

Courtesy Bus Problem

Giulia Calanca

A.A 2021/2022

Contents

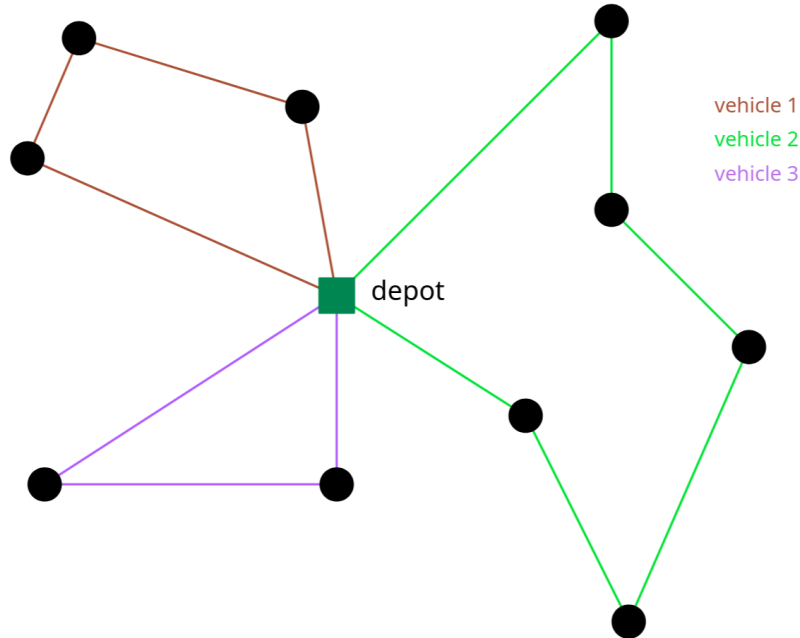
1	Il problema in letteratura	2
1.1	Metodi di soluzione esatti	3
1.2	Modellazione del problema	4
1.2.1	Modellazione CVRP	4
1.2.2	Modellazione CVRPTW	5
1.2.3	Modello esteso CVRPTW	6
1.3	Tecniche euristiche	7
2	Definizione del problema	7
2.1	Obiettivi	7
2.2	Vincoli	7
3	Formalizzazione del problema	8
4	Modello	8
4.1	Variabili	8
4.2	Funzione obiettivo	9
4.3	Vincoli	9
4.4	Note	10
5	Euristiche e metaeuristiche	10
5.1	Euristica costruttiva	10
5.2	Local search	11
5.3	Local search multi-start	11
5.4	Simulated annealing	11
6	Struttura del progetto	12

7	Istanze del problema e risultati	13
8	Bibliografia	15

1 Il problema in letteratura

Il problema di vehicle routing (VRP) è un classico problema di ottimizzazione combinatoria e programmazione intera. Introdotto da Dantzig and Ramser (1959) ¹ è una generalizzazione del problema del commesso viaggiatore (TSP, Travelling salesman problem). In VRP si richiede di trovare l'insieme ottimo di percorsi che un gruppo di veicoli deve percorrere affinché tutti i clienti vengano serviti. Esistono numerose varianti del VRP e altrettante implementazioni commerciali di casistiche specifiche. Trovare una soluzione ottima al problema è NP-hard, per questo motivo nelle implementazioni reali, dove dimensione e frequenza di applicazione non permettono di raggiungere l'ottimo, si ricorre solitamente all'uso di euristiche. In figura 1 è rappresentato un esempio di istanza di VRP: il quadrato verde al centro è il deposito da dove partono e ritornano i veicoli, mentre i cerchi neri sono i clienti da servire.

¹G. Dantzig, J. Ramser, The truck dispatching problem, Management Science, 6 (1959), pp. 80-91



Come già menzionato esistono parecchie varianti del problema VRP nelle quali cambiano, o si aggiungono, vincoli di capacità, tempo, punti di partenza, numero di viaggi, etc. La variante del problema implementata in questo progetto può essere considerata come una sotto-variante del CVRPTW (i.e. *Capacitated Vehicle Routing Problem with Time Windows*) nella quale i veicoli hanno una capacità limitata e sono presenti vincoli di tempo per la consegna.

