Progetto Metodi di Ottimizzazione della Logistica

di Marcello Bertoli, Micaela Verucchi e Samuele Zecchini

AA 2013/2014

Problema dell'Automated Guided Vehicle (AGV)

- Breve analisi letteratura
- Modello matematico ed implementazione Xpress
- Rilassamenti e lower bound
- Euristiche costruttive e/o di ricerca locale/metaeuristiche
- Predisposizione istanze di test
- Campagna di test

Breve analisi letteratura

Consegna:
"Dato un layout di un magazzino, definire algoritmi per l'ottimizzazione dei percorsi di veicoli automatici che effettuano il picking"

Un AGV è un veicolo a guida automatica, utilizzato nei magazzini o negli stabilimenti industriali per lo spostamento delle merci. Esiste quindi la necessità di un software che organizzi le reoute che tali veicoli devono compiere, in modo da ottimizzare i costi in tempo e spazio.

Inizialmente abbiamo un magazzino, un numero di muletti nMuletti e un numero di merci nMerci che dobbiamo prendere e riportare alla sorgente da cui partiamo.

Vogliamo :

- Ottimizzare i percorsi dei muletti, per cui ogni muletto deve fare il percorso di costo minore per raccogliere le merci.
- Minimizzare il numero di muletti, facendone partire solo quelli necessari per raccogliere le nMerci

Considerando che :

- I muletti hanno un peso massimo pesoMax che possono trasportare
- I muletti hanno un volume massimo volMax che possono trasportare

Passo 1
Abbiamo deciso di risolvere il problema partendo da un grafo iniziale che corrisponde al magazzino intero, ma, dato che sappiamo di partenza quali merci dobbiamo raccogliere, abbiamo poi pensato di ridurre il grafo iniziale molto grande ad un grafo più piccolo di cui i vertici sono la sorgente e le merci da raccogliere.
Per fare ciò abbiamo pensato di calcolare i cammini minimi tra la sorgente e le merci e tra le merci stesse, così da poter costruire un grafo molto ridotto, con archi riferiti al grafo originale, ma con solo i vertici che ci interessano.

Passo 2
Dopo aver ridotto il grafo abbiamo deciso di applicare il CVRP per risolvere completamente il problema. In tal modo troveremo i percorsi ottimi compiuti dal minor numero di muletti.

Modello matematico ed implementazione Xpress

Per risolvere i cammini minimi inizialmente abbiamo usato un modello matematico, ma l'abbiamo abbandonato perchè poco efficiente e sostituito con l'algoritmo esatto di Dijkstra.
La funzione Djikstra trova il minimo percorso tra la sorgente s e la destinazione d.

```
function Djikstra (s:integer, d :integer): integer

      forall(i in 1..nNodi) do
            distanze(i) := inf
            visitati(i) := false
      end-do
      distanze(s) := 0;

      while(true) do
            m := inf
            forall (i in 1..nNodi)
                  if(visitati(i)=false and distanze(i)<=m ) then
                        m := distanze(i)
                        j := i
                  end-if
            visitati(j) := true ;

            if( j = d or m = inf)then
                  returned:= distanze(d)
                  break;
            end-if

            forall (i in 1..nNodi)
                  if(pesiArchi(j,i) <> 0 and
                  distanze(i) > distanze(j)  +pesiArchi(j,i))then
                        distanze(i) := distanze(j) + pesiArchi(j,i)
                  end-if
      end-do
end-function
```

Nella procedura CamminiMinimi troviamo la minima distanza tra tutte le sorgenti s e le destinazioni d che ci servono, ovvero tutti i percorsi tra il punto di partenza e le merci da raccogliere e tutti quelli tra una merce e tutte le altre. In tal modo otterremo un grafo con tutti i vertici connessi tra loro. Nella procedura CamminiMinimi viene creata anche la matrice del grafo in questione.

```
procedure CamminiMinimi

      sorgenti :=1
      destinazioni :=2
      partenza:=2
      riga:=1
      colonna:=partenza
      forall(i in 1..round(nSD/2)) do

            !riempo la matrice finale

            peso := Djikstra (SD(sorgenti),SD(destinazioni))
            pesiArchiFinali(riga,colonna) := peso
            pesiArchiFinali(colonna,riga) := peso

            if(colonna = 11) then
                  riga := riga+1
                  partenza := partenza+1
                  colonna := partenza
            else
                  colonna:=colonna+1
            end-if

            !passo ai prossimi sorgente-destinazione
            sorgenti := sorgenti +2
            destinazioni := destinazioni +2

      end-do

end-procedure
```

Dopodiché abbiamo deciso di applicare il CVRP alla matrice appena ottenuta. Il Capacity VRP è un caso particolare di VRP, in cui si tiene conto della capacità dei muletti.

