UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Vehicle Routing Problem with Time Windows

Elaborato di Algoritmi di Ottimizzazione Combinatoria e su Rete

Davide Mancinelli matr. M63001674

Angelo Caliolo matr. M63XXXX

Indice

Capitolo 1

Introduzione

Il Vehicle Routing with Time Windows (VRPTW) è una variante del Vehicle Routing Problem (VRP), in cui ogni flotta di veicoli con capacità limitata deve prelevare o consegnare merci (o persone) in differenti punti, ognuno dei quali accessibile solo in determinati intervalli di tempo. Quindi, a differenza del VRP standard, bisogna rispettare anche vincoli temporali per le consegne.

1.1 Assunzioni del problema

Prima di formalizzare il problema, è necessario elencare le seguenti assunzioni:

- E' presente un nodo deposito, da cui la flotta di veicoli inizia e termina il suo percorso
- Sono presenti dei nodi (chiamati punti vendita) in cui almeno un veicolo della flotta deve passare
- Ogni punto vendita deve essere transitato esclusivamente una volta
- Ogni punto vendita presenta un orario di apertura (prima del quale non è possibile ricevere consegne) e un orario di chiusura (oltre il quale non è possibile ricevere consegne)

- Nel caso in cui un veicolo arrivasse in un punto vendita prima dell'orario di apertura, tale veicolo deve attendere l'apertura del punto vendita

- Tutti i punti vendita devono essere serviti entro l'orario di chiusura

- L'obiettivo è minimizzare i costi di trasporto

1.2 Approcci per la risoluzione

Ci sono differenti approcci per la risoluzione del VRPTW; di seguito sono presentati quelli affrontati in questo elaborato:

- Algoritmi esatto: [CONTINUA]

- Algotimo euristico: [CONTINUA]

Capitolo 2

Definizione del problema

Il VRPWT può essere definito da:

- Flotta **V** di veicoli; con m=|V|
- Insieme **N** di nodi; tale insieme è costituito a sua volta da:
  - Insieme **P={1,...,p}** di punti vendita
  - Deposito **D={s,t}**; rappresentato da 2 nodi distinti: s come nodo di partenza, t come nodo di destinazione

  E' possibile quindi definire n=|N|=|P|+2, con N={s,1,...,p,t}
- Insieme **A** di archi orientati (i,j), dove $i \neq j$
- Grafo **G(V,N)** orientato e pieno

Altri elementi da prendere in considerazione sono:

- Il costo c_{ij} associato a ogni arco (i,j)
- Il tempo t_{ij} di percorrenza associato a ogni arco (i,j)
- La portata massima Q_k di ogni ogni veicolo k
- La richiesta d_i di merce da soddisfare per ogni punto vendita i
- La finestra di tempo $f_i = [a_i, b_i]$ in cui ogni punto vendita è aperto

- L'insieme di cluster $C = \{C_1, ..., C_m\}$; è possibile dividere l'insieme N in sottoinsiemi cluster C_k, ognuno associato al veicolo k
- Il tempo di servizio s_{ik} sul nodo i; in altre parole, è il tempo da attendere affinchè il nodo i sia totalmente servito. Ipotizziamo $s_{sk} = 0, s_{tk} = 0 \quad \forall k \in V$

2.1 Funzione obiettivo

L'obiettivo è quello di minimizzare i costi di ogni veicolo per servire tutti i punti vendita negli orari adeguati. Possiamo scrivere la funzione obiettivo come segue

$$min \sum_{k \in V} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ijk}$$

2.2 Variabili di decisione

Definiamo quindi le seguenti variabili di decisione:

- $x_{ijk} = 1$ se l'arco (i,j) è percorso dal veicolo k, 0 altrimenti
- Il tempo di arrivo l_{ik} del veicolo k al nodo i. Ipotizziamo $l_{sk} = 0 \quad \forall k \in V$, con $s \in D$

2.3 Vincoli

Definiamo i seguenti vincoli:

- Vincolo di capacità; la somma di tutte le richieste soddisfatte non deve superare la capacità massima del singolo veicolo

$$\sum_{i \in N} \sum_{j \in N} d_i x_{ijk} \leq Q_k \qquad \forall k \in V$$

- Vincoli di routing
  - Vincolo di unicità dei punti vendità; ogni punto vendita i deve essere visitato una sola volta

    $$\sum_{k \in V} \sum_{j \in N} x_{ijk} = 1 \qquad \forall i \in P$$

  - Vincolo sulla partenza del singolo veicolo; controlla che il veicolo sia partito dal deposito

    $$\sum_{j \in P} x_{sjk} = 1 \quad \forall k \in V$$

    In alternativa, nel caso in cui decidessimo che non è necessario che partano tutti i veicoli, basta imporre tale sommatoria ≤ 1
  - Vincolo sull'arrivo del singolo veicolo; controlla che il veicolo sia arrivato dal deposito

    $$\sum_{i \in P} x_{itk} = 1 \quad \forall k \in V$$

    In alternativa, nel caso in cui decidessimo che non è necessario che partano tutti i veicoli, basta imporre tale sommatoria ≤ 1
  - Vincolo di coerenza partenza-arrivo; tutti i veicoli che sono partiti, devono essere tornati

    $$\sum_{j \in P} x_{stk} = \sum_{i \in P} x_{jik} \quad \forall k \in V$$

  - Vincolo sul flusso dei punti vendita; una volta che un veicolo è entrato in un punto vendita, deve anche uscirne. In alternativa, possiamo dire che la somma dei nodi entrati in un punto vendita deve essere uguale a quella dei nodi uscenti

$$\sum_{i \in N} x_{ijk} = \sum_{i \in N} x_{jik} \quad \forall j \in P, \ \forall k \in V$$

Bisogna tenere conto del fatto che $j \in P$ e non N, poichè i nodi s e t (compatibili con il deposito) possono anche presentare solo nodi entranti o solo nodi uscenti

- Vincolo sui sottogiri; si vogliono evitare cicli, facendo in modo che, per ogni sottoinsieme di P, la somma degli archi con origine e destinazione in questo sottoinsieme non può superare la cardinalità di S-1, altrimenti si avrebbero cicli

  $$\sum_{i,j \in S} x_{ijk} \leq |S| - 1 \quad S \subset P \quad |S| \geq 2 \quad \forall k \in V$$

  Un esempio applicativo di questo vincolo è il seguente. Sia $P = 2, 3, 4, 5$ ed $S = 2, 3, 4$; il vincolo imponerebbe $\sum_{i,j \in S} x_{ijk} \leq 3 - 1 = 2$. Di conseguenza, se il percorso del veicolo fosse $2 \rightarrow 3 \rightarrow 4 \rightarrow 2$, la somma degli archi sarebbe pari a 3, violando il vincolo

- Vincoli temporali
  - Vincolo di inizio servizio; il tempo di inizio servizio l_{jk} deve essere almeno pari all'istante l_{ik} in cui si è attivati nel generico punti i, più il tempo per servire quel nodo, più il tempo per spostarsi lungo l'arco (i,j)

    $$l_{jk} \geq (l_{ik} + s_{ik} + t_{ij}) - (1 - x_{ijk})M \quad \forall i, j \in N \quad \forall k \in V$$

    con M costante sufficientemente grande. Tale costante viene aggiunta affinchè siano esclusi tutti gli archi non transitati
  - Vincolo sulla finestra temporale

    $$a_i \leq l_i \leq b_i$$

- Vincoli di dominio

- $x_{ijk} \in \{0, 1\}$
- $i, j \in N$
- $k \in V$
- $l_{ik} \in R^+ \cup \{0\}$
- $s_{ik} \in R^+ \cup \{0\}$

Capitolo 3

Risoluzione tramite algortimo esatto

Un problema VRPTW può essere risolto tramite un algoritmo esatto. Nel nostro caso è stato utilizzato **PuLP**, un modellatore scritto in Python

