UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

Corso di Laurea Magistrale in Ingegneria dell'Automazione e Robotica

Dipartimento di
Ingegneria Elettrica e delle Tecnologie dell'Informazione

Elaborato finale di
MODELLI E METODI DELLA RICERCA OPERATIVA

TRACCIA 1:
THE PARALLEL DRONE SCHEDULING TSP

Docente:
Prof. Claudio Sterle

Studenti:
Emmanuel Patellaro P38/0239
Carmine Maisto P38/0242
Dario Barbato P38/0243

ANNO ACCADEMICO 2023/2024

Indice

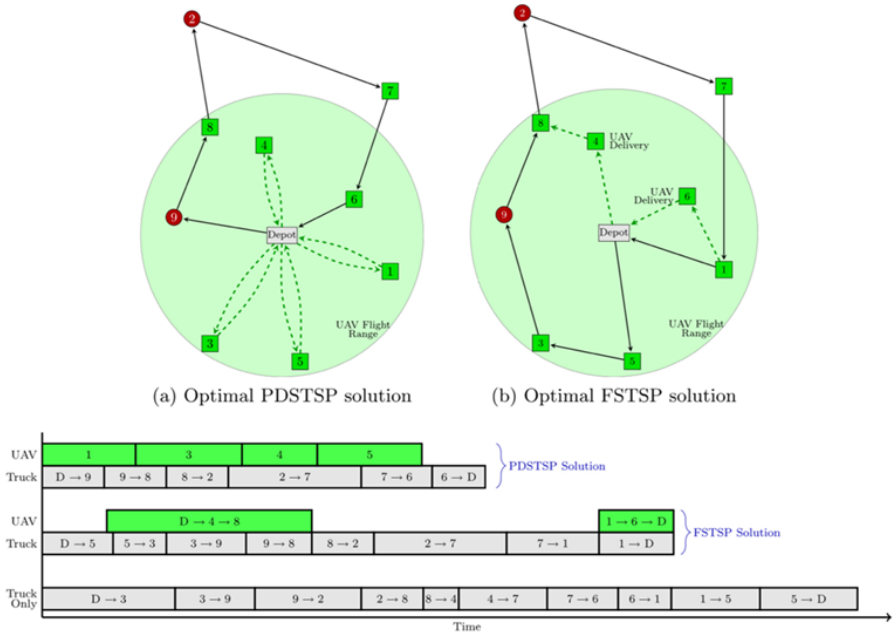
Capitolo 1

Introduzione

I veicoli aerei senza equipaggio, in precedenza, erano utilizzati solo ed esclusivamente in ambito militare. Nell'ultimo periodo, gli UAV (Unmanned Aerial Vehicles), comunemente noti come droni, stanno guadagnando popolarità in ambito commerciale, specialmente per quanto riguarda le operazioni di logistica, quali la consegna di pacchi a breve distanza. Mentre sono in corso significativi sforzi di ricerca per migliorare la tecnologia necessaria per consentire la consegna tramite drone, meno attenzione è stata dedicata alle sfide operative associate all'utilizzo di questa tecnologia.

L'articolo da noi esaminato, dal titolo "The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery" di Chase C. Murray e Amanda G. Chu, propone due modelli di programmazione matematica mirati al routing e alla pianificazione ottimali di UAV e camion per la consegna di pacchi. In tali modelli l'UAV lavora in collaborazione con un tradizionale camion per la distribuzione dei pacchi. Le soluzioni a questi problemi permettono che un tale sistema di consegna fornirà una ricezione più rapida degli ordini dei clienti a minor costo per il distributore e con impatti ambientali ridotti.

La principale contribuzione di questo documento è introdurre una nuova variante del tradizionale problema del commesso viaggiatore (TSP) che affronta la sfida di determinare gli assegnamenti ottimali dei clienti per un UAV che lavora in parallelo con un camion di consegna. Il problema da noi esaminato riguarda la definizione degli assegnamenti ottimali di camion e UAV nel caso di un centro di distribuzione situato nelle vicinanze dei clienti. Questo problema di pianificazione parallela del drone prende il nome di PDSTSP (PARALLEL DRONE SCHEDULING TSP) il quale, a differenza del FSTSP, consiste nel consegnare i pacchi ai clienti da parte di vari UAV e di un camion, indipendentemente gli uni dagli altri. Gli UAV infatti, alla fine di una consegna, rientrano al deposito per poi essere pronti ad effettuare un'altra eventuale consegna ad un cliente diverso.



(a) Optimal PDSTSP solution (b) Optimal FSTSP solution

Figura 1.1:

In Figura 1.1 possiamo notare un semplice esempio di PDSTSP. L'FSTSP è applicabile in scenari in cui il centro di distribuzione è relativamente lontano dalle posizioni dei clienti e un singolo UAV è disponibile per operare in sincronia con un camion di consegna. Tuttavia, se una significativa porzione di clienti si trova entro la portata di volo di un UAV dal centro di distribuzione, sorge un problema diverso, quello del PDSTSP. Esiste un singolo deposito, da cui deve partire e ritornare un singolo camion di consegna e una flotta di uno o più UAV identici (dati dall'insieme V). Il camion serve i clienti lungo un percorso TSP, mentre gli UAV servono i clienti direttamente dal centro di distribuzione. Il PDSTSP, dunque, è un amalgama di due problemi classici della ricerca operativa. In primo luogo, esiste un TSP per sequenziare, come detto, quei clienti assegnati al camion di consegna. In secondo luogo, c'è il problema della pianificazione dei restanti clienti alla flotta di UAV, il quale è equivalente al problema della pianificazione parallela di macchine identiche con l'obiettivo di minimizzare il makespan. I due problemi, entrambi NP-hard, sono collegati dalla necessità di selezionare la suddivisione dei clienti da servire sia dal camion che da uno o più UAV.

Capitolo 2

Modello Matematico

2.1 Notazione e Formulazione Matematica

Indicando con $C = \{1, 2, ..., c\}$ l'insieme di tutti i clienti da servire, è possibile individuare un insieme $C^{'} \subseteq C$ contenente tutti i clienti che possono ricevere la consegna del loro pacco tramite UAV (cioè, il peso del pacco non supera la capacità di carico dell'UAV, non è richiesta la firma del cliente e la posizione del cliente è accessibile dall'UAV). Inoltre, consideriamo $C^{''} \subseteq C'$ l'insieme dei clienti che sono anche entro la portata dell'UAV dal centro di distribuzione (cioè, il cliente $i \in C^{'}$ è nell'insieme $C^{''}$ se $\tau^{'}_{0,i} + \tau^{'}_{i,c+1} \leq e$ dove $\tau^{'}_{0,i}$ e $\tau^{'}_{i,c+1}$ sono rispettivamente i tempi di andata e ritorno dell'UAV dal deposito al cliente ed e è una costante di tempo che nel nostro problema è pari a 30 minuti (0.5 h)).

L'insieme degli UAV è indicato dall'insieme $V = \{1, ..., d\}$, dove d è il numero di UAV. Sebbene esista una singola posizione fisica del deposito, per convenzione la assegniamo a due numeri di nodo unici, in modo che i veicoli partano dal deposito (nodo 0) e ritornino al deposito (nodo c+1). Pertanto, $N = \{0, 1, \ldots, c + 1\}$ rappresenta l'insieme di tutti i nodi nella rete. Per facilitare ulteriormente la struttura della rete del problema, consideriamo $N_0 = \{0, 1, \ldots, c\}$ l'insieme dei nodi da cui il camion può partire, e consideriamo $N_1 = \{1, 2, \ldots, c + 1\}$ l'insieme dei nodi che un veicolo (camion o UAV) può visitare durante il corso di un tour. Il tempo richiesto al conducente del camion per viaggiare dal nodo $i \in N_0$ al nodo $j \in N_1$ è dato da $\tau_{i,j}$. Invece il tempo richiesto all'UAV per viaggiare dal nodo 0 al nodo $i \in C^{''}$ è dato da $\tau^{'}_{0,i}$, mentre il tempo richiesto per ritornare al deposito c+1 dal nodo $i \in C^{''}$ è dato da $\tau^{'}_{i,c+1}$. Ai fini pratici e realistici è stata creata da noi un'equazione che permette di considerare i tempi di andata e ritorno, di un drone, differenti in quanto, al suo ritorno al deposito, questo è privo del pacco al suo interno. Tale equazione è la seguente:

$$\tau^{'}_{i,c+1} = \tau^{'}_{0,i} - (\tfrac{1}{4}\tau^{'}_{0,i} \cdot \ \tfrac{P}{3})$$

dove:

- $\tau'_{0,i}$ = tempo di andata del drone dal deposito al cliente i;

- $\tau'_{i,c+1}$ = tempo di ritorno del drone dal cliente i al deposito;

- P = peso del pacco trasportato.

2.2 Variabili Decisionali

Una formulazione di programmazione lineare mista del PDSTSP può essere costruita con l'introduzione di due variabili decisionali binarie:

- $x_{i,j}$, che è pari a 1 se il camion si sposta dal nodo $i \in N_0$ al nodo $j \in \{N_1 : j \neq i\}$

- $y_{i,v}$, che è pari a 1 se il cliente $i \in C''$ è servito dall'UAV $v \in V$

Il modello utilizza anche la variabile decisionale ausiliaria intera $1 \leq u_i \leq c + 2$ che mi permette di eliminare i possibili sottogiri. u_i non è altro che l'ordine di visita del vertice i nella soluzione. Tale meccanismo mi permette di avere una formulazione con un numero polinomiale di vincoli n^2, ma un eventuale rilassamento continuo del modello fornisce un LB molto più basso del LB fornito da altre formulazioni. Tali vincoli sono poco stringenti e dunque, facendo il B&B, l'albero che potrebbe uscirne fuori potrebbe essere molto grande e non è detto che sia possibile arrivare a soluzione.

2.3 Funzione Obiettivo e Vincoli

L'obiettivo è minimizzare l'ultimo momento in cui un veicolo ritorna al deposito, in modo che ogni cliente venga servito esattamente una volta. Tale problema, dunque, è un problema di MIN-MAX. A differenza di un classico problema di MIN-SUM (nel quale minimizzo una sommatoria), il problema di MIN-MAX non è un problema lineare. Non essendo lineare, non è possibile utilizzare il B&B o il simplesso e GUROBI non saprebbe gestirlo. Per permettere, dunque, tale risoluzione è necessario introdurre una nuova variabile continua alla quale vengono imposti dei vincoli, in questo caso (essendo di MIN-MAX e non di MAX-MIN) di $\geq$. Tale problema è preferito nei sistemi sociali, a differenza del problema del MIN-SUM che è preferito nei sistemi industriali.

Figura 2.1:

Nella Figura 2.1 possiamo individuare a destra una classica soluzione di un problema di MIN-MAX mentre a sinistra una soluzione di un problema di MIN-SUM.

La funzione obiettivo, sulla base di quanto detto, sarà la seguente:

$$\text{Min z} \quad (1)$$

dove i seguenti vincoli:

$$z \geq \sum_{i \in N_0} \sum_{\substack{j \in N_1 \\ j \neq i}} \tau_{i,j} x_{i,j} \quad (2)$$

$$z \geq \sum_{i \in C''} (\tau'_{0,i} + \tau'_{i,c+1}) y_{i,v} \quad \forall v \in V \quad (3)$$

forniscono i limiti inferiori su z, basati rispettivamente sulle assegnazioni del camion e degli UAV. Gli altri vincoli sono stati costruiti sulla base di alcune considerazioni.

Per garantire che ogni cliente sia visitato esattamente una volta, sia dal camion che dagli UAV si ha il seguente vincolo:

$$\sum_{\substack{i \in N_0 \\ i \neq j}} x_{i,j} + \sum_{\substack{v \in V \\ j \in C''}} y_{j,v} = 1 \quad \forall j \in C \quad (4)$$

Per assicurare che il camion lasci il deposito esattamente una volta (5) e che ci ritorni (6), risulta:

$$\sum_{j \in N_1} x_{0,j} = 1 \quad (5)$$

$$\sum_{i \in N_0} x_{i,c+1} = 1 \quad (6)$$

Seguono ulteriori vincoli di instradamento:

$$\sum_{\substack{i \in N_0 \\ i \neq j}} x_{i,j} = \sum_{\substack{k \in N_1 \\ k \neq j}} x_{j,k} \quad \forall j \in C \quad (7)$$

$$u_i - u_j + 1 \leq (c+2)(1 - x_{i,j}) \quad \forall i \in C, j \in \{N_1 : j \neq i\} \quad (8)$$

dove, il vincolo (7) specifica che un camion che entra in un nodo cliente deve anche lasciare tale nodo e il vincolo (8) è un vincolo standard di eliminazione dei sotto-circuiti (come già detto per la definizione della variabile decisionale ausiliaria u).