I dati del problema saranno :

- una matrice simmetrica dei pesi degli archi detta pesiArchiFinale(nNodiFinali)(nNodiFinali)
- un numero nMuletti di muletti disponibili
- un peso massimo pesoMax per ogni muletto
- un volume massimo volMax per ogni muletto
- un insieme V contenente tutti i vertici del grafo pesiArchiFinale
- un insieme V' contenente tutti i vertici del grafo pesiArchiFinale escluso il punto di partenza
- un insieme Φ contenente tutti gli $S : S \subseteq V'$, ovvero tutte le route possibili in base alle richieste

Il modello matematico a due indici per il CVRP è :

$$z = Min \sum_{i \in V} \sum_{j \in V} pesiArchiFinale(i, j)\, x(i, j)$$

$$\sum_{i \in V} x(i, j) = 1 \; \forall \, j \in V'$$

$$\sum_{j \in V} x(i, j) = 1 \; \forall \, i \in V'$$

$$\sum_{i \in V} x(i, 0) \leq nMuletti$$

$$\sum_{j \in V} x(0, j) \leq nMuletti$$

$$\sum_{i \in \bar{S}} \sum_{j \in S} x(i, j) \geq r(S) \in \Phi$$

$$x(i, j) \in 0,1 \, , \forall \, i, j \in V$$

dove r(S) è il numero di veicoli necessari per servire i clienti di S. Il valore esatto sarebbe quello del BPP associato, ma l'abbiamo approssimato a

$$r(S) = Max(\lceil \frac{peso(S)}{pesoMax} \rceil, \lceil \frac{vol(S)}{volMax} \rceil)$$

Nella procedura generaS generiamo tutti i Capacity Cut Constraints: creiamo tutti i possibili insiemi S, che sono 2^{nMerci} attraverso un algoritmo che da tutte le combinazioni possibili senza ripetizioni in modo ricorsivo. Tale funzione ci fa capire che il modello matematico non può essere usato per qualsiasi numero di merci, poiché dato che gli insiemi S sono di numero esponenziale, è necessario che nMerci sia un numero basso per avere tempi accettabili.

Per esempio con 10 merci, il modello impiega circa 3 secondi, ma già con 15 ne impiega circa 68.

```
procedure generaS(S: set of integer, sA : set of integer, k: integer, n:
integer, inizio:integer)
if(k = 0)then
      forall(j in sA)do

            forall(i in sA) do
          volS := volS + vMerci(i-1)
            pesoS := pesoS + pMerci (i-1)
         end-do

         rV:= ceil(volS / volMax)
         rP:= ceil(pesoS / pesoMax)

         Segn := Ver - sA
         r:= maxlist (rV,rP)

         sum(i1 in Segn, j1 in sA) x(i1,j1) >= r

         volS :=0
         pesoS := 0
      end-do

      sA-={sA(getsize(sA))}

      else
            i:=inizio

            while (i<=n) do
                  sA+={S(i)}
                  generaS(S,sA,k-1,n,i+1)
                  i:=i+1
            end-do

            if(getsize(sA) <> 0)then
            sA-={sA(getsize(sA))}
            end-if
end-if

end-procedure
```

Nella procedura VRP inseriamo tutti i vincoli del problema matematico e minimizziamo la z.

```
procedure VRP

      forall (i in 1..nMerci+1) Ver += {i}

      forall (i in 1..nMerci+1) pesiArchiFinali(i,i) := inf

      forall(i in 1..nMerci+1, j in 1..nMerci+1 )
                x(i,j) is_binary

      forall (j in 1..nMerci+1 | j<>1)
            sum(i in 1..nMerci+1) x(i,j) =1

      forall (i in 1..nMerci+1 | i<>1)
            sum(j in 1..nMerci+1) x(i,j) =1

      sum(i in 1..nMerci+1) x(i,1) <= nMuletti
      sum(j in 1..nMerci+1) x(1,j) <= nMuletti

      volS :=0
      pesoS := 0

      forall(i in 1..nMerci)
            S+={i+1}
      forall (k in 1..nMerci)
            generaS(S,sA,k,nMerci,1)

      z := sum(i1 in 1..nMerci+1,j1 in 1..nMerci+1)
            pesiArchiFinali(i1,j1)*x(i1,j1)

      minimize(z)

end-procedure
```

Rilassamenti e lower bound

R si dice problema rilassato rispetto a P se valgono le seguenti condizioni:
(a) $F(P) \subseteq F(R)$
(b) $\Phi(x) \leq f(x) \ \forall x \in F(P)$

Rilassamento per eliminazione (b-matching)

Il rilassamento per eliminazione consiste nel definire un problema rilassato R nel quale alcuni vincoli sono stati cancellati.

