Courtesy Bus Problem

Giulia Calanca

A.A 2021/2022

Contents

1 Definizione del problema

Questo problema è una variazione del più noto CVRP (Capacitated Vehicle Routing Problem) con time windows. Il problema si può descrivere nel seguente modo: in un pub si hanno alcuni courtesy bus, ovvero dei veicoli che effettuano un servizio taxi per i suoi clienti. Essi devono appunto riportare a casa i clienti dopo la serata trascorsa al pub a partire dall'orario da loro

richiesto. Ogni cliente fornisce un lower bound dell'orario al quale vuole arrivare a casa e la soddisfazione del cliente dipenderà da quanto l'effettiva ora d'arrivo differirà da quella richiesta.

Gli **obiettivi** sono:

- trovare l'insieme di route che minimizzino il costo di percorrenza dei vari courtesy bus
- massimizzare la soddisfazione dei clienti portandoli a casa il prima possibile considerando l'orario richiesto

I **vincoli** da rispettare sono:

- non eccedere la capacità dei bus
- portare a casa tutti i clienti
- rispettare le time windows
- far partire e ritornare ogni route dal pub

2 Formalizzazione del problema

Sia:

- K l'insieme dei bus di capacità Q
- C l'insieme dei clienti del pub
- a_i l'orario di arrivo a casa desiderato, richiesto da ogni cliente $i \in C$
- $[a_i, +\infty]$ la time windows in cui portare a casa il cliente associato al nodo i
- $G = (V, A)$ un grafo orientato con \$V=\0\ $\cup$ C \$, dove il nodo $\{0\}$ rappresenta il pub e con A insieme degli archi (i, j)
- $t_{i,j}$ il tempo di attraversamento dell'arco $(i, j) \in A$
- $c_{i,j}$ il costo di attraversamento dell'arco $(i, j) \in A$

3 Modello

3.1 Variabili

Oltre alle variabili $t_{i,j}$ e $c_{i,j}$ che rappresentano rispettivamente il tempo ed il costo di attraversamento, definiamo una variabile tridimensionale $x_{i,j,k}$ per capire quali bus percorrono quali archi.

$$x_{i,j,k} = \begin{cases} 1 & \text{if bus } k \text{ travels from } i \text{ to } j \text{ directly} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Per implementare le time windows c'è bisogno di un modo per determinare quando un cliente viene riportato a casa. Introduciamo quindi due variabili temporali:

- z_i rappresenta l'istante in cui il cliente i arriva a casa
- $y_{i,k}$ che rappresenta l'istante nel quale il bus k arriva a casa del cliente i

In ultimo aggiungiamo la variabile $w_{i,k}$ che determina se il bus k porta a casa il cliente i:

$$w_{i,k} = \begin{cases} 1 & \text{if bus } k \text{ takes customer } i \text{ home} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

3.2 Funzione obiettivo

Considerando di dover minimizzare anche il tempo che impiego a riportare a casa il cliente la f.o. diventa:

$$\min \alpha \sum_{k \in K} \sum_{(i,j) \in A} c_{i,j} x_{i,j,k} + \beta \sum_{i \in C} z_i - a_i \tag{3}$$

Con α e β parametri per stabilire a quale delle componenti della f.o. dare più importanza.

3.3 Vincoli

1. Non eccedere la capacità dei bus

$$\sum_{(i,j) \in A(-,-,k)} x_{i,j,k} \leq Q;\ k \in K \tag{4}$$

2. I clienti vengono portati a casa ognuno una sola volta e da un solo bus

$$\sum_{k \in K} \sum_{i \in A(-,j,k)} x_{i,j,k} = 1;\ j \in C \tag{5}$$

3. Bilanciamento del flusso

$$\sum_{i \in A(-,h,k)} x_{i,h,k} - \sum_{j \in A(h,-,k)} x_{h,j,k} = 0;\ h \in C, k \in K \tag{6}$$

4. Vincoli viaggi bus: ogni bus parte dal pub e vi ritorna alla fine del giro. Ogni bus, se tra quelli selezionati, è utilizzato una sola volta.

   a. Il bus parte dal pub, nodo $\{0\}$.

$$\sum_{j \in A(0,-,k)} x_{0,j,k} <= 1;\ k \in K \tag{7}$$

   b. Il bus ritorna al pub, nodo $\{0\}$.

$$\sum_{i \in V} x_{i,0,k} <= 1;\ k \in K \tag{8}$$

5. Lower bound del tempo di arrivo desiderato

$$z_i \geq a_i, \text{ for } i \in C \tag{9}$$

6. Valorizzazione $y_{i,k}$: consecutività tempi di arrivo di un bus

   a. Lower bound

$$y_{j,k} \geq y_{i,k} + t_{i,j} x_{i,j,k} - M(1 - x_{i,j,k}) \tag{10}$$

   b. Upper bound

$$y_{j,k} \leq y_{i,k} + t_{i,j} x_{i,j,k} + M(1 - x_{i,j,k}) \tag{11}$$

7. Valorizzazione z_i

$$z_i = \sum_{k \in K} y'_{i,k}\ ;\ \ i \in I \tag{12}$$

8. Valorizzazione $w_{i,k}$

$$w_{i,k} = \sum_{j \in A(i,-,k)} x_{i,j,k};\ i \in C,\ k \in K \tag{13}$$

9. MW

$$Mw_{i,k} = M \cdot w_{i,k};\ i \in C,\ k \in K \tag{14}$$

10. Valorizzazione $y'_{i,k}$

$$y'_{i,k} = min(Mw_{i,k},\ y_{i,k}) \tag{15}$$

3.4 Note

- M, detta big-M, è una costante dal valore tendente a $+\infty$
- A è una matrice che rappresenta quali archi vengono percorsi da quale bus. È formata da 3 elementi (arco_i, arco_j, bus). La notazione $A(-, j, k)$ indica gli archi con arco i fisso e j, k non fissi.