1.1 Metodi di soluzione esatti

La principale tecnica impiegata per la risoluzione di VRP è la programmazione lineare intera (ILP). Esistono diverse modellazioni del problema - nelle sezioni successive ne vengono descritte due tra le più conosciute: la modellazione compatta, con un numero di variabili e vincoli polinomiale e la modellazione estesa, ovvero con variabili e vincoli in numero esponenziale.

1.2 Modellazione del problema

1.2.1 Modellazione CVRP

Una possibile formulazione del problema CVRP tramite ILP è la seguente:

$$\begin{aligned} \min \sum_{i,j \in V} c_{ij} \sum_{k \in K} x_{ijk} \\ \text{sub. to} \end{aligned}$$

$$\sum_{i \in V} \sum_{k \in K} x_{ijk} = 1 \quad \forall j \in V \setminus 0 \quad (1)$$

$$\sum_{j \in V} \sum_{k \in K} x_{ijk} = 1 \quad \forall j \in V \setminus 0 \quad (2)$$

$$\sum_{i \in V} \sum_{k \in K} x_{ihk} - \sum_{j \in V} \sum_{k \in K} x_{hjk} = 0 \quad \forall k \in K, h \in V \quad (3)$$

$$\sum_{i \in V} q_i \sum_{j \in V} x_{ijk} \leq Q_k \quad \forall k \in K \quad (4)$$

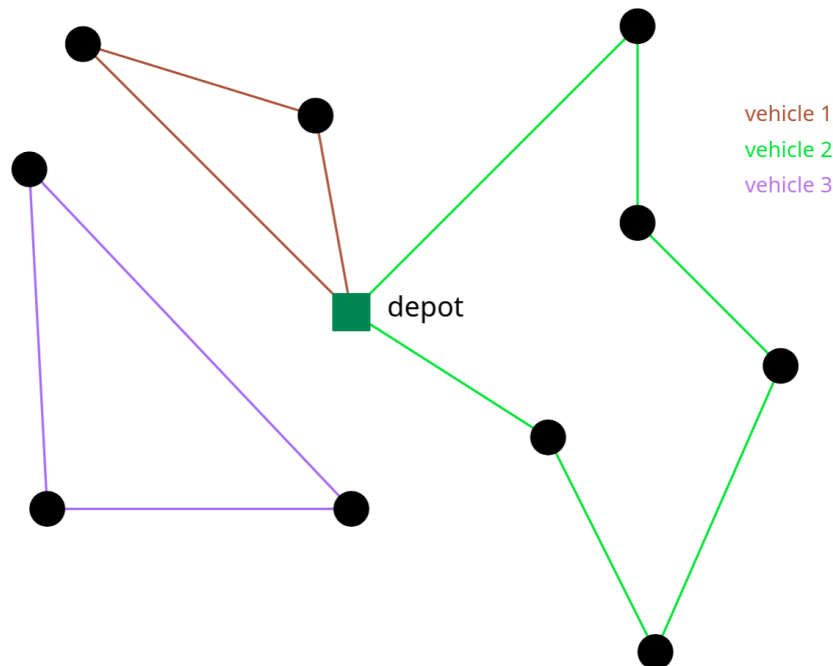
$$\sum_{ijk} x_{ijk} = |S| - 1 \quad \forall S \subseteq P(V), 0 \notin S \quad (5)$$

$$x_{ijk} \in 0, 1 \quad (6)$$

Con:

- V l'insieme dei nodi (clienti e depot), con depot = 0
- K l'insieme dei bus
- c_{ij} costo dell'attraversamento dell'arco (i, j)
- $x_{ijk} = 1$ se il bus k attraversa l'arco (i, j) , $x_{ijk} = 0$ altrimenti
- Q_k la capacità del bus $k \in K$