Infine, i vincoli (9), (10) e (11) specificano le definizioni delle variabili decisionali:

$$1 \leq u_i \leq c+2 \quad \forall i \in N_1 \quad (9)$$

$$x_{i,j} \in \{0,1\} \quad \forall i \in N_0, j \in \{N_1 : j \neq i\} \quad (10)$$

$$y_{i,v} \in \{0,1\} \quad \forall i \in C'', v \in V \quad (11)$$

2.4 Parametri di Input

I parametri di input del problema preso in esame sono letti da file di tipo .txt. Un esempio può essere il seguente:

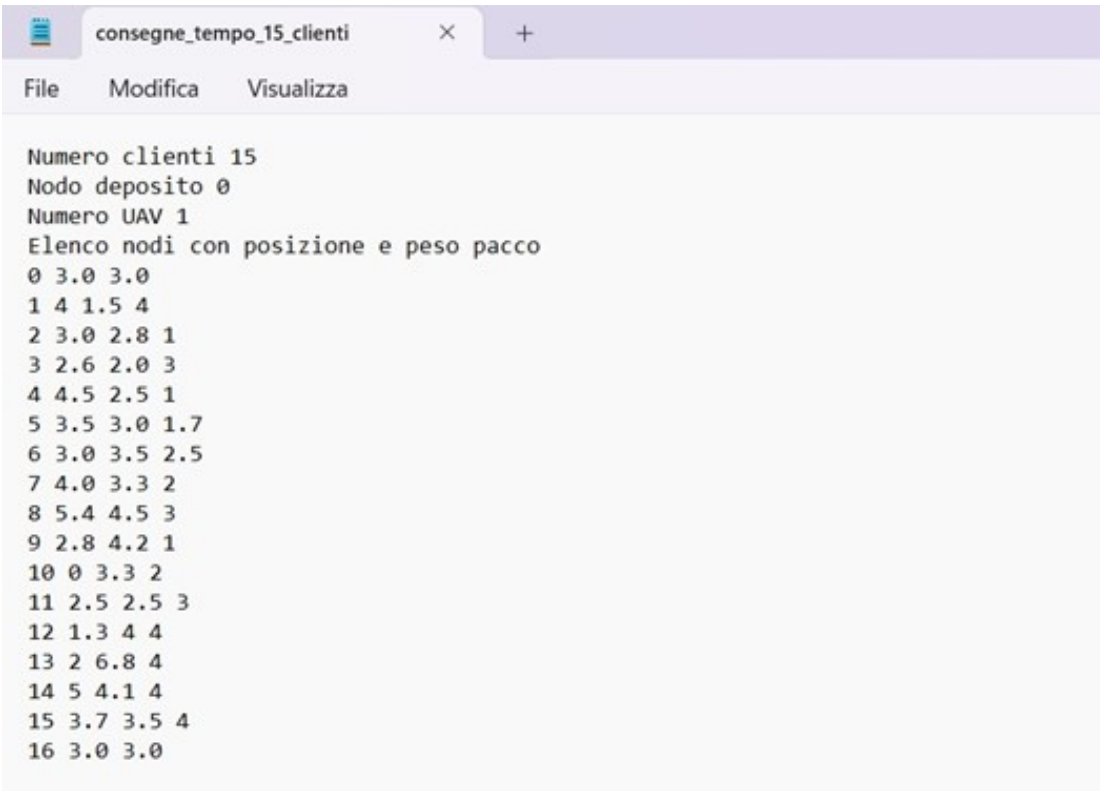


Figura 2.2:

Come si nota in Figura 2.2, all'interno del file sono inseriti i parametri utili per la risoluzione del problema.
Abbiamo:

- Numero di clienti;
- Nodo di partenza;
- Numero di UAV utilizzati per la consegna;

- Elenco dei nodi con rispettive posizioni (coordinate cartesiane utilizzate, in seguito, per il calcolo della distanza euclidea tra i vari nodi). Per i nodi clienti è indicato anche il peso del pacco da consegnare. Il primo e l'ultimo nodo rappresentano il nodo deposito (andata e ritorno).

All'interno del programma sono state inserite anche le velocità del camion e degli UAV. Nell'esempio descritto la velocità del camion corrisponde a 40 Km/h, mentre quella del drone a 50 Km/h. Tali valori sono utilizzati per il calcolo dei tempi $\tau_{i,j}, \tau_{0,i}^{'}$ e $\tau_{i,c+1}^{'}$ sulla base della definizione delle distanze euclidee tra i vari punti ($d_{i,j} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$), utilizzando la formula $\tau = \frac{d}{v}$.

Capitolo 3

Risoluzione Tramite Gurobi-Python

Il modello descritto nei capitoli precedenti è di seguito implementato e risolto attraverso l'utilizzo del software di ottimizzazione Gurobi, in particolare attraverso l'interfaccia Python fornita da Gurobi Optimization, LLC che consente di utilizzare Gurobi attraverso il linguaggio di programmazione Python, rendendo in questo modo più accessibile l'implementazione di soluzioni avanzate. Tale interfaccia prende, appunto, il nome di Gurobi-Python.

3.1 Codice

The parallel drone scheduling TSP

Importazione dei moduli in Python e inizializzazione del modello Gurobi

```
import sys
import math
import random
from itertools import permutations
import networkx as nx
import matplotlib.pyplot as plt
import gurobipy as gp
from gurobipy import GRB

# Inizializza il modello
```

```
m = gp.Model('The parallel drone scheduling TSP')

# Definizione della funzione subtour
def subtour(vals):
    cycle = []  # Inizializza una lista per contenere
    #il ciclo
    nodes = set(i for i, j in vals.keys() if vals[i, j] > 0.5)  #
        Ottiene l'insieme dei nodi con valore 1
    #nella soluzione
    visited_nodes = set()  # Inizializza un insieme per
    #tenere traccia dei nodi visitati

    while nodes:# Continua finche' ci sono nodi non visitati
        current_node = nodes.pop() # Estrae un nodo da quelli non
            visitati
        visited_nodes.add(current_node)  # Aggiunge il nodo a
            quelli visitati
        cycle.append(current_node)  # Aggiunge il nodo al ciclo
            corrente

        while True:
            next_node_candidates = [j for i, j in vals.keys() if i =
                = current_node and vals[i, j] > 0.5]  # Ottiene i
                nodi successivi

            if not next_node_candidates or next_node_candidates[0]
                in visited_nodes:  # Se non ci sono nodi successivi
                o sono gia' visitati, interrompi il ciclo
                break

            next_node = next_node_candidates[0]  # Estrae il nodo
                successivo
            if next_node in nodes:
                nodes.remove(next_node)  # Rimuove il nodo
                    successivo dagli "unvisited"

```

```
37                visited_nodes.add(next_node)  # Aggiunge il nodo
                      successivo a quelli visitati
38                cycle.append(next_node)  # Aggiunge il nodo successivo
                      al ciclo corrente
39                current_node = next_node  # Imposta il nodo corrente al
                      successivo
40
41        return cycle  # Restituisce il ciclo
```

Lettura da file dei dati di input

```
1  # LETTURA DA FILE PER NUMERO DI CLIENTI E DRONI
2
3  file = open("consegne_tempo_32_clienti.txt", "r")  # Apro il file
       di input per leggere tutti i dati di interesse
4
5  lines = file.readlines()  # Leggo le righe del file
6
7  c = int(lines[0].strip("\n").split("-")[2])  # Leggo dal file il
       numero di clienti ai quali bisogna consegnare un pacco
8  d = int(lines[2].strip("\n").split("-")[2])  # Leggo dal file il
       numero di UAV da utilizzare per la consegna in aggiunta al truck
9
10 N = range(0, c+2)  # Range di valori per indicare tutti i nodi
       nella rete
11 N0 = range(0, c+1)  # Range di valori per indicare i nodi da cui un
       veicolo puo' partire
12 N1 = range(1, c+2)  # Range di valori per indicare i nodi che un
       veicolo puo' visitare durante il corso di un tour.
13 C = range(1, c+1)  # Range di valori per indicare l'insieme di
       tutti i clienti
14 V = range(d)  # Range di valori per indicare l'insieme degli UAV
15
16 PosClienti = {}  # Posizioni di tutti i clienti
17 PosNodi = {}  # Posizione di tutti i nodi della rete (compresi i
       due nodi di andata e ritorno che indicano il deposito)
18
```

```
# Creo un dizionario che ha come chiave il nodo i-esimo e come valore la coppia di coordinate cartesiane del nodo
for k in range(0, c+2):
    row = lines[4+k].strip("\n").split("-")
    PosNodi[int(row[0])] = (float(row[1]), float(row[2]))

P = []  # Vettore Pesi Pacchi
CL = []  # Vettore Clienti (utilizzato per il calcolo LB)

# Come prima creo un dizionario ma solo con i nodi clienti. Inoltre creo due vettori, uno contenente il peso dei pacchi da
# consegnare ai clienti e uno contenente i clienti stessi (ai fini del LB non posso usare C definito come range)
for k in range(c):
    row = lines[5+k].strip("\n").split("-")
    PosClienti[int(row[0])] = (float(row[1]), float(row[2]))
    P.append(float(row[3]))
    CL.append(int(row[0]))

print("Posizione dei clienti ai quali effettuare la consegna:", '\n', PosClienti, '\n')
print("Posizione di tutti i nodi della rete:", '\n', PosNodi, '\n')

C_P = []  # C' e' un sottoinsieme di C e rappresenta l'insieme di quei clienti che possono ricevere la consegna
# del loro pacco tramite UAV in quanto il peso del pacco da consegnare non supera la capacita' massima di carico dell'UAV
# Creo C' leggendo da file i clienti che soddisfano la condizione sopra scritta

for k in range(c):
    row = lines[5+k].strip("\n").split("-")
    if float(row[3]) <= 5.0:
        C_P.append(int(row[0]))

print("Clienti appartenenti all'insieme C':", '\n', C_P, '\n')

```

```
49 file.close()  # Chiusura del file
```

Definizione delle variabili decisionali e dei costi di ogni arco

```
1  # Calcolo della distanza euclidea tra i vari nodi della rete
2  dist = {(i, j):
3          math.sqrt(sum((PosNodi[i][k] - PosNodi[j][k])**2 for k in
               range(2)))
4          for i in range(0, c+2) for j in range(0, c+2) if i != j}
5
6  print("Distanze Euclidee tra i vari nodi della rete:", '\n', dist,
       '\n')
7
8  # Velocita' in km/h
9  vel_truck = 40
10 vel_UAV = 50
11
12 # Calcolo dei tempi tau del truck
13 tau = {(i, j):
14        round(dist[i, j] / vel_truck, 3) for i, j in dist.keys()}
15 # Utilizzo 'round(-,3)' per approssimare il valore ottenuto alle
       prime 3 cifre decimali
16
17 print("Tempi di consegna da un punto ad un altro del truck:", '\n',
       tau, '\n')
18
19 # tau'(0,i)+tau'(i,c+1) <= e
20 # e costante di tempo. Nel nostro esempio essa e' pari a 0.5,
       ovvero 30 minuti
21 # tau'(0,i) = tempo che l'UAV impiega per consegnare il pacco dal
       deposito al cliente i
22 # tau'(i,c+1) = tempo che l'UAV impiega per tornare al deposito una
       volta consegnato il pacco al cliente i
23 tau1 = {}
24 C_S = []  # C'' e' un sottoinsieme di C'. Esso denota quei clienti,
       idonei alla consegna tramite UAV,
```

```
# che sono entro la portata dell'UAV dal centro di distribuzione (deve essere rispettata la relazione sopra scritta)

# Relazione tra tempo di andata e tempo di ritorno di un UAV che va da un cliente che appartiene all'insieme C''
# T_R=T_A-T_A*(1-3/4)*(P/3)
# P=Peso del Pacco, T_A=Tempo di Andata, T_R=Tempo di Ritorno

# Valuto tra i clienti appartenenti all'insieme C' quali rispettano la condizione di tempo di #consegna del drone e creo
# un vettore C_S che contiene questi ultimi
for i in C_P:
    tau1[0, i] = round(dist[0, i] / vel_UAV, 3)
    T_A = tau1[0, i]
    tau1[i, c+1] = round(T_A - T_A * (1 - 3/4) * P[i-1] / 3, 3)
    T_R = tau1[i, c+1]
    if T_A + T_R <= 0.5:
        C_S.append(i)

print("Clienti appartenenti all'insieme C'':", '\n', C_S, '\n')

# Definisco la variabile x[i,j] che e' pari a 1 se il truck si sposta dal nodo i appartenente a N0 al nodo j appartenente all'insieme {N+:j!=i}
x = m.addVars(tau.keys(), obj=tau, vtype=GRB.BINARY, name='x')

# Definisco la variabile y[i,v] che e' pari a 1 se il cliente i appartiene a C'' e' servito dall'UAV v che appartiene a V.
y = m.addVars(C_S, d, obj=tau1, vtype=GRB.BINARY, name='y')

# Definisco la variabile decisionale ausiliaria u che mi permette di eliminare i possibili sottogiri
# u[i] non e' altro che l'ordine di visita del vertice i nella soluzione
u = {}
for i in N:
```

```
53      u[i] = m.addVar(obj=0, vtype=GRB.INTEGER, lb=0, ub=c+2,
            name='u'+str(i))  # La variabile u deve essere compresa tra
            1 e c+2
54      if i == 0:
55          m.addConstr(u[i] == 0)  # Il nodo di partenza deve avere un
                valore di u associato pari a 0 essendo il primo nodo
                visitato
56
57  #print("Variabile x:", '\n', x, '\n')
58  #print("Variabile y:", '\n', y, '\n')
59  #print("Variabile u:", '\n', u, '\n')
```

Definizione della funzione obiettivo e dei vincoli

Funzione Obiettivo

```
1  # Variabile che andro' a mettere nella funzione obiettivo per
       linearizzare il problema di min-#MAX che e' NON LINEARE
2  # Essendo un problema di min-MAX avro' vari vincoli di >= su questa
       variabile
3  z = m.addVar(vtype=GRB.CONTINUOUS, name='z')
4
5  # Definizione della funzione obiettivo (problema di MINIMIZZAZIONE)
6  # La funzione obiettivo Cerca di minimizzare l'orario di ritorno
       piu tardivo al deposito sia #per l'UAV che per il truck
7  obj = m.setObjective(z, GRB.MINIMIZE)
```