$$z = Min \sum_{i \in V} \sum_{j \in V} pesiArchiFinale(i,j)\, x(i,j)$$
$$\sum_{i \in V} x(i,j) = 1 \ \forall j \in V'$$
$$\sum_{j \in V} x(i,j) = 1 \ \forall i \in V'$$
$$\sum_{i \in V} x(i,0) \leq nMuletti$$
$$\sum_{j \in V} x(0,j) \leq nMuletti$$
$$x(i,j) \in 0,1, \forall i,j \in V$$

Nel nostro caso abbiamo eliminato tale vincolo

$$\sum_{i \in \bar{S}} \sum_{j \in S} x(i,j) \geq r(S) \in \Phi$$

Poiché i problemi P e R hanno la stessa funzione obiettivo siamo certi che la condizione (b) sia verificata. Inoltre, è banale verificare che $F(P) \subseteq F(R)$ in quanto F(R) è ottenuto da F(P) rimuovendo alcuni vincoli. In generale, la soluzione rilassata non e ammissibile per il problema originale quindi il rilassamento fornisce solo un lower bound sul valore della soluzione ottima .
Inoltre nella soluzione fornita potrebbero esserci cicli disconnessi dal deposito.

Euristiche costruttive e/o di ricerca locale/metaeuristiche

Come euristica abbiamo adottato l'algoritmo di Crarke e Wright, in sostituzione del modello matematico esatto del CVRP.

Funzionamento dell'algoritmo :

1. Si costruiscono nMerci route del tipo $(0,i,0), \forall i \in V'$;
2. Si calcola il saving $s(i,j), \forall i,j \in V', i \neq j$, dove il saving del cliente i rispetto al cliente j è definito come $s(i,j) = c_{i0} - c_{ij} + c_{0j}$
3. Costruiamo una lista $IndexList\{(i_1, j_1),(i_2, j_2)..\} tale\ che\ s(i_1, j_1) \geq s(i_2, j_2) \geq ... \geq s(i_n, j_n)$;
4. Estraiamo il primo elemento (i_k, j_k) della lista IndexList e controlliamo se rispetta i vincoli di capacità e se è ammissibile
5. Selezioniamo il prossimo elemento in ordine e ripetiamo il punto 4 finché la lista non è stata completamente esaminata

La funzione calcSavings calcola i saving per ogni coppia (i,j) , in tal modo : $s(i,j) = c_{i0} - c_{ij} + c_{0j}$

```
function calcSavings(i: integer,j: integer) : integer

      s := pesiArchiFinali(i,1) - pesiArchiFinali(i,j) +
pesiArchiFinali(1,j)
      returned:= s

end-function
```

La funzione cerca controlla se è possibile la fusione tra due route.

```
function cerca(i: integer,j: integer) : boolean

      returned := true
      if((route(i,3) = 1 and route(j,1) = 1)) then
            returned := false
      end-if

end-function
```

La procedura ClarkWright implementa l'algoritmo costruttivo di Clark & Wright per il VRP nella sua versione parallela.
L'algoritmo e' uno dei più performanti in termini di tempo computazionale, rispetto agli altri euristici per il VRP, ma la soluzione cosi ottenuta non sempre è la migliore approssimazione dell'ottimo.

1. La procedura crea "nMerci" route del tipo (0,i,0) per ogni i appartenente alle merci da raggiungere. Le strade sono create nella matrice "route" la quale ha altre 2 colonne, dove rispettivamente vengono inseriti i pesi e i volumi della merce raggiunta da una certa route.
2. Si crea una matrice dei risparmi, calcolati tramite la funzione calcSavings. Tale matrice sarà sicuramente simmetrica, quindi possiamo limitarci a utilizzare la parte triangolare superiore.
3. creiamo la lista IndexList (implementata attraverso un matrice di 2 colonne e (nNodiFinali*nNodiFinali/2)-nNodiFinali righe, cioè il numero di elementi nella parte triangolare superiore di una matrice) contenente gli indici dei savings ordinati per valore decrescente .
4. effettuiamo la fusione delle route, finché la IndexList non è terminata, in modo tale che le nuove route siano ammissibili cioè che:

   – rispettino i vincoli di capacita (sia volume che peso)
   – non modifichino soluzioni intermedie corrette e ammissibili(prima e terza colonna di route diverse da 1). Tale punto viene risolto attraverso la funzione cerca, la quale prende in ingresso i due indici da fondere per creare la nuova route e controlla se la fusione è realmente fattibile, cioè che nessuno dei due indici faccia parte di un altra route intermedia . La fusione è possibile quando una route ha come punto di partenza il deposito, mentre l'altra lo ha come punto di arrivo. Cerca ritorna falso se la fusione può essere fatta, vero altrimenti.

   Nel caso in cui le condizioni di merge siano verificate vengono aggiornati :
   – i valori della route per segnare il tragitto percorso
   – pesi e volumi delle route sommando quelli delle route di cui si è effettuata la fusione.

