nepero"]` e `costanti["accelerazione"]`.

`costanti["pigreca"]` non esiste! Non c'è alcun elemento associato alla chiave `"pigreca"`: `KeyError`

Dizionari (III)

```
diz1 = {4: 100, 3: 34, 1: 101, 10: 33}
diz2 = {1: 63, 2: 89, 3: 11, 4: 102}
```

In questi ultimi due casi, si utilizzano delle chiavi numeriche. Per accedere agli elementi di `diz1` utilizzeremo le chiavi `4`, `3`, `1` e `10`: per esempio, `diz1[1]` corrisponderà all'elemento `101`.

Per accedere agli elementi di `diz2`, si usano invece le chiavi `1`, `2`, `3`, `4`. **Nota**: con questa particolare scelta di chiavi, abbiamo di fatto introdotto un'indicizzazione numerica progressiva che parte da `1` fino a `len(diz2)`:

`diz2[1]` corrisponderà all'elemento 63
`diz2[2]` corrisponderà all'elemento 89
`diz2[3]` corrisponderà all'elemento 11
`diz2[4]` corrisponderà all'elemento 102

`diz2[0]` non esiste! Non c'è alcun elemento associato alla chiave `0`: `KeyError`

Dizionari (IV)

È possibile definire anche dizionari bidimensionali, dove ogni elemento è identificato da una coppia di chiavi. Supponiamo per esempio di voler definire la seguente matrice bidimensionale

```
m = [[80,75,85,90,95],
     [75,80,75,85,100],
     [80,80,80,90,95]]
```

come un dizionario bidimensionale.
Una possibile soluzione potrebbe essere:

Dizionari (V)

```
d = {1: {1: 80, 2: 75, 3: 85, 4: 90, 5: 95},
     2: {1: 75, 2: 80, 3: 75, 4: 85, 5: 100},
     3: {1: 80, 2: 80, 3: 80, 4: 90, 5: 95}}
```

Così facendo, avremo che, per esempio:

`d[1][1]` corrisponderà all'elemento 80
`d[2][5]` corrisponderà all'elemento 100
`d[3][3]` corrisponderà all'elemento 80

...

e così via

`d[5][3]` non esiste! Non c'è alcun elemento associato alla coppia di chiavi `(5,3)`:

`KeyError`

Nota: `m[2][4]` non indicizza lo stesso elemento di `d[2][4]`

dicts

Per creare un *dizionario* (un raggruppamento) di variabili, ciascuna identificata da una **chiave univoca**, ma accomunate dalla stessa tipologia e dallo stesso limite inferiore e superiore, in PuLP si può usare *dicts*:

```
dicts(name, indices=None, lowBound=None,
upBound=None, cat='...')
```

- *name* = prefisso comune al nome delle variabili.
- *indices* = lista di **chiavi** (stringhe), una per ogni variabile.
- *lowBound* = limite inferiore delle variabili.
- *upBound* = limite superiore delle variabili.
- *cat* = categoria delle variabili.

```
# Variabili
#y_1 = LpVariable("y_1", lowBound=0, cat=LpBinary)
#y_2 = LpVariable("y_2", lowBound=0, cat=LpBinary)
#y_3 = LpVariable("y_3", lowBound=0, cat=LpBinary)
var_y = LpVariable.dicts("y", indici_candidati, 0, 1, LpBinary);

#x_11 = LpVariable("x_11", lowBound=0, cat=LpBinary)
#x_12 = LpVariable("x_12", lowBound=0, cat=LpBinary)
#x_13 = LpVariable("x_13", lowBound=0, cat=LpBinary)
#x_21 = LpVariable("x_21", lowBound=0, cat=LpBinary)
#x_22 = LpVariable("x_22", lowBound=0, cat=LpBinary)
#x_23 = LpVariable("x_23", lowBound=0, cat=LpBinary)
#x_31 = LpVariable("x_31", lowBound=0, cat=LpBinary)
#x_32 = LpVariable("x_32", lowBound=0, cat=LpBinary)
#x_33 = LpVariable("x_33", lowBound=0, cat=LpBinary)
var_x = LpVariable.dicts("x", (indici_candidati, indici_lavori), 0, 1, LpBinary);
```

lpSum

Per calcolare la somma di una lista di espressioni lineari, si può usare la funzione *lpSum*:

pulp.lpSum(vector)