Vincoli

```
1  # Definizione dei vincoli
2
3  # Limite inferiore su z basato sull'assegnazione del truck
4  m.addConstr(z >= gp.quicksum(tau[i, j] * x[i, j] for i in N0 for j
       in N1 if j != i))
5
6  # Limite inferiore su z basato sull'assegnazione dell'UAV
7  for v in V:
```

```
    m.addConstr(z >= gp.quicksum((tau1[0, i] + tau1[i, c+1]) * y[i,
        v] for i in C_S))

# Ogni cliente e' visitato una sola volta sia dal truck (prima
    sommatoria) che dall'UAV (seconda sommatoria)
for j in C:
    m.addConstr((gp.quicksum(x[i, j] for i in N0 if i != j) +
        gp.quicksum(y[j, v] for v in V if j in C_S)) == 1)

# Il truck deve lasciare il deposito esattamente una volta
m.addConstr(gp.quicksum(x[0, j] for j in N1) == 1)

# Il truck deve ritornare al deposito
m.addConstr(gp.quicksum(x[i, c+1] for i in N0) == 1)

# Quando il truck entra in un nodo cliente deve anche lasciare quel
    nodo
for j in C:
    m.addConstr(gp.quicksum(x[i, j] for i in N0 if i != j) ==
        gp.quicksum(x[j, k] for k in N1 if k != j))

# Vincolo standard di eliminazione dei sottocircuiti
m.addConstrs(u[i] - u[j] + 1 <= (c+2) * (1 - x[i, j]) for i in N
    for j in N1 if j != i)
```

Ottimizzazione

```
# Ottimizzazione del modello
m.optimize()

m._vars = x
vals = m.getAttr('X', x)
tour = subtour(vals)

# Stampa il percorso ottimale e il costo associato
print('')
print('Tour Ottimo del Truck: %s' % str(tour))
```

```
print('Costo Ottimo: %g' % m.ObjVal)
print('')

# Stampa dei clienti raggiunti dagli UAV
for i,v in y.keys():
    if y[i,v].x > 0.5:
        print("L'UAV",str(v+1),"ha raggiunto il cliente",str(i))
```

Visualizzazione grafica della posizione dei clienti

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.DiGraph()

# Aggiunta dei nodi
G.add_nodes_from(N)

# Aggiunta degli archi
G.add_edges_from(tau)
plt.figure(figsize=(25,20))
nx.draw(G, PosNodi, with_labels=True, node_size=2000)
# nx.draw_networkx_edge_labels(G,PosNodi, edge_labels=tau,
    label_pos=0.75)
plt.draw()
```

Visualizzazione grafica della posizione dei clienti visitabili dall'UAV per il vincolo di peso

```
import networkx as nx
import matplotlib.pyplot as plt

G1 = nx.DiGraph()

# Aggiunta dei nodi
```

```
G1.add_nodes_from(N)

# Aggiunta degli archi
G1.add_edges_from(tau1)
plt.figure(figsize=(15,10))
nx.draw(G1, PosNodi, with_labels=True, node_size=700)
# nx.draw_networkx_edge_labels(G1,PosNodi, edge_labels=tau1,
    label_pos=0.75)
plt.draw()
```

Percorso ottimo del Truck

```
plt.figure(figsize=(18,15))
def plot_tsp_solution(tour, pos, title=""):
    G2 = nx.DiGraph()

    # Aggiunta dei nodi al grafo
    for node in pos:
        G2.add_node(node, pos=pos[node])

    # Aggiunta degli archi al grado
    for i in range(len(tour) - 1):
        G2.add_edge(tour[i], tour[i + 1])

    # Percorso ottimale
    G2.add_edge(tour[-1], tour[0])

    # Posizione dei nodi
    node_positions = nx.get_node_attributes(G2, 'pos')

    # Grafo
    nx.draw(G2, pos=node_positions, with_labels=True,
        node_size=700, font_size=15, font_color='black',
        node_color='royalblue', edge_color='white', linewidths=2)

    # Archi del percorso ottimale
```

```
23      nx.draw_networkx_edges(G2, pos=node_positions,
            edgelist=[(tour[i], tour[i + 1]) for i in range(len(tour) -
            1)], edge_color='skyblue', width=4.0)
24
25      # Aggiunta titolo
26      plt.title(title)
27
28      # Visualizzazione grafica
29      plt.show()
30
31
32  #Visualizzazione percorso ottimale
33  plot_tsp_solution(tour, PosNodi, title="Percorso Ottimale Truck")
```

Capitolo 4

Lower Bound & Upper Bound

Nel seguito vengono definite cinque procedure per due possibili determinazioni di un limite inferiore (LB) e tre possibili determinazioni di un limite superiore (UB) del problema.

4.1 Lower Bound

Il Lower Bound rappresenta un limite inferiore teorico della funzione obiettivo che si sta cercando di minimizzare. Indica un valore al di sotto del quale la soluzione ottima del problema non può andare. Ai fini pratici sono state introdotte due procedure per il calcolo di un LB, una basata sul rilassamento dei vincoli di interezza e una basata su un metodo prettamente pratico e numerico.

4.1.1 Lower Bound con Rilassamento del Vincoli di Interezza

In un problema di minimizzazione, è possibile ricavare un Lower Bound del problema rilassando uno dei vincoli, in questo caso rilassando i vincoli di interezza, avendo nel nostro problema delle variabili binarie che possono assumere solo valori pari a 0 e a 1. Tale rilassamento si riferisce alla pratica di rendere meno rigidi o meno vincolanti i requisiti di interezza nelle soluzioni di un problema di ottimizzazione. Per questi motivi, la soluzione che ne verrà fuori sarà non ammissibile. Qualora fosse ammissibile allora il Lower Bound ottenuto sarà anche la soluzione ottima.

CALCOLO LB

LB con rilassamento dei vincoli di interezza

```
#Rilasso ora i vincoli di interezza e ottimizzo il nuovo modello
m_r = gp.Model('TSP-Truck')
xr = m_r.addVars(tau.keys(), obj=tau, lb=0, ub=1,
    vtype=GRB.CONTINUOUS, name = 'xr')
yr = m_r.addVars(C_S,d, obj=tau1, vtype=GRB.BINARY,name = 'yr')
ur = {}
for i in N:
    ur[i] = m_r.addVar(obj=0,vtype=GRB.INTEGER,lb=0,ub=c+2 ,
        name='ur'+str(i))
    if i == 0:
        m_r.addConstr(ur[i]==0)
zr = m_r.addVar(vtype=GRB.CONTINUOUS, name = 'zr')

objr = m_r.setObjective(zr, GRB.MINIMIZE)

m_r.addConstr( zr >= gp.quicksum(tau[i,j]*xr[i,j] for i in N0 for j
    in N1 if j != i))

for v in V:
    m_r.addConstr( zr >= gp.quicksum((tau1[0,i]+tau1[i,c+1]) *
        yr[i,v] for i in C_S))

for j in C:
    m_r.addConstr((gp.quicksum(xr[i,j] for i in N0 if i != j) +
        gp.quicksum(yr[j,v] for v in V if j in C_S)) == 1)

m_r.addConstr(gp.quicksum(xr[0,j] for j in N1) == 1)

m_r.addConstr(gp.quicksum(xr[i,c+1] for i in N0) == 1)

for j in C:
    m_r.addConstr(gp.quicksum(xr[i,j] for i in N0 if i != j) ==
        gp.quicksum(xr[j,k] for k in N1 if k != j))

```

```
m_r.addConstrs(ur[i]-ur[j]+1 <= (c+2)*(1-xr[i,j]) for i in N for j
    in N1 if j != i);

m_r.optimize()
# Ottimizzazione del modello

LB_r = m_r.ObjVal
LB_r = round(LB_r,3)
print('\n',"LB del problema rilassato:", LB_r)
```

4.1.2 Lower Bound con Metodo Pratico

È stato inserito anche quest'altro metodo che in alcuni casi potrebbe dare un risultato migliore e un LB più grande del precedente. Tale metodo consiste nel considerare tutti i clienti che non appartengono all'insieme C'' e prendere i minimi di ogni riga dalla nuova matrice vertice-arco, tenendo presente che tali clienti possono essere raggiunti solo dal camion (considero i tempi $\tau_{i,j}$).

LB con metodi pratici

```
#Considero solo i clienti non appartenenti all'insieme C'' in
    quanto sono quei clienti che possono essere raggiunti
#solo ed esclusivamente dal Truck in modo tale da ottenere
    sicuramente un LB non ammissibile
C_NS = [elem for elem in CL if elem not in C_S] #Clienti non
    appartenenti all'insieme C''

print('\n', "Clienti non appartenenti all'insieme C'':",C_NS)

if len(C_NS) != 0:
    C_NS.insert(0,0) #Aggiungo il nodo deposito (nella matrice
        svolge sia il ruolo di nodo di partenza che nodo di
        arrivo(13))
matrice_vertice_arco_1 = [[tau.get((i,j),0) if i != j else 0 for j
    in C_NS] for i in C_NS] #Costruzione matrice con costi tau
#Stampa matrice
#for riga in matrice_vertice_arco_1:
#    print(riga)
```

```
LB1 = sum([min(filter(lambda x: x!= 0,riga)) for riga in
    matrice_vertice_arco_1]) #Somma dei minimi di ogni riga
LB1 = round(LB1,3)
print('\n',"LB1: ", LB1)
```

4.1.3 Scelta del miglior Lower Bound

Una volta fatto ciò basta scegliere tra i due il Lower Bound migliore, ovvero il LB più grande.

Scelta miglior LB

```
#Prendo tra i due LB calcolati il migliore.
LB=max(LB1,LB_r)
print('\n',"LB: ",LB)
#Il primo metodo, in alcuni dei casi considerati, e' risultato
    migliore di quello del problema rilassato. Piu clienti
#appartengono all'insieme C'', peggiore sara' pero' tale LB
```

4.2 Upper Bound

L'Upper Bound, invece, rappresenta un limite superiore teorico della funzione obiettivo e, nel caso di un problema a minimizzare, rappresenta anche una soluzione ammissibile. L'UB è, dunque, un valore oltre il quale la soluzione ottima non può andare. Esso rappresenta la soluzione ottima solo ed esclusivamente se coincide con il LB. Ai fini pratici sono state introdotte tre procedure per il calcolo di un UB, una basata sull'ottimizzazione di un problema di TSP considerando solo il percorso del camion, una basata su un metodo prettamente pratico e numerico (metodo del Nearest Neighbor), utile qualora il metodo precedente risulti troppo lungo e oneroso, e uno basato sull'ottimizzazione di un problema di TSP con in aggiunta un numero di clienti serviti dagli UAV pari al numero degli UAV stessi.

4.2.1 Upper Bound con TSP del Truck

È possibile calcolare un UB del nostro problema ipotizzando che tutte le consegne vengano effettuate solo ed esclusivamente dal camion, senza considerare nessuno degli UAV a disposizione. In questo modo otterrò una soluzione sicuramente ammissibile ma con ogni probabilità peggiore della soluzione ottima.

CALCOLO UB

UB con TSP del Truck

```
#Costruiamo un nuovo modello, uguale a quello precedente ma con la mancanza degli UAV e, dunque, delle variabili y[i,v]
#L'ottimo di questo nuovo modello rappresenta un UB del mio modello di partenza
m_truck = gp.Model('TSP-Truck')
xt = m_truck.addVars(tau.keys(), obj=tau, vtype=GRB.BINARY, name = 'xt')
ut = {}
for i in N:
    ut[i] = m_truck.addVar(obj=0,vtype=GRB.INTEGER,lb=0,ub=c+2 , name='ut'+str(i))
    if i == 0:
        m_truck.addConstr(ut[i]==0)
zt = m_truck.addVar(vtype=GRB.CONTINUOUS, name = 'zt')

obj_truck = m_truck.setObjective(zt, GRB.MINIMIZE)

m_truck.addConstr( zt >= gp.quicksum(tau[i,j]*xt[i,j] for i in N0 for j in N1 if j != i))

for j in C:
    m_truck.addConstr(gp.quicksum(xt[i,j] for i in N0 if i != j) == 1)

m_truck.addConstr(gp.quicksum(xt[0,j] for j in N1) == 1)

m_truck.addConstr(gp.quicksum(xt[i,c+1] for i in N0) == 1)

for j in C:
    m_truck.addConstr(gp.quicksum(xt[i,j] for i in N0 if i != j) == gp.quicksum(xt[j,k] for k in N1 if k != j))

m_truck.addConstrs(ut[i]-ut[j]+1 <= (c+2)*(1-xt[i,j]) for i in N for j in N1 if j != i);
```

```
27
28
29 m_truck.optimize()
30 # Ottimizzazione del modello
31 m_truck._vars = xt
32 vals_truck = m_truck.getAttr('X', xt)
33 tour_truck = subtour(vals_truck)
34
35 print('')
36 print('Tour Ottimo del Truck senza UAV: %s' % str(tour_truck))
37 UB = m_truck.ObjVal
38 UB = round(UB,3)
39 print('\n',"UB: ", UB)
```

4.2.2 Upper Bound con Metodo Pratico

È stata inserita questa ulteriore procedura in quanto, in alcuni casi, come ad esempio la presenza di un elevato numero di clienti in prossimità del deposito, il solutore potrebbe impiegare un tempo abbastanza elevato per risolvere anche questo problema di TSP (oltre a quello iniziale di interesse). Tale procedura utilizza il metodo del Nearest Neighbor, il quale consiste nel selezionare, ad ogni passo, il nodo successivo come quello più vicino al nodo corrente. Questo processo continua fino a quando tutti i nodi sono visitati. In questo modo ottengo sicuramente una soluzione ammissibile, ma tale metodo produrrà un UB sicuramente superiore, o al massimo, in rari casi, uguale a quello calcolato con il metodo precedente.

UB con metodo pratico

```
1 #Utilizzo quest'altro metodo per il calcolo dell'UB quando non e'
      possibile calcolarlo in tempo utile e veloce con
2 #il metodo precedente. Questo avviene quando ho un numero elevato
      di nodi vicino al deposito.
3 matrice_vertice_arco_tot = [[tau.get((i,j),0) if i != j else 0 for
      j in N0] for i in N0] #Costruzione matrice con costi tau
4 #Metodo Nearest Neighbor: seleziono iterativamente il nodo
      successivo che e' il piu vicino al nodo attuale costruendo in
      questo
5 #modo un percorso che visita tutti i nodi esattamente una volta
6
```