Infine calcoliamo il peso totale dei cammini e il numero di muletti impiegati.

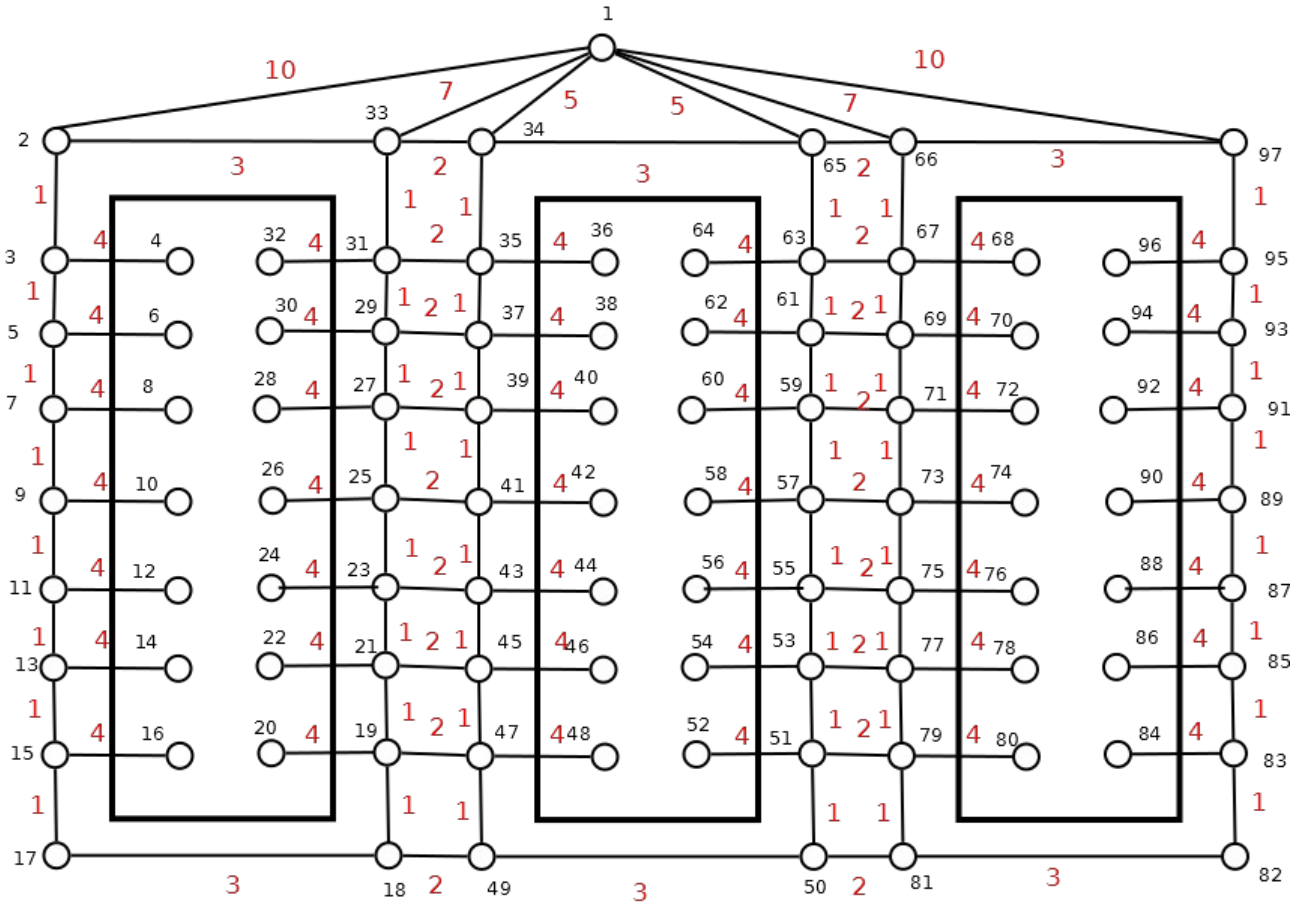
```
procedure ClarkWright

      forall(i in 1..nMerci) do
            forall(j in 1..5) do
                  if(j = 2) then
                        route(i,j) := i + 1
                  else if (j = 4 ) then
                        route(i,j) := pMerci(i)
                        else if (j = 5) then
                              route(i,j) := vMerci(i)
                              else
                                    route(i,j) := 1
                              end-if
                        end-if
                  end-if
            end-do
      end-do




      h:=1
```

```
forall(i in 2..nNodiFinali) do
      forall(j in 2..nNodiFinali)
            if(i <> j) then
                  savings(i,j):=calcSavings(i,j)
            end-if
end-do

ia:=0
ja:=0
mapp := 0
forall(it in 1..(round((nNodiFinali*nNodiFinali/2))-nNodiFinali))do
      mx:=0
      forall(i in 1..nNodiFinali)
            forall(j in 1..nNodiFinali)
                  if(j>=i and savings(i,j) >= mx) then
                        mx := savings(i,j)
                        ia := i;
                        ja := j
                  end-if
      IndexList(it,1) := ia
      IndexList(it,2) := ja
      if(mx = 0) then
            savings(ia,ja) := -1
      else
            savings(ia,ja) := -savings(ia,ja)