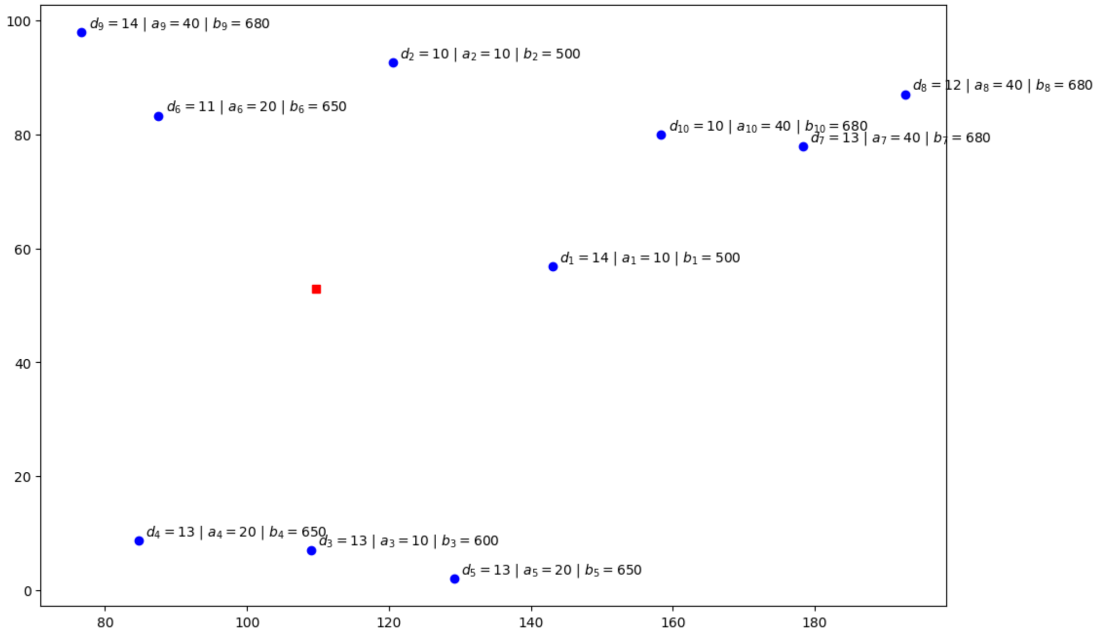
3.1 Creazione dei dati

I dati sono stati scritti secondo la seguente logica:

- Il numero dei punti vendita è stato scelto arbitrariamente da noi
- Le coordinate x e y di ogni punto vendita sono state scelte casualmente

```
n = 10
xc = rnd.rand(n+1)*200
xc = np.append(xc, xc[0])
yc = rnd.rand(n+1)*100
yc = np.append(yc, yc[0])
```

- L'insieme P dei punti vendita è un vettore che va da 1 a n
- L'insieme N dei nodi è costituito da P, preceduto da 0 e succeduto da n+1



- L'insieme A degli archi è un vettore dato dalle combinazione dei vari nodi
- I costi associati a ogni arco sono stati scelti casualmente
- Il tempo di percorrenza di ogni arco è pari alla distanza tra i nodi associati a ogni arco
- La richiesta di Ogni nodo è scelta casualmente
- Le finestre temporali di ogni nodo sono state scelte arbitrariamente da noi
- L'insieme Q della capacità è stato scelto arbitrariamente da noi
- I tempi di servizio di ogni nodo sono stati scelti casualmente

```
P = [il for il in range(1,n+1)]
N = [0] + P + [n+1]

A = [(i,j) for i in N for j in N if i!=j and
    (i,j)!=(n+1,0) and (i,j)!=(0,n+1)]

c = {(i,j): np.random.randint(2,15) for i,j
   in A}
t = {(i,j): int(np.hypot(xc[i]-xc[j], yc[i]-
   yc[j])) for i,j in A}
```

```
np.random.seed(0)
d = {i: np.random.randint(10,15) for i in P}
d[0] = 0
d[n+1] = 0

a = {0:0, 1:10, 2:10, 3:10, 4:20, 5:20,
   6:20, 7:40, 8:40, 9:40, 10:40, 11:0}
   # Orario di apertura
b = {0:200, 1:500, 2:500, 3:600, 4:650,
   5:650, 6:650, 7:680, 8:680, 9:680, 10:680,
    11:700} # Orario di chiusura

V = [1,2,3,4]
Q = {1: 50, 2:50, 3:25, 4:25}

s = {(i,k): np.random.randint(3,5) if i!=0
   and i!=n+1 else 0 for i in N for k in V}
```