```
def costruisci_ub_nearest_neighbor(matrice_vertice_arco):
    numero_nodi = len(matrice_vertice_arco)
    nodo_attuale = 0  # Iniziamo dal nodo 0
    nodi_non_visitati = set(range(1, numero_nodi))
    percorso = [nodo_attuale]
    costo_totale = 0

    while nodi_non_visitati:
        nodo_successivo = min(nodi_non_visitati, key=lambda x:
            matrice_vertice_arco[nodo_attuale][x])
        #Trovo il nodo x nell'insieme nodi_non_visitati che ha la
            distanza minima dal nodo attuale,utilizzando la funzione
        #lambda come criterio di confronto per trovare il nodo piu
            vicino a quello attuale non ancora visitato
        costo_totale+=
            matrice_vertice_arco[nodo_attuale][nodo_successivo]
        percorso.append(nodo_successivo)
        nodi_non_visitati.remove(nodo_successivo)
        nodo_attuale = nodo_successivo

    costo_totale += matrice_vertice_arco[percorso[-1]][percorso[0]]
        # Torniamo al nodo di partenza
    return percorso, costo_totale

percorso_nn,ub_nn =
    costruisci_ub_nearest_neighbor(matrice_vertice_arco_tot)

percorso_nn.append(c+1)
print("Percorso Ammissibile Truck:", percorso_nn)
print("\nUB:", ub_nn)
#Tale UB sara' sempre maggiore o al massimo (difficilmente) uguale a
    quello trovato con il TSP del Truck.
```

4.2.3 Upper Bound con UAV a servizio unico e TSP del Truck

È possibile calcolare un UB del nostro problema ipotizzando che ognuno degli UAV disponibili effettui una sola consegna, mentre i restanti clienti vengano serviti dal camion. In questo modo otterrò una soluzione sicuramente ammissibile e un UB, con ogni probabilità, migliore rispetto a quello calcolato solo con il TSP del camion.

UB con UAV a servizio unico e TSP del Truck

```
# UB con UAV bloccato a servire un numero di clienti pari al numero
    di UAV
#Costruiamo un nuovo modello, uguale a quello precedente ma con
    l'insieme C_S (insieme dei nodi raggiungibili dagli UAV)
#contenente un numero di nodi pari, al piu', a quello degli UAV.
#L'ottimo di questo nuovo modello rappresenta un UB del mio modello
    di partenza

m_UAV = gp.Model('UAV e TSP Truck')
C_S1 = []

t = 0
for i in C_P:
    tau1[0,i]=round(dist[0,i]/vel_UAV,3)
    T_A=tau1[0,i]
    tau1[i,c+1]=round(T_A-T_A*(1-3/4)*P[i-1]/3,3)
    T_R=tau1[i,c+1]
    if t<d:
        if T_A+T_R <= 0.5:
            C_S1.append(i)
            t=t+1

print("Clienti appartenenti all'insieme C'':", '\n', C_S1, '\n')

xu = m_UAV.addVars(tau.keys(), obj=tau, vtype=GRB.BINARY, name =
    'xu')
yu = m_UAV.addVars(C_S,d, obj=tau1, vtype=GRB.BINARY,name = 'yu')

uu = {}
for i in N:
```

```
27      uu[i] = m_UAV.addVar(obj=0,vtype=GRB.INTEGER,lb=0,ub=c+2,
            name='uu'+str(i))
28      if i == 0:
29          m_UAV.addConstr(uu[i]==0)
30
31  zu = m_UAV.addVar(vtype=GRB.CONTINUOUS, name = 'zu')
32
33  obj_UAV = m_UAV.setObjective(zu, GRB.MINIMIZE)
34
35  m_UAV.addConstr( zu >= gp.quicksum(tau[i,j]*xu[i,j] for i in N0 for
        j in N1 if j != i))
36
37  for v in V:
38      m_UAV.addConstr( zu >= gp.quicksum((tau1[0,i]+tau1[i,c+1]) *
            yu[i,v] for i in C_S1))
39
40  for j in C:
41      m_UAV.addConstr((gp.quicksum(xu[i,j] for i in N0 if i != j) +
            gp.quicksum(yu[j,v] for v in V if j in C_S1)) == 1)
42
43  m_UAV.addConstr(gp.quicksum(xu[0,j] for j in N1) == 1)
44
45  m_UAV.addConstr(gp.quicksum(xu[i,c+1] for i in N0) == 1)
46
47  for j in C:
48      m_UAV.addConstr(gp.quicksum(xu[i,j] for i in N0 if i != j) ==
            gp.quicksum(xu[j,k] for k in N1 if k != j))
49
50  m_UAV.addConstrs(uu[i]-uu[j]+1 <= (c+2)*(1-xu[i,j]) for i in N for
        j in N1 if j != i);
51
52  m_UAV.optimize()
53  # Ottimizzazione del modello
54  m_UAV._vars = xu
55  vals_UAV = m_UAV.getAttr('X', xu)
56  tour_UAV = subtour(vals_UAV)
57
```

```
print('')
print('Tour Ottimo del Truck senza un numero di nodi pari al numero
    di UAV: %s' % str(tour_UAV))
UB3 = m_UAV.ObjVal
UB3 = round(UB3,3)
print('\n',"UB con UAV a serivzio unico: ", UB3)
```

4.2.4 Scelta del miglior UB

Una volta fatto ciò basta scegliere, tra i tre, l'Upper Bound migliore, ovvero l'UB più piccolo (con ogni probabilità tale UB sarà il terzo calcolato).

Scelta del miglior UB

```
#Prendo tra i tre UB calcolati il migliore.
UB=min(UB1,ub_nn,UB3)
print('\n',"UB: ",UB)
```

Capitolo 5

Istanze

Per testare il codice introdotto nei capitoli precedenti sono mostrate nel seguito quattro istanze di dimensioni crescenti, con un'ultima istanza che evidenzia un esempio attraverso il quale il problema viene stressato e per il quale Gurobi non trova una soluzione in tempi brevi.

5.1 Istanza Piccola

```
Numero clienti 17
Nodo deposito 0
Numero UAV 1
Elenco nodi con posizione e peso pacco
0 10 10
1 10.5 20.0 3.8
2 30.3 28.2 10.5
3 26.4 -20.4 3.2
4 -35.7 15.0 8.1
5 15.7 10.1 1.7
6 20.9 10.2 2.5
7 0 0 2
8 7.4 -8.0 3
9 10.1 5.1 1
10 0 10.5 2
11 2.5 2.5 3
12 13.7 7.0 4
13 -10.2 -2.6 7
14 -5.4 9.8 2.7
15 2.3 -6.9 3.9
16 4.8 -9.4 4.9
17 7.8 6.9 5.2
18 10 10
```

Figura 5.1: File input contenente 17 clienti

```
Posizione dei clienti ai quali effettuare la consegna:
 {1: (10.5, 20.0), 2: (30.3, 28.2), 3: (26.4, -20.4), 4: (-35.7, 15.0), 5: (15.7, 10.1), 6: (20.9, 10.2), 7: (0.0, 0.0), 8: (7.
4, -8.0), 9: (10.1, 5.1), 10: (0.0, 10.5), 11: (2.5, 2.5), 12: (13.7, 7.0), 13: (-10.2, -2.6), 14: (-5.4, 9.8), 15: (2.3, -6.
9), 16: (4.8, -9.4), 17: (7.8, 6.9)}

Posizione di tutti i nodi della rete:
 {0: (10.0, 10.0), 1: (10.5, 20.0), 2: (30.3, 28.2), 3: (26.4, -20.4), 4: (-35.7, 15.0), 5: (15.7, 10.1), 6: (20.9, 10.2), 7:
(0.0, 0.0), 8: (7.4, -8.0), 9: (10.1, 5.1), 10: (0.0, 10.5), 11: (2.5, 2.5), 12: (13.7, 7.0), 13: (-10.2, -2.6), 14: (-5.4, 9.
8), 15: (2.3, -6.9), 16: (4.8, -9.4), 17: (7.8, 6.9), 18: (10.0, 10.0)}

Clienti appartenenti all'insieme C':
 [1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16]
```

Figura 5.2:

```
Clienti appartenenti all'insieme C'':
 [1, 5, 6, 9, 10, 11, 12]
```

Figura 5.3:

```
Root relaxation: objective 4.487263e+00, 50 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    4.48726    0   23    7.59700    4.48726  40.9%     -    0s
     0     0    4.81495    0   21    7.59700    4.81495  36.6%     -    0s
H    0     0                       5.9950000    4.88250  18.6%     -    0s
H    0     0                       5.8650000    4.88250  16.8%     -    0s
     0     0    4.88250    0   23    5.86500    4.88250  16.8%     -    0s
     0     0    4.88700    0   24    5.86500    4.88700  16.7%     -    0s
H    0     0                       5.5960000    4.88700  12.7%     -    0s
H    0     0                       5.1990000    4.88700  6.00%     -    0s
     0     0    4.88700    0   25    5.19900    4.88700  6.00%     -    0s
     0     0    4.88700    0   28    5.19900    4.88700  6.00%     -    0s
     0     0    4.88700    0   23    5.19900    4.88700  6.00%     -    0s
H    0     0                       5.1330000    4.88700  4.79%     -    0s
     0     0    4.88700    0   27    5.13300    4.88700  4.79%     -    0s
     0     0    4.88700    0   27    5.13300    4.88700  4.79%     -    0s
     0     0    4.88700    0   30    5.13300    4.88700  4.79%     -    0s
     0     0    4.88700    0   30    5.13300    4.88700  4.79%     -    0s
     0     0    4.88700    0   27    5.13300    4.88700  4.79%     -    0s
     0     0    4.88700    0   28    5.13300    4.88700  4.79%     -    0s
H    0     0                       5.1310000    4.88700  4.76%     -    0s
     0     0    4.89955    0   30    5.13100    4.89955  4.51%     -    0s
H    0     0                       5.1180000    4.89955  4.27%     -    0s
     0     0    4.89955    0   28    5.11800    4.89955  4.27%     -    0s
H    0     0                       5.0520000    4.90268  2.96%     -    0s
     0     0    4.90268    0   33    5.05200    4.90268  2.96%     -    0s
     0     0    4.90268    0   26    5.05200    4.90268  2.96%     -    0s
     0     0    4.90268    0   34    5.05200    4.90268  2.96%     -    0s
     0     0    4.90268    0   42    5.05200    4.90268  2.96%     -    0s
     0     0    4.90268    0   40    5.05200    4.90268  2.96%     -    0s
     0     0    4.93500    0   35    5.05200    4.93500  2.32%     -    0s
     0     0    4.93500    0   21    5.05200    4.93500  2.32%     -    0s
     0     0    4.93500    0   17    5.05200    4.93500  2.32%     -    0s
     0     0    4.93500    0   29    5.05200    4.93500  2.32%     -    0s
     0     0    4.95000    0   32    5.05200    4.95000  2.02%     -    0s
     0     0    5.00700    0   30    5.05200    5.00700  0.89%     -    0s
     0     0    5.01916    0   31    5.05200    5.01916  0.65%     -    0s
     0     0    5.01932    0   31    5.05200    5.01932  0.65%     -    0s
```

Figura 5.4:

```
Cutting planes:
  Learned: 2
  Gomory: 6
  Cover: 1
  Implied bound: 9
  MIR: 10
  StrongCG: 1
  Zero half: 2
  RLT: 7
  Relax-and-lift: 7

Explored 1 nodes (518 simplex iterations) in 0.26 seconds (0.06 work units)
Thread count was 8 (of 8 available processors)

Solution count 9: 5.052 5.118 5.131 ... 7.597

Optimal solution found (tolerance 1.00e-04)
Best objective 5.052000000000e+00, best bound 5.052000000000e+00, gap 0.0000%

Tour Ottimo del Truck: [0, 2, 3, 8, 16, 15, 7, 13, 4, 14, 17, 18]
Costo Ottimo: 5.052