4 Euristiche e metaeuristiche

4.1 Euristica costruttiva

Per generare una prima semplice soluzione accettabile sono stati testati tre differenti algoritmi greedy:

1. Il primo algoritmo riempie un bus alla volta aggiungendo ad ogni passo il cliente più vicino
2. Il secondo algoritmo scorre invece la lista dei bus e aggiunge un cliente alla volta scegliendo quello più vicino
3. L'ultimo algoritmo abbina in modo casuale ogni cliente ad un bus che abbia sufficiente capacità.

Tutti e tre gli algoritmi generano soluzioni simili in termini di costo, ma l'ultimo algoritmo genera una soluzione che risulta più adatta (con più possibilità di miglioramento) come soluzione iniziale da dare in input alla local

4.2 Local search

La local search si compone essenzialmente di 1 mossa:

- `MoveAndOptTime(node,bus,pos)`, una versione della *insert* che sposta il nodo `node` nella lista di clienti del bus `bus` nella posizione `pos`. `bus` può essere lo stesso di partenza o un altro.

All'interno della mossa è presente una sub-mossa che ottimizza il tempo di partenza del bus `bus`. Viene calcolato per ogni nodo il tempo minimo di partenza che rispetti le time windows e tra questi viene selezionato il massimo. Un'altra mossa presa in considerazione è stata la `2_Opt`, che prende due archi all'interno di un trip e li scambia. Questa mossa però è risultata meno efficiente della `MoveNode` in quanto crea cambiamenti troppo grandi all'interno dei percorsi, soprattutto in termini di rispetto delle time windows dei clienti.

4.3 Local search multi-start

Come ultimo passaggio di ottimizzazione per questa euristica si è aggiunta la possibilità di utilizzare la LS con un multistart. Il risolutore greedy costruisce ogni volta una soluzione diversa, che viene ottimizzata con la ls finchè non si supera il tempo limite stabilito.

4.4 Simulated annealing

Come metaeuristica si è implementata una versione dell'algoritmo di simulated annealing. I parametri utilizzati di default e che in generale hanno dato risultati migliori sono i seguenti:

- COOLING_RATE = 0.98
- INITIAL_TEMPERATURE = 10
- MINIMUM_TEMPERATURE = 1
- ITERATIONS_PER_TEMPERATURE = 10000

Ma rimane comunque la possibilità di parametrizzare la simulated annealing a piacimento in base al dataset del problema. L'algoritmo è strutturato nel seguente modo:

```
solution = costructive_solver.solve()

while temperature > min_temperature & (time < end_time):
  for i < n_of_iterations:
    new_solution = solution

    MoveAndOptTime(new_solution, random_src_node,
   random_dst_bus, random_dst_pos).apply()

    if new_solution is feasible do
    delta = old_cost - new_cost
      if random() <= exp(delta/temperature)
solution = new_solution

      if new_cost < best_cost:
best_solution = solution
    i++
  temperature = temperature * cooling_rate

return best_solution
```

5 Struttura del progetto

A livello di struttura il progetto si compone dei seguenti moduli:

- `model.py` che definisce la classe `Model`, dove sono contenuti i dati del problema
- `solution.py` che definisce la classe `Solution`, dove è contenuta la soluzione
- `validator.py` che definisce la classe `Validator`, la quale valida una soluzione in base modello corrispondente e calcola anche i costi sia delle route sia in termini di soddisfazione dei clienti
- `gurobysolver.py` che definisce il risulutore ottimo creato con Gurobi
- `heuristics.py` che contiene le varie classi che implementano le euristiche

- `commons.py` che contiene anche help functions utilizzate all'interno dei vari moduli

Il progetto infine contiene un `main` che effettua il parsing dei parametri passati in input e chiama i differenti risolutori.

6 Risultati

solver	# nodes	# buses	seconds to solve	solution
gurobi	3	2	10	47.2431
greedy	3	2	-	98.9890
ls	3	2	10	47.2431
ls-ms	3	2	10	47.2431
sa	3	2	10	47.2431
gurobi	4	2	10	56.2986
greedy	4	2	-	141.7909
ls	4	2	10	63.2672
ls-ms	4	2	10	56.2986
sa	4	2	10	56.2986
gurobi	7	3	10	116.6939
greedy	7	3	-	527.6650
ls	7	3	10	118.4272
ls-ms	7	3	10	116.6939
sa	7	3	10	116.6939
gurobi	9	3	10	202.4896
greedy	9	3	-	2167.4507
ls	9	3	10	261.5168
ls-ms	9	3	10	202.4896
sa	9	3	10	202.4896
gurobi	11	10	100	213.2659
greedy	11	10	-	826.6167
ls	11	10	100	227.3628
ls-ms	11	10	100	227.3628
sa	11	10	100	227.3628
gurobi	20	5	100	3238.1498
greedy	20	5	-	11480.8114
ls	20	5	100	3379.7927
ls-ms	20	5	100	2998.8899
sa	20	5	100	2998.8899
gurobi	41	8	200	772.44067
greedy	41	8	-	4474.2547
ls	41	8	200	715.7607
ls-ms	41	8	200	660.1733
sa	41	8	200	588.4076
gurobi	52	11	200	4610.3419
greedy	52	11	200	17768.6750
ls	52	11	200	3131.1859
ls-ms	52	11	200	2984.3199
sa	52	11	200	2889.8812
gurobi	200	13	200	-
greedy	200	13	200	77783.4134
ls	200	13	200	
ls-ms	200	13	200	
sa	200	13	200	5105.4852