      end-if
      mapp := mx
end-do

forall(i in 1..(round((nMerci*nMerci/2))-nMerci)) do
      if(route(IndexList(i,1)-1,4) + route(IndexList(i,2)-1,4) <=
            pesoMax and route(IndexList(i,1)-1,5) +
            route(IndexList(i,2)-1,5) <= volMax and
            cerca(IndexList(i,1)-1,IndexList(i,2)-1) = false ) then

            route(IndexList(i,1)-1,3) := IndexList(i,2)
            route(IndexList(i,2)-1,1) := IndexList(i,1)
            route(IndexList(i,1)-1,4) := route(IndexList(i,1)-1,4) +
                  route(IndexList(i,2)-1,4)
            route(IndexList(i,2)-1,4) := route(IndexList(i,2)-1,4) +
                  route(IndexList(i,1)-1,4)
            route(IndexList(i,1)-1,5) := route(IndexList(i,1)-1,5) +
                  route(IndexList(i,2)-1,5)
            route(IndexList(i,2)-1,5) := route(IndexList(i,2)-1,5) +
                  route(IndexList(i,1)-1,5)
      end-if
end-do

pesoEuristico:=0
forall(i in 1..nMerci) do
            pesoEuristico:= pesoEuristico +
                  pesiArchiFinali(route(i,2),route(i,3))
end-do




forall(i in 1..nMerci) do
```

```
      if(route(i,1) = 1) then
            pesoEuristico:= pesoEuristico +
            pesiArchiFinali(route(i,1),route(i,2))
      end-if
end-do

mulettiEuristico :=0
forall(i in 1..nMerci) do
      if(route(i,3) = 1)then
            mulettiEuristico := mulettiEuristico +1
      end-if
end-do
```