L'UAV 1 ha raggiunto il cliente 1
L'UAV 1 ha raggiunto il cliente 5
L'UAV 1 ha raggiunto il cliente 6
L'UAV 1 ha raggiunto il cliente 9
L'UAV 1 ha raggiunto il cliente 10
L'UAV 1 ha raggiunto il cliente 11
L'UAV 1 ha raggiunto il cliente 12
```

Figura 5.5: Stampa del percorso ottimale, costo associato e clienti serviti dall'UAV

Di seguito sono riportate le visualizzazioni grafiche della posizione dei clienti e di quelle visitabili dall'UAV in funzione del vincolo sul peso del pacco, rispettivamente in Figura 5.6 e 5.7

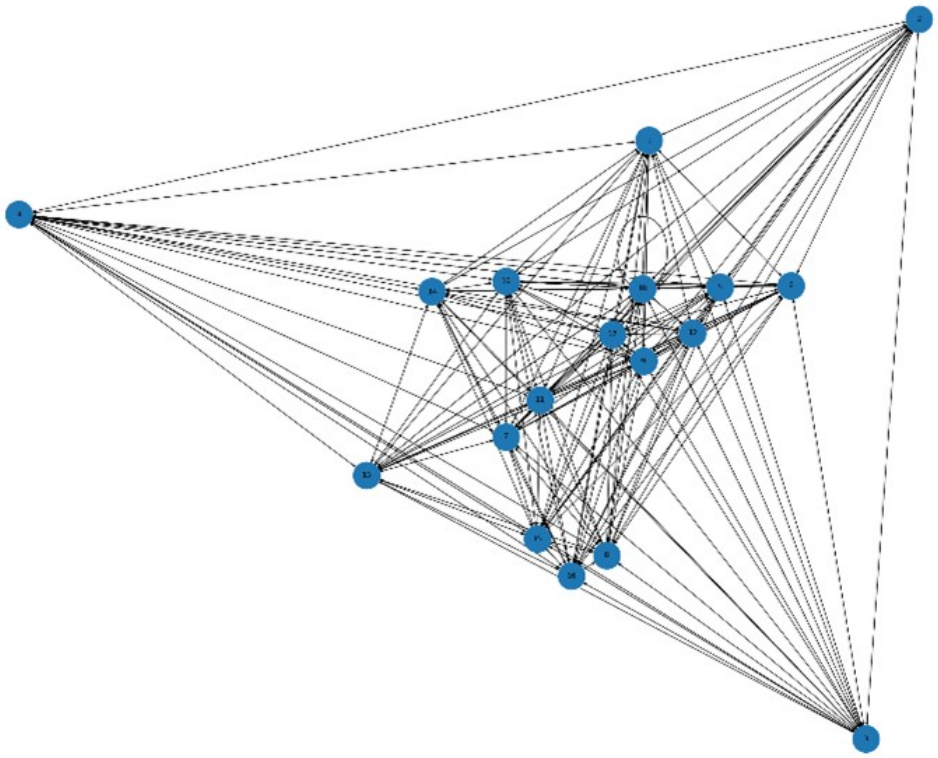


Figura 5.6: Posizione dei nodi della rete e relativi archi

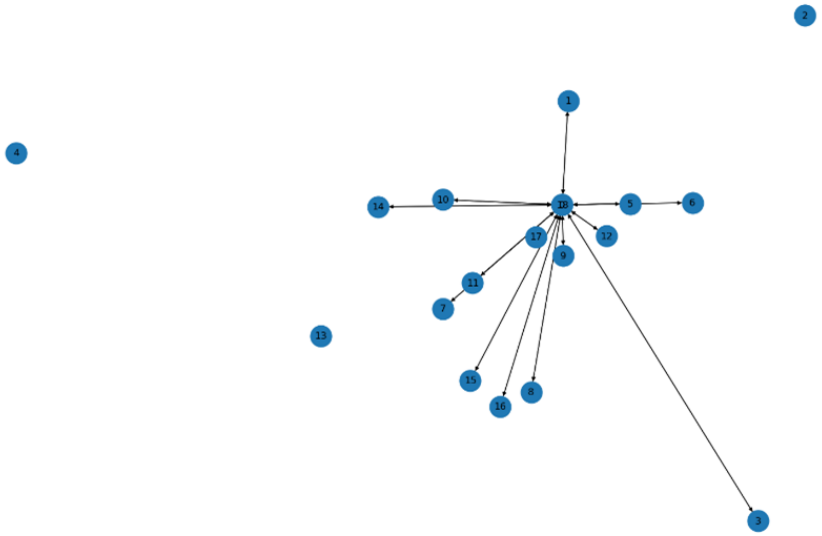


Figura 5.7: Nodi raggiungibili dall'UAV per il vincolo sul peso

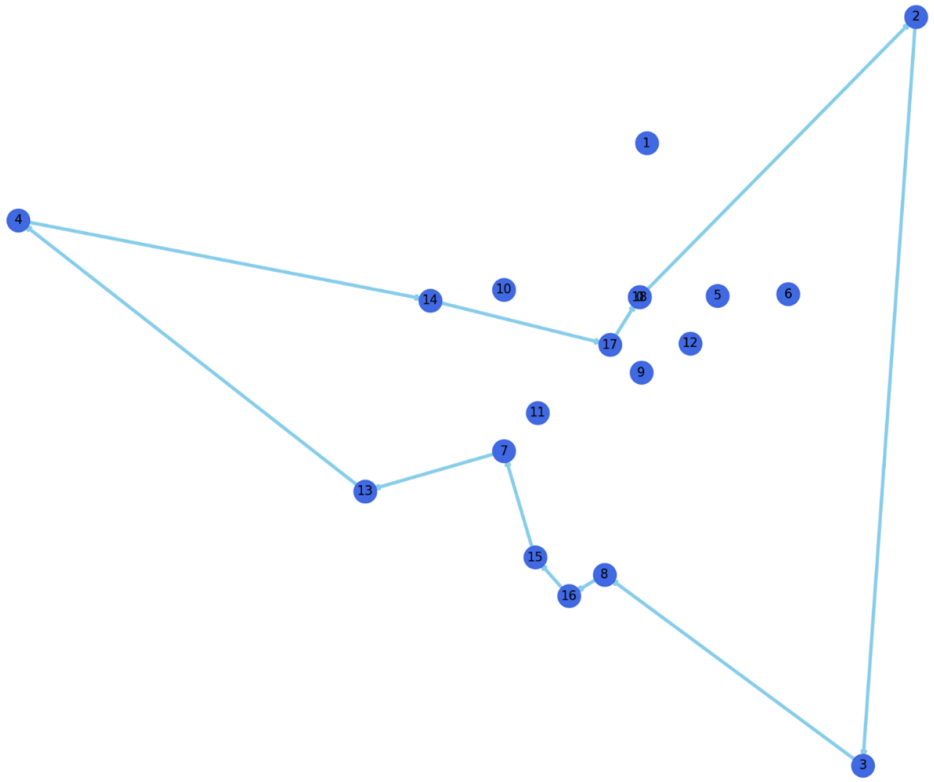


Figura 5.8: Percorso ottimo del truck

Nella Figura 5.8 è evidenziato il percorso ottimale del Truck. Come si può notare dagli output del programma, Figura 5.5, il tour ottimo del furgone è [0, 2, 3, 8, 16, 15, 7, 13, 4, 14, 17, 18], con un costo di 5.052 ore. Il resto dei clienti viene servito contemporaneamente dell'UAV. Si può confrontare l'ottimo riscontrato con i valori di Upper Bound e Lower Bound calcolati con i metodi descritti nel Capitolo 4.

```
Clienti non appartenenti all'insieme C'':  [2, 3, 4, 7, 8, 13, 14, 15, 16, 17]

LB1:  3.169
```

Figura 5.9: LB con metodo pratico

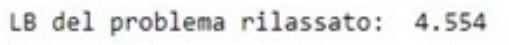


Figura 5.10: LB del problema rilassato

Il Lower Bound più grande tra i due calcolati viene preso come valore finale, che in questo caso corrisponde a LB = 4.554. Come ci si aspettava è inferiore rispetto all'ottimo, rappresentando di fatto un Lower Bound.

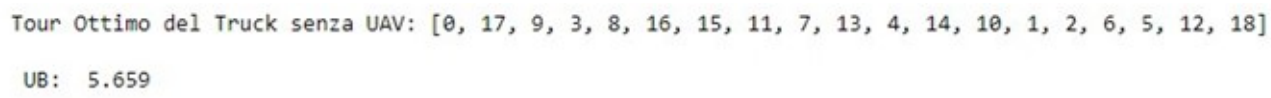


Figura 5.11: UB con TSP del Truck

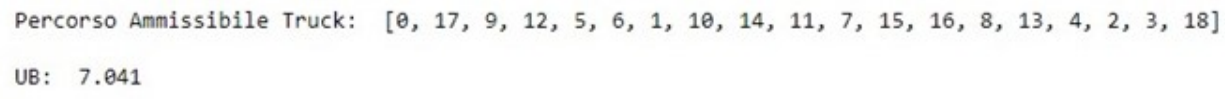


Figura 5.12: UB con metodo del Nearest Neighbor

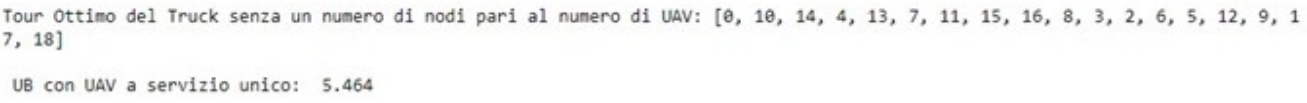


Figura 5.13: UB con UAV a servizio unico e TSP del Truck

L'Upper Bound più piccolo tra i tre calcolati viene preso come valore finale, che in questo caso corrisponde a UB = 5.464. Essendo maggiore rispetto all'ottimo, risulterà essere l'Upper Bound del problema.

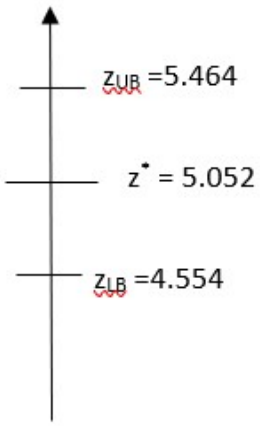


Figura 5.14: Confronto UB e LB con il valore ottimo

5.2 Istanza Media

```
Numero clienti 25
Nodo deposito 0
Numero UAV 2
Elenco nodi con posizione e peso pacco
0 10 10
1 10 20 3.8
2 30 28 10.5
3 26 20 3.2
4 35 15 8.1
5 15 10 1.7
6 20 10 2.5
7 0 0 2
8 7 8 3
9 10 5 1
10 0 10 2
11 2.5 2.5 3
12 13 7 4
13 12 6 10
14 18 19 2
15 6 5 3
16 4 3 1
17 27 2 15
18 16 16 1
19 14 13 5
20 10 19 4
21 21 29 6
22 28 30 9
23 2 6 4
24 1 5 2
25 20 21 1
26 10 10
```

Figura 5.15: File input contenente 40 clienti

```
Posizione dei clienti ai quali effettuare la consegna:
 {1: (10.0, 20.0), 2: (30.0, 28.0), 3: (26.0, 20.0), 4: (35.0, 15.0), 5: (15.0, 10.0), 6: (20.0, 10.0), 7: (0.0, 0.0), 8: (7.0,
8.0), 9: (10.0, 5.0), 10: (0.0, 10.0), 11: (2.5, 2.5), 12: (13.0, 7.0), 13: (12.0, 6.0), 14: (18.0, 19.0), 15: (6.0, 5.0), 16:
(4.0, 3.0), 17: (27.0, 2.0), 18: (16.0, 16.0), 19: (14.0, 13.0), 20: (10.0, 19.0), 21: (21.0, 29.0), 22: (28.0, 30.0), 23: (2.
0, 6.0), 24: (1.0, 5.0), 25: (20.0, 21.0)}

Posizione di tutti i nodi della rete:
 {0: (10.0, 10.0), 1: (10.0, 20.0), 2: (30.0, 28.0), 3: (26.0, 20.0), 4: (35.0, 15.0), 5: (15.0, 10.0), 6: (20.0, 10.0), 7: (0.
0, 0.0), 8: (7.0, 8.0), 9: (10.0, 5.0), 10: (0.0, 10.0), 11: (2.5, 2.5), 12: (13.0, 7.0), 13: (12.0, 6.0), 14: (18.0, 19.0), 1
5: (6.0, 5.0), 16: (4.0, 3.0), 17: (27.0, 2.0), 18: (16.0, 16.0), 19: (14.0, 13.0), 20: (10.0, 19.0), 21: (21.0, 29.0), 22: (2
8.0, 30.0), 23: (2.0, 6.0), 24: (1.0, 5.0), 25: (20.0, 21.0), 26: (10.0, 10.0)}

Clienti appartenenti all'insieme C':
 [1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 18, 19, 20, 23, 24, 25]
```

Figura 5.16:

```
Clienti appartenenti all'insieme C'':
 [1, 5, 6, 8, 9, 10, 11, 12, 14, 15, 16, 18, 19, 20, 23, 24]
```

Figura 5.17:

```
Root relaxation: objective 1.804945e+00, 300 iterations, 0.01 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

    0     0    1.80495    0   35          -    1.80495      -     -    0s
    0     0    2.06750    0   33          -    2.06750      -     -    0s
    0     0    2.06750    0   34          -    2.06750      -     -    0s
    0     0    2.20500    0   30          -    2.20500      -     -    0s
    0     0    2.20500    0   31          -    2.20500      -     -    0s
    0     0    2.20544    0   30          -    2.20544      -     -    0s
    0     0    2.20544    0   30          -    2.20544      -     -    0s
H   0     0                       3.6270000    2.20544  39.2%     -    0s
    0     0    2.20544    0   37    3.62700    2.20544  39.2%     -    0s
H   0     0                       3.5150000    2.20544  37.3%     -    0s
    0     0    2.20544    0   37    3.51500    2.20544  37.3%     -    0s
H   0     0                       3.4730000    2.20900  36.4%     -    0s
H   0     2                       3.4190000    2.20900  35.4%     -    0s
    0     2    2.20900    0   37    3.41900    2.20900  35.4%     -    0s
H   31    39                      3.1150000    2.21816  28.8%   8.8    0s
H  107   118                      3.0070000    2.21816  26.2%   5.8    0s
H  113   118                      2.8980000    2.21816  23.5%   5.7    0s
H 1173   867                      2.7620000    2.30943  16.4%   4.1    1s
H 1195   837                      2.7610000    2.35570  14.7%   4.1    2s
 5731  1850    2.67067   41   40    2.76100    2.46657  10.7%   6.7    5s
22694  7907 infeasible   61         2.76100    2.52437  8.57%   6.9   10s
29264  9307    2.74717   50   32    2.76100    2.53890  8.04%   7.0   15s
29316  9342    2.58433   46   45    2.76100    2.53890  8.04%   7.0   20s
33300  9826    2.75292   88   21    2.76100    2.53890  8.04%   7.1   25s
46133 10490    2.61700   63   18    2.76100    2.54266  7.91%   7.1   30s
57104  9685    2.65599   67   26    2.76100    2.56492  7.10%   7.2   35s
68229 11627    2.69100   67   11    2.76100    2.58489  6.38%   7.3   40s
81126 12991    2.62323   63   34    2.76100    2.60237  5.75%   7.5   46s
92886 13954    2.66137   62   19    2.76100    2.61720  5.21%   7.7   50s
107244 14121    2.73120   77   21    2.76100    2.63415  4.59%   7.9   55s
122074 13269 infeasible   58         2.76100    2.65240  3.93%   8.0   60s
135803 11489    2.67863   71   19    2.76100    2.66972  3.31%   8.1   65s
148811  8548 infeasible   67         2.76100    2.68807  2.64%   8.3   70s
161456  3348    2.74556   68   26    2.76100    2.71711  1.59%   8.4   75s
```

Figura 5.18:

```
Cutting planes:
  Learned: 32
  Gomory: 40
  Lift-and-project: 22
  Cover: 108
  Implied bound: 48
  Projected implied bound: 7
  MIR: 98
  StrongCG: 4
  Flow cover: 298
  Inf proof: 96
  Zero half: 102
  RLT: 34
  Relax-and-lift: 57

Explored 167125 nodes (1405494 simplex iterations) in 77.10 seconds (32.50 work units)
Thread count was 8 (of 8 available processors)

Solution count 9: 2.761 2.762 2.898 ... 3.627

Optimal solution found (tolerance 1.00e-04)
Best objective 2.761000000000e+00, best bound 2.761000000000e+00, gap 0.0000%

Tour Ottimo del Truck: [0, 11, 7, 13, 17, 4, 3, 2, 22, 21, 25, 26]
Costo Ottimo: 2.761

L'UAV 2 ha raggiunto il cliente 1
L'UAV 1 ha raggiunto il cliente 5
L'UAV 2 ha raggiunto il cliente 6
L'UAV 1 ha raggiunto il cliente 8
L'UAV 1 ha raggiunto il cliente 9
L'UAV 2 ha raggiunto il cliente 10
L'UAV 2 ha raggiunto il cliente 12
L'UAV 1 ha raggiunto il cliente 14
L'UAV 2 ha raggiunto il cliente 15
L'UAV 1 ha raggiunto il cliente 16
L'UAV 1 ha raggiunto il cliente 18
L'UAV 1 ha raggiunto il cliente 19
L'UAV 2 ha raggiunto il cliente 20
L'UAV 1 ha raggiunto il cliente 23
L'UAV 1 ha raggiunto il cliente 24
```

Figura 5.19: Stampa del percorso ottimale, costo associato e clienti serviti dall'UAV

Di seguito sono riportate le visualizzazioni grafiche della posizione dei clienti e di quelle visitabili dall'UAV in funzione del vincolo sul peso del pacco, rispettivamente in Figura 5.20 e 5.21

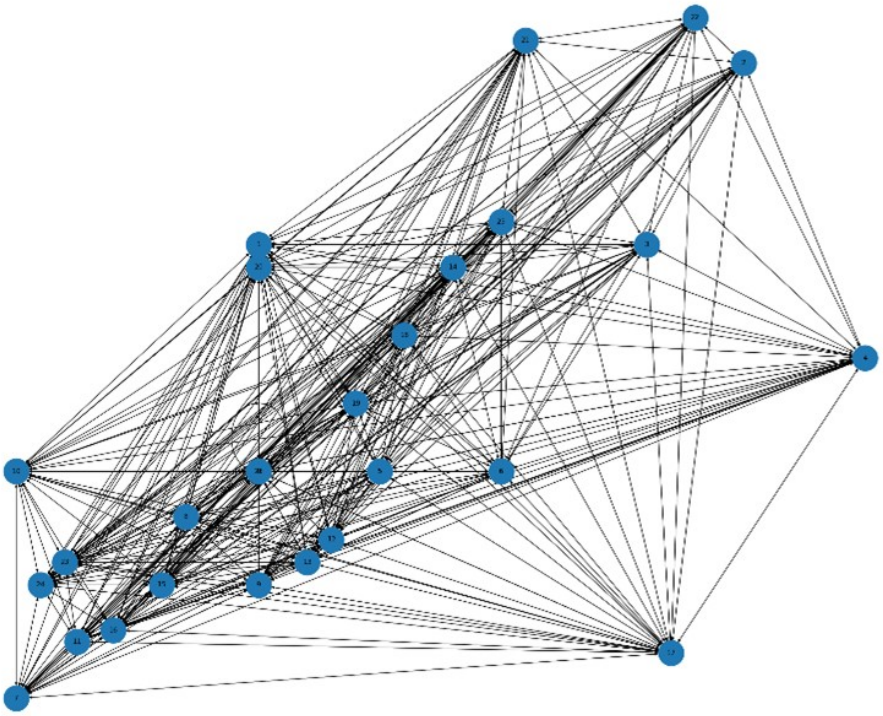


Figura 5.20: Posizione dei nodi della rete e relativi archi

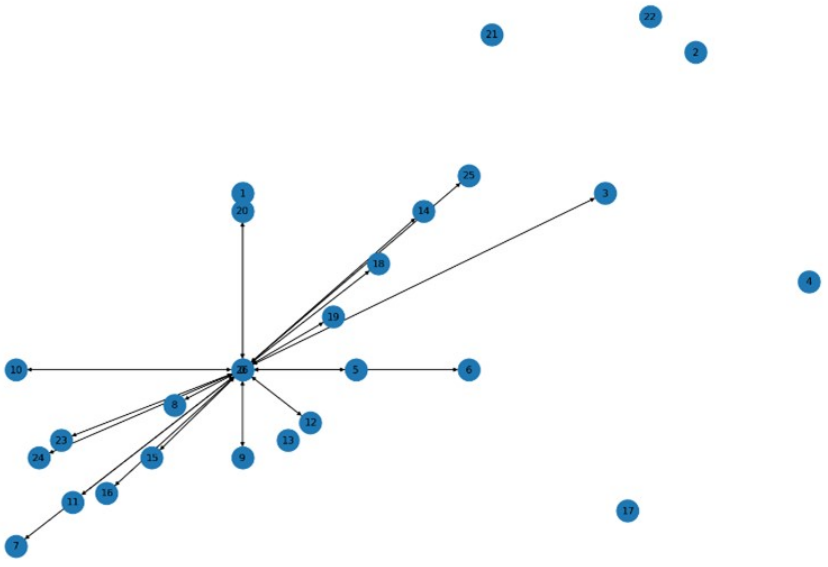


Figura 5.21: Nodi raggiungibili dall'UAV per il vincolo sul peso

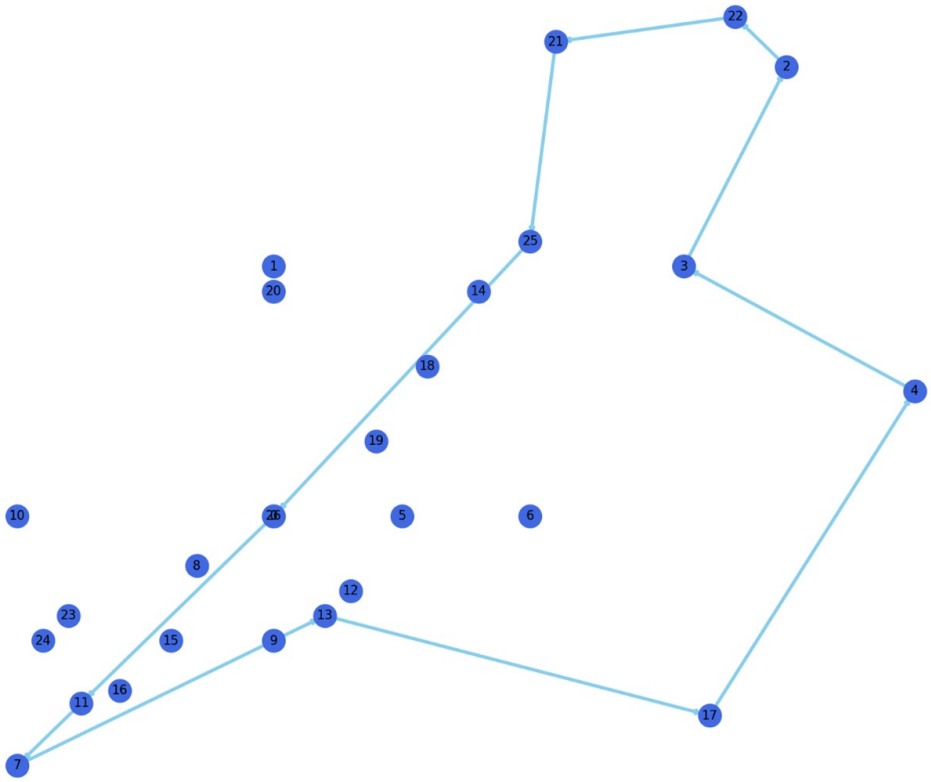


Figura 5.22: Percorso ottimo del truck

Nella Figura 5.22 è evidenziato il percorso ottimale del Truck. Come si può notare dagli output del programma, Figura 5.19, il tour ottimo del furgone è [0, 11, 7, 13, 17, 4, 3, 2, 22, 21, 25, 26], con un costo di 2.761 ore. Il resto dei clienti viene servito contemporaneamente dell'UAV. Si può confrontare l'ottimo riscontrato con i valori di Upper Bound e Lower Bound calcolati con i metodi descritti nel Capitolo 4.

```
Clienti non appartenenti all'insieme C'':  [2, 3, 4, 7, 13, 17, 21, 22, 25]
LB1:  1.821
```

Figura 5.23: LB con metodo pratico

```
LB del problema rilassato:  1.858
```

Figura 5.24: LB del problema rilassato

Il Lower Bound più grande tra i due calcolati viene preso come valore finale, che in questo caso corrisponde a LB = 1.858. Come ci si aspettava è inferiore rispetto all'ottimo, rappresentando di fatto un Lower Bound.

```
Tour Ottimo del Truck senza UAV: [0, 19, 20, 1, 18, 14, 25, 21, 22, 2, 3, 4, 17, 6, 5, 12, 13, 9, 15, 16, 11, 7, 24, 23, 10, 8,
26]

 UB con TSP Truck:  3.401
```

Figura 5.25: UB con TSP del Truck

```
Percorso Ammissibile Truck:  [0, 8, 15, 16, 11, 24, 23, 10, 7, 9, 13, 12, 5, 19, 18, 14, 25, 3, 2, 22, 21, 1, 20, 6, 17, 4, 26]

UB_NN:  4.166
```

Figura 5.26: UB con metodo del Nearest Neighbor

```
Tour Ottimo del Truck senza un numero di nodi pari al numero di UAV: [0, 8, 10, 23, 24, 7, 11, 16, 15, 9, 13, 12, 6, 17, 4, 3,
2, 22, 21, 25, 14, 18, 20, 19, 26]

 UB con UAV a servizio unico:  3.339
```

Figura 5.27: UB con UAV a servizio unico e TSP del Truck

L'Upper Bound più piccolo tra i tre calcolati viene preso come valore finale, che in questo caso corrisponde a UB = 3.339. Essendo maggiore rispetto all'ottimo, risulterà essere l'Upper Bound del problema.

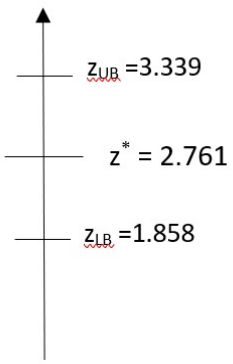


Figura 5.28: Confronto UB e LB con il valore ottimo

5.3 Istanza Grande

```
Numero clienti 40
Nodo deposito 0
Numero UAV 3
Elenco nodi con posizione e peso pacco
0 10 10
1 10 20 3.8
2 30 28 10.5
3 26 20 3.2
4 35 15 8.1
5 15 10 1.7
6 20 10 2.5
7 0 0 2
8 7 8 3
9 10 5 1
10 0 10 2
11 2.5 2.5 3
12 13.2 5.4 4
13 12 6 10
14 18 19 2
15 6 5 3
16 4 3 1
17 21.4 2.5 2.3
18 16.1 14.2 1
19 14 13 3
20 10 19 4
21 21 29 6
22 28 30 4
23 2 6 4
24 1 5 2
25 11.1 11.9 1
26 6.6 -6.6 4
27 -2.2 2.8 4.4
28 1.2 5.9 10
29 1.4 2.9 1.2
30 3.4 5.6 2
31 3.9 -4.2 4.6
32 -2.5 -6.4 4.9
33 3.6 -2.6 2.1
34 -1.7 4.2 4.6
35 -20.4 20.4 10.2
36 -2.6 4.9 7.5
37 5.9 -5.9 5.9
38 3.6 -5.2 1.2
39 2.5 6.8 4.6
40 2.4 6.8 3.6
41 10 10
```

Figura 5.29: File input contenente 40 clienti

```
Posizione dei clienti ai quali effettuare la consegna:
 {1: (10.0, 20.0), 2: (30.0, 28.0), 3: (26.0, 20.0), 4: (35.0, 15.0), 5: (15.0, 10.0), 6: (20.0, 10.0), 7: (0.0, 0.0), 8: (7.0,
8.0), 9: (10.0, 5.0), 10: (0.0, 10.0), 11: (2.5, 2.5), 12: (13.2, 5.4), 13: (12.0, 6.0), 14: (18.0, 19.0), 15: (6.0, 5.0), 16:
(4.0, 3.0), 17: (21.4, 2.5), 18: (16.1, 14.2), 19: (14.0, 13.0), 20: (10.0, 19.0), 21: (21.0, 29.0), 22: (28.0, 30.0), 23: (2.
0, 6.0), 24: (1.0, 5.0), 25: (11.1, 11.9), 26: (6.6, -6.6), 27: (-2.2, 2.8), 28: (1.2, 5.9), 29: (1.4, 2.9), 30: (3.4, 5.6), 3
1: (3.9, -4.2), 32: (-2.5, -6.4), 33: (3.6, -2.6), 34: (-1.7, 4.2), 35: (-20.4, 20.4), 36: (-2.6, 4.9), 37: (5.9, -5.9), 38:
(3.6, -5.2), 39: (2.5, 6.8), 40: (2.4, 6.8)}