- *vector* = lista di espressioni lineari

```
# Funzione obiettivo
#model += y_1 * 1450 + y_2 * 1600 + y_3 * 1300 + x_11 * 150 + x_12 * 230 + x_13 * 110 +
    x_21 * 100 + x_22 * 90 + x_23 * 150 + x_31 * 350 + x_32 * 410 + x_33 * 210
model += lpSum(var_y[i] * stipendi[i] for i in indici_candidati) + lpSum(var_x[i][j] *
    bonus[i][j] for i in indici_candidati for j in indici_lavori)
```

```
# Vincoli
#model += x_11 + x_12 + x_13 <= 3 * y_1
#model += x_21 + x_22 + x_23 <= 3 * y_2
#model += x_31 + x_32 + x_33 <= 3 * y_3
for i in indici_candidati:
    model += lpSum(var_x[i][j] for j in indici_lavori) <= 3 * var_y[i]

#model += x_11 + x_21 + x_31 >= 2
#model += x_12 + x_22 + x_32 >= 1
#model += x_13 + x_23 + x_33 >= 1
for j in indici_lavori:
    model += lpSum(var_x[i][j] for i in indici_candidati) >= num_min_candidati_lavori[j]
```

Parte 2:
Lavoro di gruppo (40 minuti)

La dieta mediterranea (I)

Secondo un nutrizionista sostenitore della dieta mediterranea, le quantità minime di nutrienti che devono essere assunte ogni giorno sono 1700 chilocalorie, 200 g di carboidrati, 70 g di proteine, 60 g di grassi e 0,7 g di calcio. Generalmente, il nutrizionista è solito prescrivere una dieta composta da otto alimenti: pane, pasta, latte, uova, pollo, tonno, cioccolato e verdure. La seguente tabella mostra quante calorie (in Kcal), carboidrati, proteine, grassi (in grammi) e calcio (in mg) fornisce una porzione di ogni alimento:

	Pane	Pasta	Latte	Uova	Pollo	Tonno	Cioccolato	Verdure
Calorie (kcal)	150	390	70	70	150	150	112	45
Carboidrati (g)	30	75	5	0	2	0	7	8
Proteine (g)	5	11	5	6	36	25	2	3
Grassi (g)	2	3	3	6	5	15	10	2
Calcio (mg)	52	5	150	50	22	4	11	50

La dieta mediterranea (II)

Il nutrizionista raccomanda anche almeno due porzioni di verdura al giorno, mentre il numero massimo di porzioni per ogni alimento è riportato nella tabella seguente:

	Pane	Pasta	Latte	Uova	Pollo	Tonno	Cioccolato	Verdure
N° max di porzioni	2	2	2	1	1	2	2	6

Il numero di porzioni di pane e pasta non può essere maggiore di 3, mentre quello delle porzioni di latte, pollo e tonno deve essere almeno 4. Inoltre, il nutrizionista vuole che nella dieta ci siano esattamente 7 degli otto alimenti proposti. Il costo (in €) di una porzione di ogni alimento è il seguente:

	Pane	Pasta	Latte	Uova	Pollo	Tonno	Cioccolato	Verdure
Costo per porzione	0,50	3,50	1,00	1,50	4,50	2,00	1,50	4,00

Determinare quali alimenti dovranno essere mangiati in un giorno per rispettare tutte le prescrizioni della dieta indicata dal nutrizionista, in modo tale da minimizzare il costo totale.