Predisposizione istanze di test

Il magazzino utilizzato è il seguente, dove le merci sono quelle dentro i rettangoli neri e la sorgente è il vertice 1.

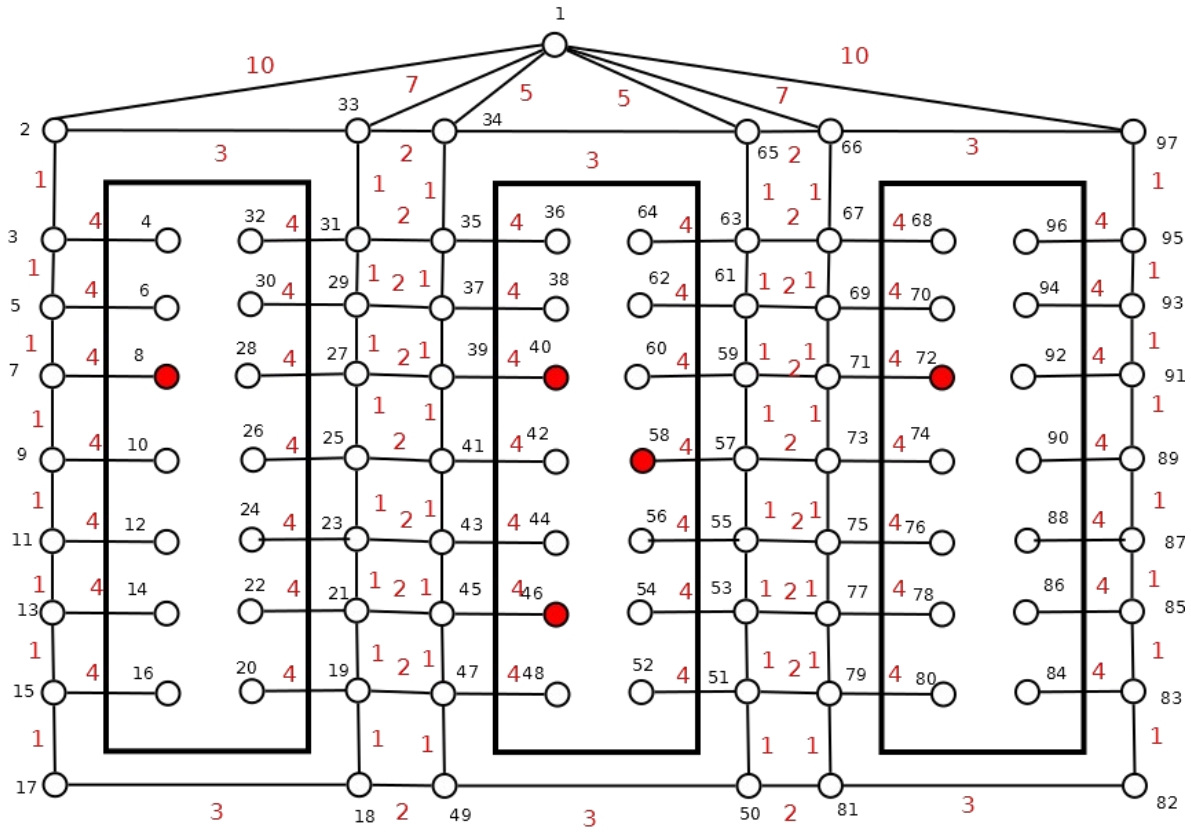


In questo caso si suppone che per ogni merce il tempo di carico sia lo stesso (nell'esempio l'arco ha peso 4), ma si può tranquillamente cambiare e dare un valore diverso ad ogni merce.
Il magazzino è memorizzato con una matrice delle adiacenze pesiArchi(nNodi, nNodi), nel nostro caso 97x97, dove l'elemento (i)(j) rappresenta il costo dell'arco (i)(j).
Dato che sappiamo a priori le merci da prendere (che al massimo sono 42) abbiamo creato un vettore SD nel quale inseriamo tutte le coppie (sorgente, destinazione) che utilizzeremo in Dijsktra per creare il sottografo da utilizzare nel CVRP.

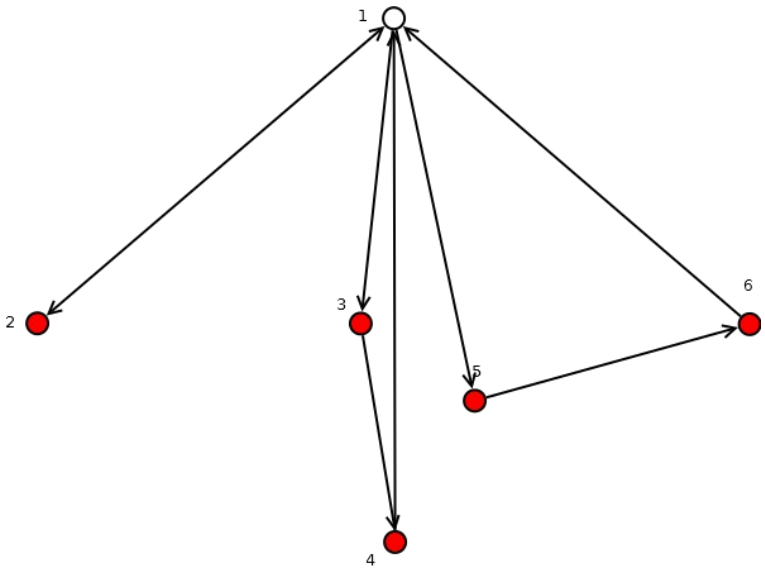
Campagna di test

TEST 1.1
Test con nMerci =5
PesoMax = 100
volMax = 1000

pMerci: [92 52 31 52 5]
vMerci: [400 10 235 220 700]



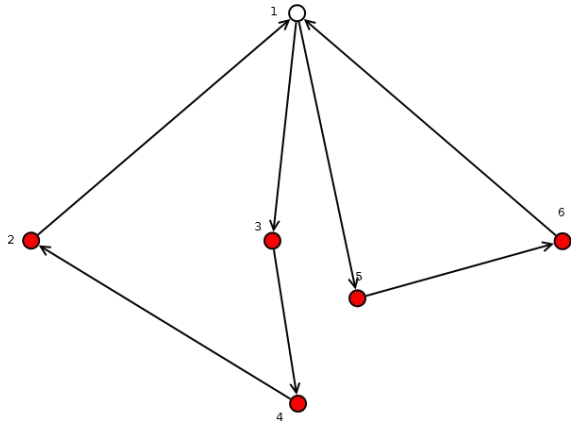
Risultato del modello :
Peso cammino minimo modello: 110
Muletti usati modello: 3
Route modello :
i: 1 j: 2 i: 2 j: 1
i: 1 j: 3 i: 3 j: 4 i: 4 j: 1
i: 1 j: 5 i: 5 j: 6 i: 6 j: 1