Posizione di tutti i nodi della rete:
 {0: (10.0, 10.0), 1: (10.0, 20.0), 2: (30.0, 28.0), 3: (26.0, 20.0), 4: (35.0, 15.0), 5: (15.0, 10.0), 6: (20.0, 10.0), 7: (0.
0, 0.0), 8: (7.0, 8.0), 9: (10.0, 5.0), 10: (0.0, 10.0), 11: (2.5, 2.5), 12: (13.2, 5.4), 13: (12.0, 6.0), 14: (18.0, 19.0), 1
5: (6.0, 5.0), 16: (4.0, 3.0), 17: (21.4, 2.5), 18: (16.1, 14.2), 19: (14.0, 13.0), 20: (10.0, 19.0), 21: (21.0, 29.0), 22: (2
8.0, 30.0), 23: (2.0, 6.0), 24: (1.0, 5.0), 25: (11.1, 11.9), 26: (6.6, -6.6), 27: (-2.2, 2.8), 28: (1.2, 5.9), 29: (1.4, 2.9),
30: (3.4, 5.6), 31: (3.9, -4.2), 32: (-2.5, -6.4), 33: (3.6, -2.6), 34: (-1.7, 4.2), 35: (-20.4, 20.4), 36: (-2.6, 4.9), 37:
(5.9, -5.9), 38: (3.6, -5.2), 39: (2.5, 6.8), 40: (2.4, 6.8), 41: (10.0, 10.0)}

Clienti appartenenti all'insieme C':
 [1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27, 29, 30, 31, 32, 33, 34, 38, 39, 40]
```

Figura 5.30:

```
Clienti appartenenti all'insieme C'':
 [1, 5, 6, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 23, 24, 25, 27, 29, 30, 31, 34, 39, 40]
```

Figura 5.31:

```
1353392 38920    4.15607   42   15    4.16900    4.14750  0.52%  11.7 1325s
1357374 37174 infeasible   62         4.16900    4.14800  0.50%  11.7 1330s
1361400 35364 infeasible   52         4.16900    4.14867  0.49%  11.7 1335s
1367404 32635    4.15396   57    8    4.16900    4.14960  0.47%  11.7 1342s
1369513 32518     cutoff   40         4.16900    4.15000  0.46%  11.7 1345s
1371970 30445 infeasible   45         4.16900    4.15036  0.45%  11.7 1350s
1376294 28430 infeasible   43         4.16900    4.15107  0.43%  11.7 1355s
1379847 26370    4.16817   54   24    4.16900    4.15190  0.41%  11.7 1360s
1383859 24143    4.16706   50   48    4.16900    4.15264  0.39%  11.7 1365s
1389556 20971     cutoff   53         4.16900    4.15400  0.36%  11.7 1371s
1393379 18919     cutoff   59         4.16900    4.15482  0.34%  11.7 1375s
1398967 15590    4.15660   55   25    4.16900    4.15633  0.30%  11.7 1381s
1402582 13414    4.16332   55   63    4.16900    4.15750  0.28%  11.7 1385s

1407446 10048 infeasible   65         4.16900    4.15923  0.23%  11.7 1390s
1413523  5759     cutoff   52         4.16900    4.16207  0.17%  11.7 1400s
1419022   567     cutoff   48         4.16900    4.16600  0.07%  11.7 1405s

Cutting planes:
  Learned: 162
  Gomory: 86
  Lift-and-project: 146
  Cover: 202
  Implied bound: 67
  Projected implied bound: 14
  MIR: 326
  StrongCG: 12
  Flow cover: 800
  GUB cover: 2
  Inf proof: 270
  Zero half: 210
  RLT: 47
  Relax-and-lift: 319
  PSD: 5
```

Figura 5.32:

```
Explored 1421223 nodes (16645278 simplex iterations) in 1406.49 seconds (704.46 work units)
Thread count was 8 (of 8 available processors)

Solution count 10: 4.169 4.2 4.202 ... 4.605

Optimal solution found (tolerance 1.00e-04)
Best objective 4.169000000000e+00, best bound 4.169000000000e+00, gap 0.0000%

Tour Ottimo del Truck: [0, 3, 4, 2, 22, 21, 35, 36, 28, 24, 7, 32, 38, 26, 37, 33, 13, 41]
Costo Ottimo: 4.169

L'UAV 1 ha raggiunto il cliente 1
L'UAV 1 ha raggiunto il cliente 5
L'UAV 2 ha raggiunto il cliente 6
L'UAV 1 ha raggiunto il cliente 8
L'UAV 2 ha raggiunto il cliente 9
L'UAV 2 ha raggiunto il cliente 10
L'UAV 1 ha raggiunto il cliente 11
L'UAV 3 ha raggiunto il cliente 12
L'UAV 2 ha raggiunto il cliente 14
L'UAV 2 ha raggiunto il cliente 15
L'UAV 1 ha raggiunto il cliente 16
L'UAV 2 ha raggiunto il cliente 17
L'UAV 3 ha raggiunto il cliente 18
L'UAV 1 ha raggiunto il cliente 19
L'UAV 1 ha raggiunto il cliente 20
L'UAV 2 ha raggiunto il cliente 23
L'UAV 2 ha raggiunto il cliente 25
L'UAV 2 ha raggiunto il cliente 27
L'UAV 3 ha raggiunto il cliente 29
L'UAV 2 ha raggiunto il cliente 30
L'UAV 3 ha raggiunto il cliente 31
L'UAV 3 ha raggiunto il cliente 34
L'UAV 2 ha raggiunto il cliente 39
L'UAV 3 ha raggiunto il cliente 40
```

Figura 5.33: Stampa del percorso ottimale, costo associato e clienti serviti dall'UAV

Di seguito sono riportate le visualizzazioni grafiche della posizione dei clienti e di quelle visitabili dall'UAV in funzione del vincolo sul peso del pacco, rispettivamente in Figura 5.34 e 5.35

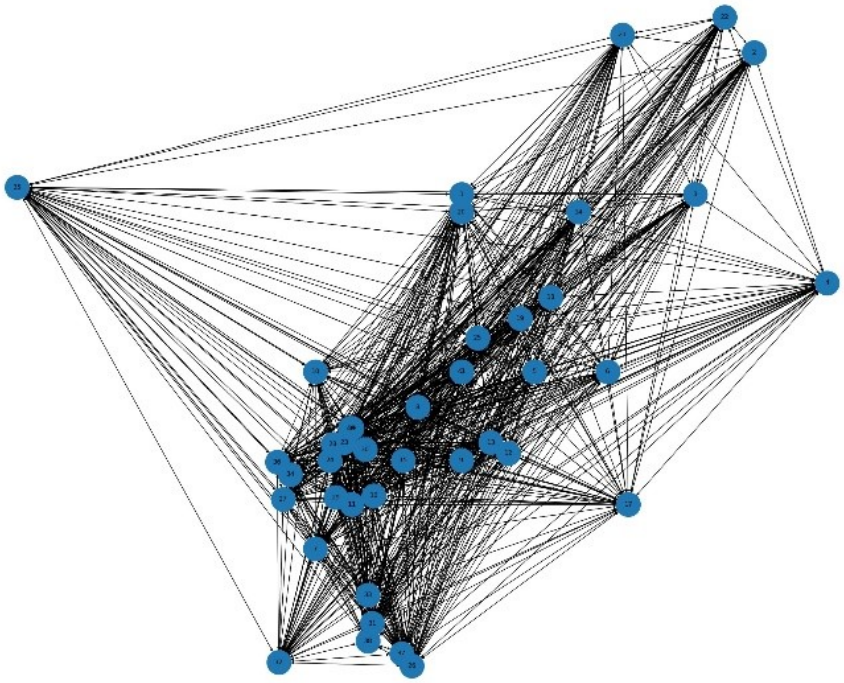


Figura 5.34: Posizione dei nodi della rete e relativi archi

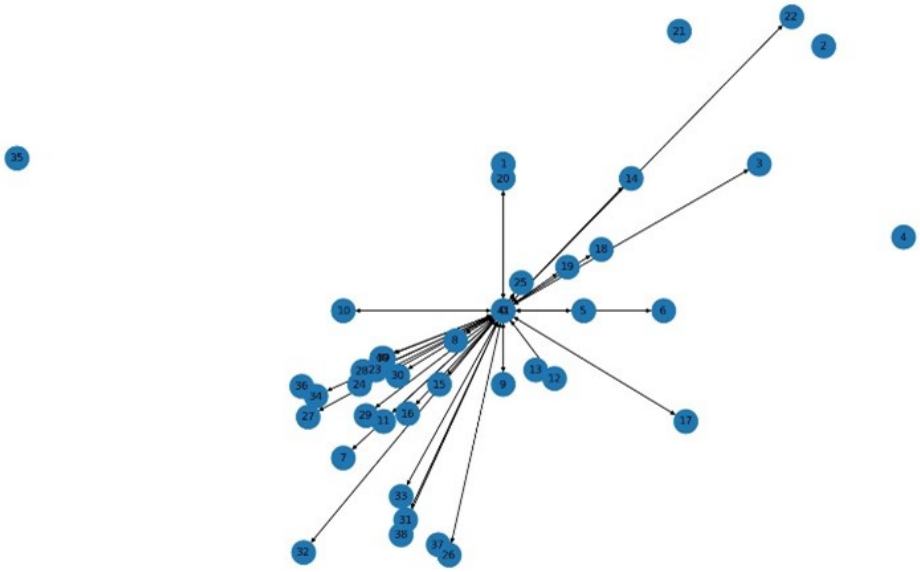


Figura 5.35: Nodi raggiungibili dall'UAV per il vincolo sul peso

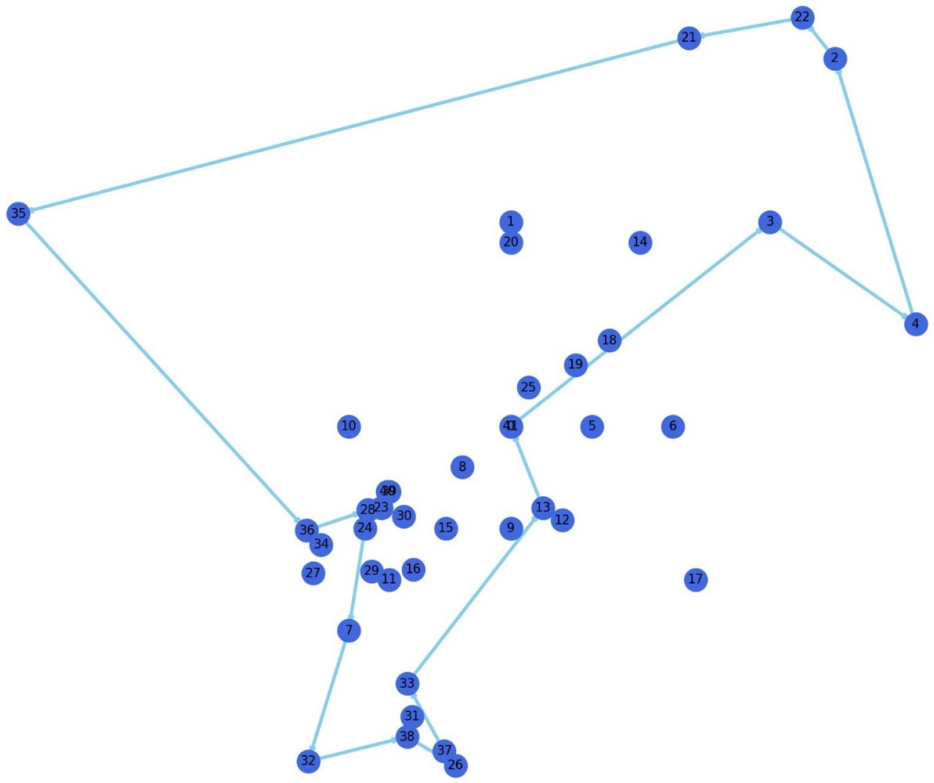


Figura 5.36: Percorso ottimo del truck

Nella Figura 5.36 è evidenziato il percorso ottimale del Truck. Come si può notare dagli output del programma, Figura 5.33, il tour ottimo del furgone è [0, 3, 4, 2, 22, 21, 35, 36, 28, 24, 7, 32, 38, 26, 37, 33 ,13, 41], con un costo di 4.169 ore. Il resto dei clienti viene servito contemporaneamente dell'UAV. Si può confrontare l'ottimo riscontrato con i valori di Upper Bound e Lower Bound calcolati con i metodi descritti nel Capitolo 4.

```
Clienti non appartenenti all'insieme C'':  [2, 3, 4, 7, 13, 21, 22, 26, 28, 32, 33, 35, 36, 37, 38]
LB1:  2.251
```

Figura 5.37: LB con metodo pratico

```
LB del problema rilassato:  2.829
```

Figura 5.38: LB del problema rilassato

Il Lower Bound più grande tra i due calcolati viene preso come valore finale, che in questo caso corrisponde a LB = 2.829. Come ci si aspettava è inferiore rispetto all'ottimo, rappresentando di fatto un Lower Bound.