3.2 Utilizzo di PuLP

Per prima cosa creiamo il modello tramite la libreria PuLP come segue:

```
model = pulp.LpProblem("VRPTW", pulp.LpMinimize)
```

3.2.1 Variabili di decisione

Si aggiungono le variabili di decisione:

```
arco_var = [(i,j,k) for i in N for j in N for k in V
   if i!=j and (i,j)!=(n+1,0) and (i,j)!=(0,n+1)]
arco_arrivo = [(i,k)for i in N for k in V]
x = pulp.LpVariable.dicts("x", arco_var, cat="Binary"
   )
l = pulp.LpVariable.dicts("x", arco_arrivo, lowBound
   = 0, cat="Continuous")
```

3.2.2 Funzione obiettivo

Si aggiuge la funzione obiettivo:

```
model += pulp.lpSum(c[i,j]*x[i,j,k] for (i,j,k) in
    arco_var)
```

In generale, è possibile anche cambiare la funzione obiettivo, ponendo per esempio all'interno della sommatoria il tempo di percorrenza invece dei costi

```
model += pulp.lpSum(t[i,j]*x[i,j,k] for (i,j,k) in
    arco_var)
```

3.2.3 Vincoli

Si aggiungono i vincoli precedentemente definiti:

```
# Vincolo di capacita'
for k in V:
  model += pulp.lpSum(d[i]*x[i,j,k] for i in P for j
     in N if i!=j) <= Q[k]

# Vincolo di unicita' dei punti di vendita
for i in P:
  model += pulp.lpSum(x[i,j,k] for j in N for k in V
     if (i,j) if i!=j) == 1

# Vincolo sulla partenza e arrivo del singolo veicolo
for k in V:
  model += pulp.lpSum(x[0,j,k] for j in P) == 1
  model += pulp.lpSum(x[i,n+1,k] for i in P) == 1

# Vincolo di coerenze partenza-arrivo
for k in V:
  model += pulp.lpSum(x[0,j,k] for j in P) == pulp.
     lpSum(x[i,n+1,k] for i in P)

# Vincolo sul flusso dei punti vendita
for j in P:
  for k in V:
    model += pulp.lpSum(x[i,j,k] for i in N if i!=j)
       == pulp.lpSum(x[j,i,k] for i in N if i!=j)

# Vincolo sui sottogiri
```

```
for r in range(2, len(P)+1):
  for S in itertools.combinations(P,r):
    for k in V:
      model += pulp.lpSum(x[i,j,k] for i in S for j
         in S if i!=j and (i,j)!=(n+1,0) and (i,j)
         !=(0,n+1)) <= len(S)-1

# Vincolo di inizio servizio
M = 10000000
for k in V:
  for j in N:
    for i in N:
      if i!=j and (i,j)!=(n+1,0) and (i,j)!=(0,n+1):
        model += l[j,k] >= (l[i,k]+s[i,k]+t[i,j]) - M
           *(1-x[i,j,k])


# Vincolo sulla finestra temporale
for (i, k) in arco_arrivo:
    model += l[i,k] >= a[i]
    model += l[i,k] <= b[i]

# Inizializzazione del deposito
for k in V:
  l[0,k] = a[0]
```

3.2.4 Esecuzione del modello

Possiamo quindi eseguire il modello:

```
model.solve()
print("Costo: ", pulp.value(model.objective))
```

Capitolo 4

Conclusion

Bibliografia