Variabili, vincoli e funzione obiettivo (I)

Variabili:

- $x_{PAN}, x_{PAS}, x_{LAT}, x_{UOV}, x_{POL}, x_{TON}, x_{CIO}, x_{VER} \geq 0$: rappresentano il numero di porzioni di ogni alimenti da inserire nella dieta;
- $y_{PAN}, y_{PAS}, y_{LAT}, y_{UOV}, y_{POL}, y_{TON}, y_{CIO}, y_{VER} \in \{0, 1\}$: ogni variabile binaria indica se l'alimento corrispondente è inserito o no nella dieta.

Vincoli:

- Calorie:
  $150x_{PAN} + 390x_{PAS} + 70x_{LAT} + 70x_{UOV} + 150x_{POL} + 150x_{TON} + 112x_{CIO} + 45x_{VER} \geq 1700$
- Carboidrati:
  $30x_{PAN} + 75x_{PAS} + 5x_{LAT} + 0x_{UOV} + 2x_{POL} + 0x_{TON} + 7x_{CIO} + 8x_{VER} \geq 200$
- Proteine:
  $5x_{PAN} + 11x_{PAS} + 5x_{LAT} + 6x_{UOV} + 36x_{POL} + 25x_{TON} + 2x_{CIO} + 3x_{VER} \geq 70$
- Grassi: $2x_{PAN} + 3x_{PAS} + 3x_{LAT} + 6x_{UOV} + 5x_{POL} + 15x_{TON} + 10x_{CIO} + 2x_{VER} \geq 60$
- Calcio:
  $52x_{PAN} + 5x_{PAS} + 150x_{LAT} + 50x_{UOV} + 22x_{POL} + 4x_{TON} + 11x_{CIO} + 50x_{VER} \geq 700$

Variabili, vincoli e funzione obiettivo (II)

- **Numero massimo di porzioni**: $x_{PAN} \leq 2y_{PAN}$; $x_{PAS} \leq 2y_{PAS}$; $x_{LAT} \leq 2y_{LAT}$; $x_{UOV} \leq 1y_{UOV}$; $x_{POL} \leq 1y_{POL}$; $x_{TON} \leq 2y_{TON}$; $x_{CIO} \leq 2y_{CIO}$; $x_{VER} \leq 6y_{VER}$.
- **Porzioni e alimenti**:
  - Verdure: $x_{VER} \geq 2$;
  - Pane e pasta: $x_{PAN} + x_{PAS} \leq 3$;
  - Pollo e tonno: $x_{LAT} + x_{POL} + x_{TON} \geq 4$.
- **Varietà**: $y_{PAN} + y_{PAS} + y_{LAT} + y_{UOV} + y_{POL} + y_{TON} + y_{CIO} + y_{VER} = 7$.

Funzione obiettivo: minimizzare i costi totali

$$0,50x_{PAN} + 3,50x_{PAS} + 1x_{LAT} + 1,50x_{UOV} + 4,50x_{POL} + 2x_{TON} + 1,50x_{CIO} + 4x_{VER}.$$

Modello completo

$$\textbf{min}\ 0,50x_{PAN} + 3,50x_{PAS} + 1x_{LAT} + 1,50x_{UOV} + 4,50x_{POL} + 2x_{TON} + 1,50x_{CIO} + 4x_{VER}$$

$$150x_{PAN} + 390x_{PAS} + 70x_{LAT} + 70x_{UOV} + 150x_{POL} + 150x_{TON} + 112x_{CIO} + 45x_{VER} \geq 1700$$

$$30x_{PAN} + 75x_{PAS} + 5x_{LAT} + 0x_{UOV} + 2x_{POL} + 0x_{TON} + 7x_{CIO} + 8x_{VER} \geq 200$$

$$5x_{PAN} + 11x_{PAS} + 5x_{LAT} + 6x_{UOV} + 36x_{POL} + 25x_{TON} + 2x_{CIO} + 3x_{VER} \geq 70$$