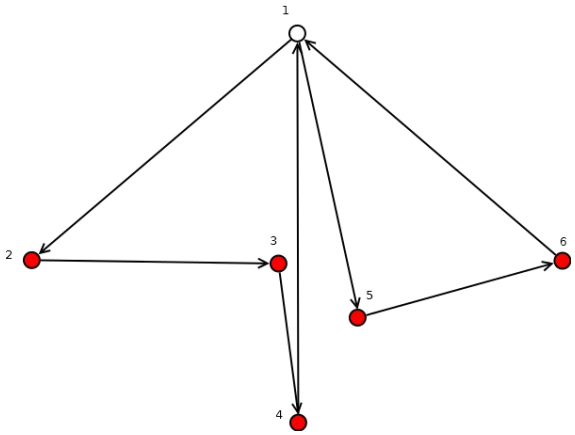
Peso cammino minimo euristico : 110
Muletti usati euristico : 3
Route euristico:
1 2 1
1 3 4 3 4 1
1 5 6 5 6 1

TEST 1.2

PesoMax = 1000
volMax = 1000



Risultato del modello :
Peso cammino minimo modello : 98
Muletti usati modello : 2
Route modello
i: 1 j: 3 i: 3 j: 4 i: 4 j: 2 i: 2 j: 1
i: 1 j: 5 i: 5 j: 6 i: 6 j: 1

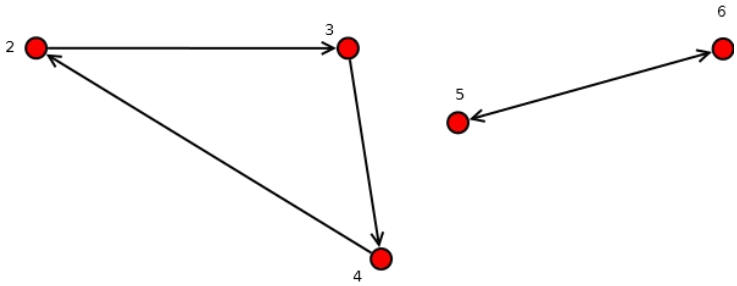


Peso cammino minimo euristico : 104
Muletti usati euristico : 2
Route euristico:
1 2 3 2 3 4 3 4 1
1 5 6 5 6 1

Applichiamo i Lower Bound al VRP

Peso cammino minimo : 76
Muletti usati : 2

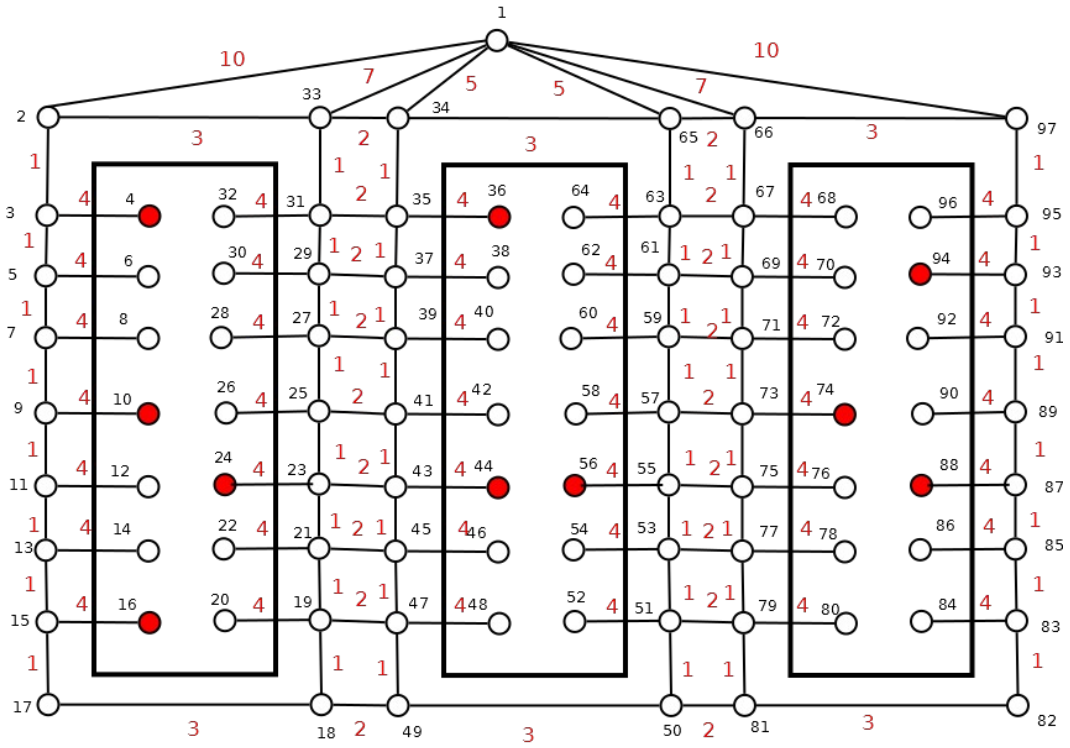
Route modello :
i: 2 j: 3 i: 3 j: 4 i: 4 j: 2
i: 5 j: 6 i: 6 j: 5



I risultati sono chiaramente non accettabili poiché non partono dalla sorgente.

