Progetto Ricerca Operativa 2024/2025

April 8, 2025

1 Descrizione

Siete stati incaricati dalla nota azienda avicola *Polli Tech di N.I. & Co.* di progettare una catena di distribuzione per la commercializzazione dei suoi prodotti. In particolare, l'azienda vi richiede di trovare il piazzamento ottimale (tra un insieme possibile di posizioni) di magazzini per le merci, i quali hanno lo scopo di distribuire i propri prodotti ai supermercati circostanti.

Ogni magazzino ha un costo di costruzione ed è in grado di servire un certo sottoinsieme di supermercati. Ogni supermercato non servito da nessun magazzino costituisce una perdita economica per l'azienda. Infine, il piazzamento dei magazzini deve tenere conto del costo di trasporto delle merci dall'azienda ai magazzini stessi, che viene effettuato con un unico mezzo che parte dall'azienda, visita ogni magazzino e ritorna all'azienda ogni giorno.

2 Dati

Ogni istanza del problema è composta dai seguenti files:

- `weights.json`: contiene i costi che l'azienda può sostenere, che sono:
  - `'construction'`: costo giornaliero dovuto alla costruzione e alla manutenzione di un magazzino (immaginiamo che la costruzione non sia pagata una tantum ma sia ammortizzata nel tempo)
  - `'missed_supermarket'`: penalità giornaliera per un supermercato non servito da nessun magazzino

  - `'travel'`: costo del carburante per ciascun chilometro di distanza percorso

- `service.csv`: matrice in cui ogni riga si riferisce a una possibile posizione dei magazzini e ogni colonna a un supermercato. Se un magazzino può servire un certo supermercato l'elemento della matrice corrispondente è pari a 1, 0 altrimenti.

- `distances.csv`: matrice delle distanze tra le possibili posizioni dei magazzini e tra le possibili posizioni dei magazzini e l'azienda. Sia sulle colonne che sulle righe, il primo elemento fa riferimento all'azienda, mentre gli altri ai magazzini, nello stesso ordine in cui si trovano in `service.csv`. Ciascun elemento della matrice (che non è necessariamente simmetrica) rappresenta la distanza in chilometri dal luogo sulla riga al luogo sulla colonna.

3 Richiesta

Utilizzando **python** come linguaggio di programmazione e **GUROBI** come solver, si sviluppi un modello di programmazione lineare per risolvere il problema.

4 Requisiti tecnici

Il progetto deve essere svolto in gruppi di 1-3 persone.

Il solver deve essere costituito da un solo file dal nome

`'solver_XXXXXX_YYYYYY_ZZZZZZ.py'`

dove `XXXXXX`, `YYYYYY` e `ZZZZZZ` sono i numeri di matricola dei componenti del gruppo. Per esempio, se un gruppo è formato da due persone le cui matricole sono 123456 e 654321, il file dovrà chiamarsi

`'solver_123456_654321.py'`,

mentre se la matricola 999999 decidesse di fare individualmente il progetto, il file dovrà chiamarsi

`'solver_999999.py'`.

Al suo interno il file dovrà contenere una classe che eredita la classe `AbstractSolver` e che si deve chiamare esattamente come il file che la contiene. Per esempio, le matricole 123456 e 654321 avranno un file che si chiama

`'solver_123456_654321.py'`

che contiene la classe

`class solver_123456_654321(AbstractSolver):.`

In questa classe è necessario implementare il metodo `solve()`, il quale non deve prendere nulla in input (i dati necessari sono presenti in `self.inst`) e deve ritornare come output, in ordine, il vettore `X` e la matrice `Y`.

Il vettore `X` è un vettore binario lungo quanto il numero totale di possibili posizioni dei magazzini e ogni elemento è pari a `1` (o `True`) se un magazzino è costruito in quella posizione, `0` (o `False`) altrimenti.

La matrice `Y` è una matrice binaria le cui dimensioni sono entrambe pari al numero di possibili posizioni dei magazzini più uno (quindi ha le stesse dimensioni della matrice delle distanze contenuta in `'distances.csv'`). La prima riga e colonna si riferiscono all'azienda, le rimandenti alle possibili posizioni per i magazzini. Il generico elemento `i,j` della matrice è uguale a `1` (o `True`) se il percorso del mezzo prevede la tratta da `i` a `j`, `0` (o `False`) altrimenti.

5 Testing

Per testare il solver necessario seguire i seguenti step:

- inserire il file all'interno della cartella `'solvers'`;
- aggiornare il file `'solvers/__init__.py'` importando il solver e aggiungendo il nome della classe dentro a `__all__`;
- sostituire il `DummySolver` nel `'main.py'` con il solver;
- lanciare il `'main.py'` per utilizzare il solver;
- lanciare il `'evaluator.py'` per vedere il costo totale della soluzione trovata dal solver.

6 Nota bene

I solver consegnati verranno testati per la valutazione esattamente come descritto nella sezione precedente, quindi con gli stessi identici mezzi che vi sono resi disponibili in fase di sviluppo. Per questo motivo, se il `'main.py'` e il `'evaluator.py'` dovessero produrre errore (a causa del nome del file, del nome della classe, del formato di output sbagliato, o per qualsiasi altro motivo) quando lanciati, al progetto verranno automaticamente assegnati 0 punti.

7 Deadline

Il progetto deve essere consegnato tramite la sezione **Elaborati** del Portale della didattica (uno solo per gruppo) entro le 23:59 del giorno 30/06/2025.