$$2x_{PAN} + 3x_{PAS} + 3x_{LAT} + 6x_{UOV} + 5x_{POL} + 15x_{TON} + 10x_{CIO} + 2x_{VER} \geq 60$$

$$52x_{PAN} + 5x_{PAS} + 150x_{LAT} + 50x_{UOV} + 22x_{POL} + 4x_{TON} + 11x_{CIO} + 50x_{VER} \geq 700$$

$$x_{PAN} \leq 2y_{PAN}$$

$$x_{PAS} \leq 2y_{PAS}$$

$$x_{LAT} \leq 2y_{LAT}$$

$$x_{UOV} \leq 1y_{UOV}$$

$$x_{POL} \leq 1,5y_{POL}$$

$$x_{TON} \leq 1,5y_{TON}$$

$$x_{CIO} \leq 1y_{CIO}$$

$$x_{VER} \leq 5y_{VER}$$

$$x_{VER} \geq 2$$

$$x_{PAN} + x_{PAS} \leq 3$$

$$x_{LAT} + x_{POL} + x_{TON} \geq 4$$

$$y_{PAN} + y_{PAS} + y_{LAT} + y_{UOV} + y_{POL} + y_{TON} + y_{CIO} + y_{VER} = 7$$

.

Implementare e risolvere il modello matematico del problema con PuLP+Python, utilizzando le nuove funzionalità introdotte.

Parte 3:
Lavoro di gruppo (1 ora e 15)

“Buongiornissimo! Kaffè?”

Una catena di bar ha stipulato un contratto commerciale con un’industria di torrefazione per la fornitura esclusiva di caffè. L’industria deve decidere quali dei suoi quattro impianti di torrefazione T_1, T_2, T_3, T_4 aprire per rifornire i tre bar B_1, B_2 e B_3 della catena. Vista la differente distanza tra gli impianti e i bar e i differenti mezzi di trasporto utilizzati, i costi di trasporto euro/chilogrammo di caffè da un impianto ad un bar risultano differenti e sono riassunti nella seguente tabella:

	B_1	B_2	B_3
T_1	$0,4$	$0,3$	$0,2$
T_2	$0,2$	$0,3$	$0,5$
T_3	$0,1$	$0,6$	$0,2$
T_4	$0,5$	$0,1$	$0,3$

Sapendo che:

- il costo fisso per l'apertura di ciascun impianto è di 1350 euro;
- non si vogliono attivare più di 3 impianti di torrefazione;
- gli impianti di torrefazione T_1, T_2, T_3, T_4 possono produrre giornalmente al massimo 75, 90, 80 e 65 Kg di caffè, rispettivamente;
- i tre bar necessitano di 60, 75 e 80 Kg di caffè, rispettivamente;

qual è la quantità da trasportare da ogni impianto a ogni bar per minimizzare **i costi totali**?

Lavoro di gruppo

1. Elaborare un modello di Programmazione Lineare Mista Intera per il problema.
2. Implementare il modello ideato tramite la libreria PuLP in Python e risolverlo, stampando:
    - il valore ottimo di tutte le variabili;
    - la produzione effettiva di caffè per ogni impianto;
    - il numero di impianti di torrefazione aperti.

Conclusione

Compiti per lunedì 14 novembre 2022

1. Finire di implementare e risolvere con Python+PuLP il problema *`Buongiornissimo! Kaffè? v2`* del secondo lavoro di gruppo.
2. Implementare e risolvere con Python+PuLP il problema *`Un treno da prendere`* (Problema del cammino minimo – ROAR II, Lezione 3 slide 40 e Lezione 4 slide 8).

Consegnare alla Prof.ssa Picchi gli script di Python sviluppati **entro giovedì 10 novembre**.

Sondaggio finale

www.menti.com – Codice: **2672 5790**