```
Tour Ottimo del Truck senza UAV: [0, 25, 20, 1, 21, 22, 2, 4, 3, 14, 18, 19, 5, 6, 17, 12, 13, 9, 15, 30, 39, 40, 23, 28, 24, 2
9, 11, 16, 33, 31, 37, 26, 38, 32, 7, 27, 34, 36, 35, 10, 8, 41]

 UB con TSP Truck:  5.226
```

Figura 5.39: UB con TSP del Truck

```
Percorso Ammissibile Truck:  [0, 25, 19, 18, 5, 12, 13, 9, 15, 30, 23, 28, 24, 29, 11, 16, 39, 40, 10, 36, 34, 27, 7, 33, 31,
8, 37, 26, 32, 8, 20, 1, 14, 3, 2, 22, 21, 6, 17, 4, 35, 41]

UB_NN:  6.8180000000000005
```

Figura 5.40: UB con metodo del Nearest Neighbor

```
Tour Ottimo del Truck senza un numero di nodi pari al numero di UAV: [0, 25, 19, 18, 20, 14, 21, 22, 2, 3, 4, 17, 12, 13, 9, 1
5, 30, 39, 40, 23, 28, 24, 29, 11, 16, 33, 31, 37, 26, 38, 32, 7, 27, 34, 36, 35, 10, 8, 41]

 UB con UAV a servizio unico:  5.011
```

Figura 5.41: UB con UAV a servizio unico e TSP del Truck

L'Upper Bound più piccolo tra i tre calcolati viene preso come valore finale, che in questo caso corrisponde a UB = 5.011. Essendo maggiore rispetto all'ottimo, risulterà essere l'Upper Bound del problema.

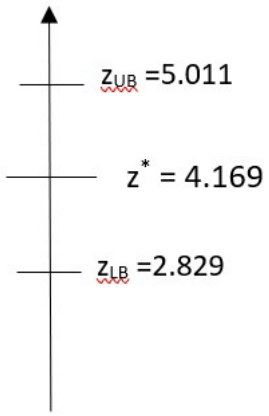


Figura 5.42: Confronto UB e LB con il valore ottimo

5.4 Istanza Critica

Negli esempi precedenti il solutore è riuscito a risolvere i vari problemi in quanto la posizione dei clienti era tale che non tutti si trovassero in prossimità del deposito. Nel momento in cui i clienti sono per la maggior parte ad una distanza tale per cui venga soddisfatta la relazione $\tau'_{0,i} + \tau'_{i,c+1} \leq e$, in particolar modo, molto vicini tra di loro e al deposito, il solutore non trova una soluzione in tempi utili. In questo modo non faccio altro che stressare il problema ad un caso critico difficilmente risolvibile. Nel seguito è mostrato un chiaro esempio di questa problematica.

```
numero clienti 32
nodo deposito 0
numero UAV 1
elenco nodi con posizione, peso pacco
0 10 10
1 1.0 2.0 3.8
2 3.0 2.8 1
3 2.6 2.0 3
4 3.5 1.5 1
5 1.5 1.0 1.7
6 2.0 1.0 2.5
7 0 0 2
8 7 8 3
9 10 5 1
10 0 10 2
11 2.5 2.5 3
12 1.3 7 4 4
13 2 7.0 4
14 5 4.5 4
15 3 3.5 4
16 4 1.5 4
17 9 4.0 3
18 4 2.1 2
19 6 1.0 5
20 6 2.8 4
21 7 2.3 1
22 8 1.5 4
23 4 4 3
24 6 6.9 2
25 6 5.0 1
26 8 7.0 2
27 2 1.0 1
28 3 1.0 2
29 1.1 -1.1 1
30 1.2 1.2 2
31 1.3 3 3
32 4 1.4 4
33 10 10
```

Figura 5.43: File input contenente 32 clienti molto vicini al deposito

```
Posizione dei clienti ai quali effettuare la consegna:
 {1: (1.0, 2.0), 2: (3.0, 2.8), 3: (2.6, 2.0), 4: (3.5, 1.5), 5: (1.5, 1.0), 6: (2.0, 1.0), 7: (0.0, 0.0), 8: (7.0, 8.0), 9: (1
0.0, 5.0), 10: (0.0, 10.0), 11: (2.5, 2.5), 12: (1.3, 7.0), 13: (2.0, 7.0), 14: (5.0, 4.5), 15: (3.0, 3.5), 16: (4.0, 1.5), 17:
(9.0, 4.0), 18: (4.0, 2.1), 19: (6.0, 1.0), 20: (6.0, 2.8), 21: (7.0, 2.3), 22: (8.0, 1.5), 23: (4.0, 4.0), 24: (6.0, 6.9), 25:
(6.0, 5.0), 26: (8.0, 7.0), 27: (2.0, 1.0), 28: (3.0, 1.0), 29: (1.1, -1.1), 30: (1.2, 1.2), 31: (1.3, 3.0), 32: (4.0, 1.4)}

Posizione di tutti i nodi della rete:
 {0: (10.0, 10.0), 1: (1.0, 2.0), 2: (3.0, 2.8), 3: (2.6, 2.0), 4: (3.5, 1.5), 5: (1.5, 1.0), 6: (2.0, 1.0), 7: (0.0, 0.0), 8:
(7.0, 8.0), 9: (10.0, 5.0), 10: (0.0, 10.0), 11: (2.5, 2.5), 12: (1.3, 7.0), 13: (2.0, 7.0), 14: (5.0, 4.5), 15: (3.0, 3.5), 1
6: (4.0, 1.5), 17: (9.0, 4.0), 18: (4.0, 2.1), 19: (6.0, 1.0), 20: (6.0, 2.8), 21: (7.0, 2.3), 22: (8.0, 1.5), 23: (4.0, 4.0),
24: (6.0, 6.9), 25: (6.0, 5.0), 26: (8.0, 7.0), 27: (2.0, 1.0), 28: (3.0, 1.0), 29: (1.1, -1.1), 30: (1.2, 1.2), 31: (1.3, 3.
0), 32: (4.0, 1.4), 33: (10.0, 10.0)}

Clienti appartenenti all'insieme C':
 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]
```

Figura 5.44:

```
Clienti appartenenti all'insieme C'':
 [1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 30, 31, 32]
```

Figura 5.45:

```
4284355 1475499    0.98430   66   16    1.03700    0.95900  7.52%  11.4 3781s
4288250 1477007 infeasible   99         1.03700    0.95900  7.52%  11.4 3785s
4292964 1478774    0.98030   47   25    1.03700    0.95900  7.52%  11.4 3790s
4295652 1480874    0.95900  103   26    1.03700    0.95900  7.52%  11.4 3795s
4300051 1483009 infeasible   86         1.03700    0.95900  7.52%  11.4 3800s
4305241 1485133    0.96205   64   26    1.03700    0.95900  7.52%  11.4 3805s
4310238 1487374     cutoff  122         1.03700    0.95900  7.52%  11.4 3811s
4314354 1488720    1.00060   98   32    1.03700    0.95900  7.52%  11.4 3815s
4320067 1491303     cutoff  103         1.03700    0.95900  7.52%  11.4 3820s
4325175 1493108    0.96452   64   27    1.03700    0.95900  7.52%  11.4 3828s
4325828 1493716    0.95900  107   37    1.03700    0.95900  7.52%  11.4 3830s
4330781 1496242    0.98665   84   12    1.03700    0.95900  7.52%  11.5 3835s
4335999 1498100    1.01174   95   20    1.03700    0.95900  7.52%  11.5 3840s
4339900 1499620    0.97378   98   31    1.03700    0.95900  7.52%  11.5 3845s
4345567 1502041    0.95900   51   20    1.03700    0.95900  7.52%  11.5 3850s
4350727 1504225    0.98186   77   36    1.03700    0.95900  7.52%  11.5 3856s
4354263 1505988    1.02606  108   28    1.03700    0.95900  7.52%  11.5 3860s
4359325 1507916    0.98905  137   12    1.03700    0.95900  7.52%  11.5 3865s
4362800 1509470    0.98800   68    9    1.03700    0.95900  7.52%  11.5 3870s
```

Figura 5.46:

Come possiamo notare dalla Figura 5.46, il solutore, dopo un tempo di circa 64 minuti, non è riuscito a trovare una soluzione ottima ma è fermo ad un gap tra UB e LB del 7.52% con LB pari a 0.95900 e UB pari a 1.03700.

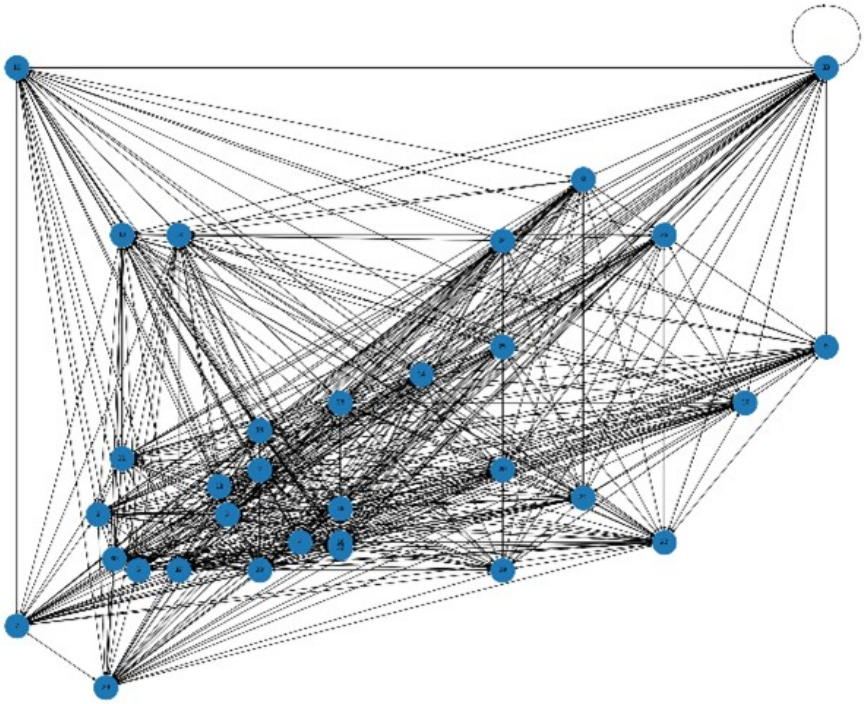


Figura 5.47: Posizione dei nodi della rete e relativi archi

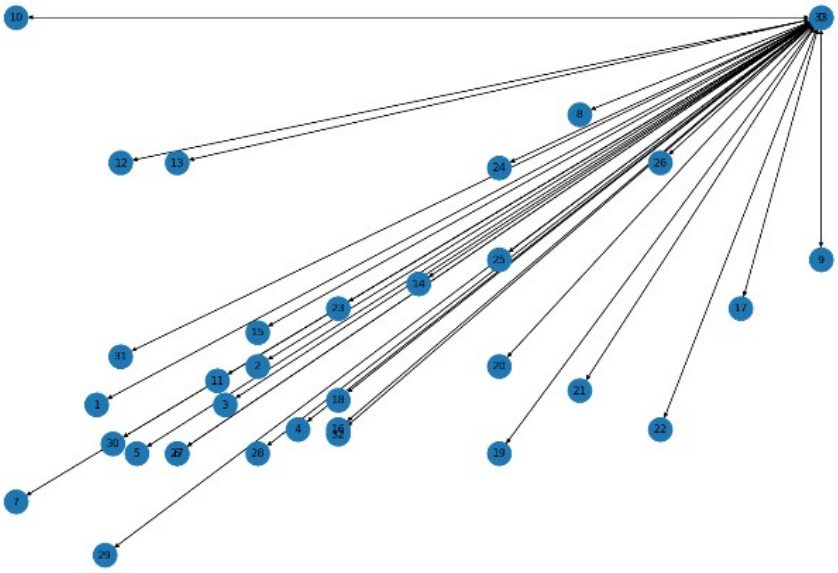


Figura 5.48: Nodi raggiungibili dall'UAV per il vincolo sul peso

```
Clienti non appartenenti all'insieme C'':  [7, 29]

LB1:  0.432
```

Figura 5.49: LB con metodo pratico

```
LB del problema rilassato:  0.71
```

Figura 5.50: LB del problema rilassato

```
Percorso Ammissibile Truck:  [0, 8, 26, 24, 25, 14, 23, 15, 2, 11, 3, 4, 16, 32, 18, 28, 6, 27, 5, 30, 1, 31, 7, 29, 19, 21, 2
0, 22, 17, 9, 13, 12, 10, 33]

UB:  1.584
```

Figura 5.51: UB con metodo del Nearest Neighbor

Questo è uno dei casi in cui è utile utilizzare un metodo pratico quanto più buono possibile per il calcolo dell'UB in quanto anche il semplice problema del TSP, quando ho un numero elevato di nodi molto vicini tra di loro, non verrà risolto in tempo utile dal solutore.

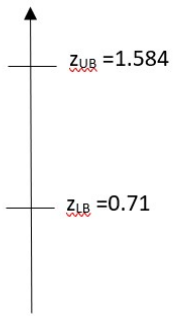


Figura 5.52: UB e LB del problema critico

Capitolo 6

Considerazioni

È utile fare alcune considerazioni riguardanti il come opera il solutore per quanto riguarda l'assegnazione dei clienti agli UAV. In presenza di più UAV, infatti, il solutore potrebbe non trovare differenze tra alcune soluzioni, come ad esempio quelle riportate in Figura 6.1. Questo provoca un albero del B&B tale per cui i nodi dell'ultimo livello potrebbero avere soluzioni uguali ma configurazioni diverse. Questo implica uno sforzo computazionale in più per il solutore in quanto esso non sa, in primo luogo, quale scegliere.

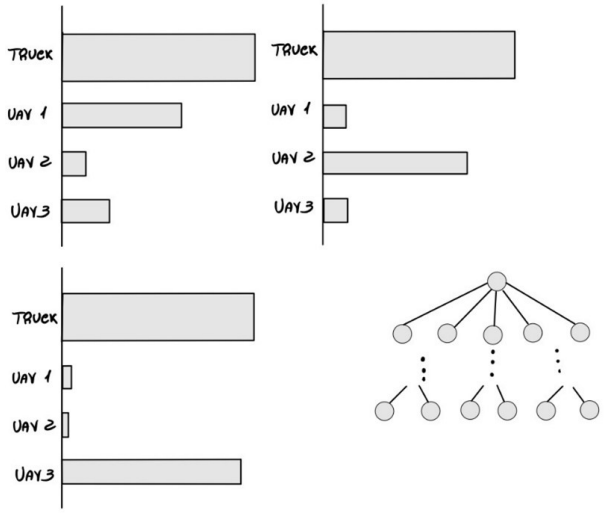


Figura 6.1:

Questo è il motivo per il quale ,spesso (come succede anche per la soluzione critica mostrata in precedenza nel paragrafo 5.4), il solutore potrebbe non trovare una soluzione in tempi utili. Sulla base di queste considerazioni, dunque, è facilmente intuibile come tale modello non controlli l'utilizzo dei vari UAV e quale di essi viene usato, più o meno, rispetto agli altri.