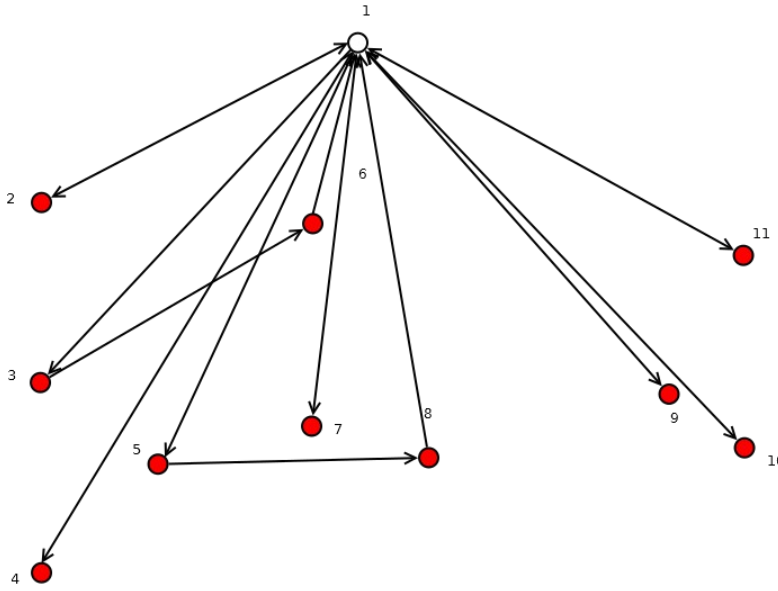
TEST 2.1
Test con nMerci = 10
pesoMax = 100
volMax = 1000

pMerci: [86 20 87 39 78 85 30 38 92 91]
vMerci: [553 630 426 565 264 604 424 605 262 72]



Risultato del modello :
Peso cammino minimo modello : 301
Muletti usati modello: 8
Route modello:

i: 1 j: 2	i: 2 j: 1	
i: 1 j: 3	i: 3 j: 6	i: 6 j: 1
i: 1 j: 4	i: 4 j: 1	
i: 1 j: 5	i: 5 j: 8	i: 8 j: 1
i: 1 j: 7	i: 7 j: 1	
i: 1 j: 9	i: 9 j: 1	
i: 1 j: 10	i: 10 j: 1	
i: 1 j: 11	i: 11 j: 1	



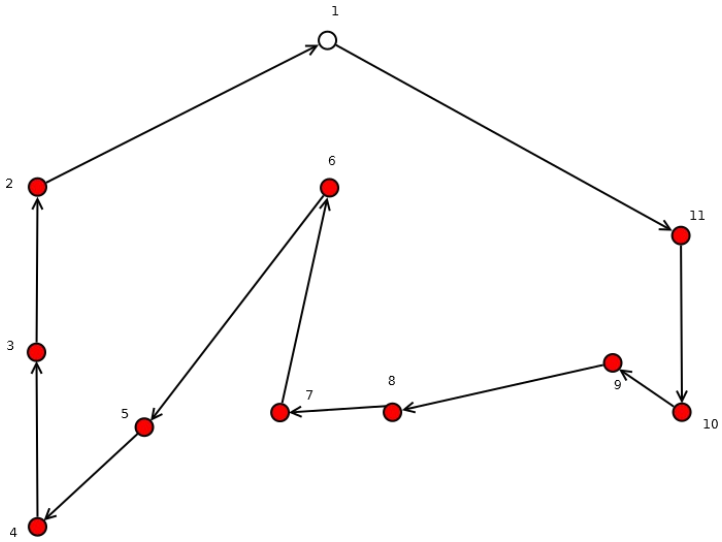
Peso cammino minimo euristico : 301
Muletti usati euristico : 8
Route euristico:

1 2 1	
1 3 6	3 6 1
1 4 1	
1 5 8	5 8 1
1 7 1	
1 9 1	
1 10 1	
1 11 1	

TEST 2.2
PesoMax = 800
VolMax = 100000

Risultato del modello :
Peso cammino minimo modello: 151
Muletti usati modello: 1
Route modello:

i: 1 j: 11	i: 11 j: 10	i: 10 j: 9
i: 9 j: 8	i: 8 j: 7	i: 7 j: 6
i: 6 j: 5	i: 5 j: 4	i: 4 j: 3
i: 3 j: 2	i: 2 j: 1	



Peso cammino minimo euristico : 155
Muletti usati euristico : 2
Route euristico:

1 2 3	2 3 4	3 4 5
4 5 7	5 7 8	7 8 9
8 9 10	9 10 11	10 11 1

1 6 1