I vincoli (1) e (2) garantiscono che ci siano rispettivamente un solo veicolo in entrata ed uno solo in uscita per ogni nodo (i.e. cliente). Il vincolo (3) garantisce che il veicolo in entrata coincida con quello in uscita per un determinato nodo, mentre il (4) garantisce sia rispettato il vincolo di capacità dei veicoli. Il vincolo (5) invece riguarda l'eliminazione dei *subtour*, ovvero dei percorsi chiusi che pur rispettando i vincoli da (1) a (4) non generano una soluzione accettabile (esempio in figura 1.2.1)



Si indica con S il subtour e con $|S|$ il numero dei nodi che lo compongono. Per tutti i subtour nell'insieme dei path $P(V)$ che non contengono il depot (indicato con 0) il numero di archi deve coincidere con la cardinalità del subtour $|S|$ meno 1. Il numero dei subtour è esponenziale nel numero di nodi del problema, per questo motivo spesso si ricorre a tecniche "lazy", ovvero si agisce sui subtour solo nel caso siano presenti nella soluzione.

1.2.2 Modellazione CVRPTW

La modellazione del CVRPTW prevede gli stessi vincoli del CVRP con l'aggiunta dei vincoli sulle time windows e la conseguente esclusione di quelli che regolano i subtour. I vincoli temporali di fatto eliminano la formazione di subtour, rendendo dunque quei vincoli non più necessari. Sia:

- $[a_i, b_i]$ la finestra di tempo nella quale un veicolo deve visitare il i
- s_{ik} l'istante in cui inizia il servizio del veicolo k in i
- t_{ij} il tempo di attraversamento dell'arco (i, j)
- M , detta big-M, una costante dal valore tendente a ∞ .

Allora i vincoli addizionali sono:

$$a_i \leq s_{ik} \leq b_i \quad (7)$$

$$s_{ik} + t_{ij} - M(1 - x_{ijk}) \leq s_{jk} \quad (8)$$

Essi garantiscono il rispetto delle time windows (vincolo (7)) e la consecutività dei tempi di arrivo nei nodi (vincolo (8)). Tramite ILP è possibile dunque trovare la soluzione ottima del problema VRP e VRPTW, questo però solamente per problemi di piccola dimensione - per problemi più grandi il costo computazionale sarebbe troppo elevato.

1.2.3 Modello esteso CVRPTW

Un'altra formulazione del problema CVRPTW è possibile tramite la column generation. Si immagina una tabella nella quale ogni riga rappresenta un cliente/nodo, mentre ogni colonna rappresenta un cammino ammissibile.

	89	76	99	45	32	...
1	1	1	1	0	0	...
2	0	1	1	0	1	...
3	0	0	0	0	0	...
4	1	0	1	1	0	...
5	1	0	0	0	0	...

Nell'esempio la prima colonna rappresenta il percorso che attraversa i nodi 1, 4, 5 (non necessariamente in questo ordine) ed ha un costo di 89 unità, la seconda colonna passa per i nodi 1, 2 e costa 76, e così via per tutti i possibili percorsi. Il problema allora diventa trovare i cammini che, attraversando tutti i nodi, minimizzano il costo totale della route. Sia K l'insieme di tutte le colonne (ovvero i possibili path), $a_{ik} = 1$ se il percorso definito dalla colonna k serve il cliente i e c_k il costo di quel percorso, allora il problema si descrive in questo modo:

$$\begin{aligned}
& \min \sum_{k \in K} c_k x_k \\
& \text{sub. to} \\
& \sum_{k \in K} a_{ik} x_k = 1 \quad \forall i \\
& x_k \in 0, 1
\end{aligned}$$

La formulazione del VRP così espressa diventa un problema di *set partitioning* ed è di per sè semplice. La difficoltà sta nel fatto che le variabili sono in

numero esponenziale e nel trovare i cammini ammissibili per i quali devono essere rispettati i vincoli temporali e di capacità. Nel sottoproblema dunque si richiede di risolvere il problema noto come *elementary shortest path with time windows and capacity constraints* (ESPPTCC) nel quale si cerca il cammino minimo che tenga conto delle finestre temporali, delle capacità e che sia elementare, ovvero visiti ogni vertice una sola volta.

1.3 Tecniche euristiche

2 Definizione del problema

Questo problema è una variante 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.

2.1 Obiettivi

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

2.2 Vincoli

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

3 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 $(i, j) \in A$

4 Modello

4.1 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.

$$x_{i,j,k} = \begin{cases} 1 & \text{if bus } k \text{ travels from } i \text{ to } j \text{ directly} \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

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 :

$$w_{i,k} = \begin{cases} 1 & \text{if bus } k \text{ takes customer } i \text{ home} \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

4.2 Funzione obiettivo

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

$$\min \alpha \sum_{k \in K} \sum_{(i,j) \in A} c_{i,j} x_{i,j,k} + \beta \sum_{i \in C} z_i - a_i \quad (11)$$

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

4.3 Vincoli

1. Non eccedere la capacità dei bus

$$\sum_{(i,j) \in A(-,-,k)} x_{i,j,k} \leq Q; k \in K \quad (12)$$

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

$$\sum_{k \in K} \sum_{i \in A(-,j,k)} x_{i,j,k} = 1; j \in C \quad (13)$$

3. Bilanciamento del flusso

$$\sum_{i \in A(-,h,k)} x_{i,h,k} - \sum_{j \in A(h,-,k)} x_{h,j,k} = 0; h \in C, k \in K \quad (14)$$

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 $\{0\}$.

$$\sum_{j \in A(0,-,k)} x_{0,j,k} \leq 1; k \in K \quad (15)$$

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

$$\sum_{i \in V} x_{i,0,k} \leq 1; k \in K \quad (16)$$

5. Lower bound del tempo di arrivo desiderato

$$z_i \geq a_i, \text{ for } i \in C \quad (17)$$

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

a. Lower bound

$$y_{j,k} \geq y_{i,k} + t_{i,j}x_{i,j,k} - M(1 - x_{i,j,k}) \quad (18)$$

b. Upper bound

$$y_{j,k} \leq y_{i,k} + t_{i,j}x_{i,j,k} + M(1 - x_{i,j,k}) \quad (19)$$

7. Valorizzazione z_i

$$z_i = \sum_{k \in K} y'_{i,k}; i \in I \quad (20)$$

8. Valorizzazione $w_{i,k}$

$$w_{i,k} = \sum_{j \in A(i,-,k)} x_{i,j,k}; i \in C, k \in K \quad (21)$$

9. MW

$$Mw_{i,k} = M \cdot w_{i,k}; i \in C, k \in K \quad (22)$$

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

$$y'_{i,k} = \min(Mw_{i,k}, y_{i,k}) \quad (23)$$

4.4 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.

5 Euristiche e metaeuristiche

5.1 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

5.2 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.

5.3 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.

5.4 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 = constructive_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
```

6 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 risolutore 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.

7 Istanze del problema e risultati

Di seguito viene riportata la tabella che raccoglie i risultati ottenuti tramite le ottimizzazioni implementate nel modello.

solver	# nodes	# buses	seconds to solve	solution
gurobi	3	2	10	47.2431*
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*
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*
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*
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*
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
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
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
ls	52	11	200	3131.1859
ls-ms	52	11	200	2984.3199
sa	52	11	200	2889.8812
gurobi	200	13	200	-
ls	200	13	200	59069.6611
ls-ms	200	13	200	57914.4437
sa	200	13	200	4914.3869

Nota: le soluzioni indicate dall'asterisco rappresentano soluzioni ottime.

8 Bibliografia

- Simulated annealing
- VRPTW letteratura
- Esempio di callback gurobi