Capacitated Vehicle Routing Problem

Civiero Vittoria 302839
Colliani Felice Paolo 302736
Panuccio Ilaria 302677

Luglio 2022

Introduzione

Al fine di risolvere il problema di *capacitated vehicle routing* abbiamo prima di tutto implementato un metodo costruttivo, basato sull'algoritmo di *Clarke-Wright*, e successivamente uno iterativo, il cui funzionamento si basa sull'approccio *2-opt*.
Il procedimento costruttivo ci ha permesso di ottenere una prima soluzione che abbiamo migliorato tramite il metodo iterativo.

Metodo Costruttivo

In questa prima parte, abbiamo scelto di implementare l'algoritmo di *Clarke-Wright* [1], tenendo presente che ad ogni cliente corrisponde una precisa posizione ed è sempre possibile calcolare la distanza euclidea tra loro.
L'algoritmo si basa sul criterio del *saving*, la cui idea è la seguente: trasformare due percorsi diversi in uno solo, riducendo la distanza percorsa e rispettando il vincolo di capacità del veicolo. Pertanto, se la tratta iniziale per tale veicolo fosse:

$$deposito \rightarrow cliente(i) \rightarrow deposito \quad e \quad deposito \rightarrow cliente(j) \rightarrow deposito$$

applicando il metodo costruttivo diventerebbe:

$$deposito \rightarrow cliente(i) \rightarrow cliente(j) \rightarrow deposito$$

e la distanza risparmiata, chiamata appunto *saving*, sarebbe:

$$s_{ij} = d_{i,\, deposito} + d_{deposito,\, j} - d_{ij}$$

Nella fase di inizializzazione, abbiamo definito le seguenti variabili:

- `nnode` è il numero totale dei clienti;
- `strada` è un *cell array* di dimensione uguale al numero di clienti, in cui andremo a salvare tutte le strade che vengono create dall'algoritmo;

[1] Distribution Logistic VRP, Brandimarte & Zotteri; esempio 8.4

Implementazione dell'algoritmo di Clarke-Wright

L'algoritmo di *Clarke-Wright* si arresta solo quando a tutti i clienti è stata assegnata una strada.

Prima di tutto, analizziamo la matrice dei saving e memorizziamo il suo valore massimo all'interno della variabile M, successivamente individuiamone i relativi indici di riga e colonna (i,j). Con i e j indicheremo i due clienti corrispondenti ad un particolare valore di saving.

La variabile `nconflitti` identifica il numero di volte in cui compare lo stesso valore di saving all'interno della matrice. Questa ci permette di considerare tutti i saving in ordine decrescente di valore. Infatti, con questo procedimento, potremo valutare tutte le diverse coppie di clienti aventi lo stesso saving.

Considerando tutti i possibili conflitti, assegniamo le strade ai clienti seguendo 3 differenti criteri:

1. Se entrambi i clienti i e j non sono stati assegnati ad alcuna strada e se la somma delle loro domande non supera la capacità massima, allora entrambi i clienti vengono assegnati ad una nuova strada;

2. Se al cliente i è stata assegnata una strada e la capacità derivante dall'aggiunta del cliente j non supera la capacità massima allora assegno la stessa strada di i al cliente j;

3. Se al cliente j è stata già assegnata una strada e la capacità derivante dall'aggiunta del cliente i non supera la capacità massima, proseguo con un assegnamento analogo a quello nel punto precedente.

Infine calcoliamo il costo di ciascuna strada sommando le distanze tra coppie di clienti consecutivi. Pertanto, il costo totale, che confronteremo con il metodo iterativo, consiste nella somma dei costi di tutte le strade.

Metodo Iterativo

In questa seconda parte abbiamo implementato l'approccio `2-opt`, il quale ci permette di ottenere un nuovo percorso eliminando e sostituendo due archi non consecutivi.
Dal metodo costruttivo abbiamo ricavato alcune informazioni: le strade assegnate a ciascun cliente (`strada`), il numero di strade totali (`nstrada`).

Applicando il procedimento iterativo miglioriamo singolarmente ciascuna strada, ignorando il vincolo di capacità, poiché già soddisfatto. Con questo metodo ci limiteremo dunque a risolvere un TSP (Traveling Salesperson Problem) su ciascun percorso.

Implementazione dell'approccio 2-opt

Eseguiamo il 2-opt per ogni strada. Per prima cosa inizializziamo le variabili `minimo_old` e `minimo_new`, le quali saranno utili per decidere se effettuare o meno lo scambio tra gli archi.

Successivamente, valutiamo la distanza iniziale data dalla somma dei costi di due archi non consecutivi e poi calcoliamo la distanza finale invertendo due nodi alla volta. Tra tutte le inversioni possibili consideriamo quella in cui il nodo destinazione del primo arco viene scambiato con l'origine del secondo arco. Ad esempio, se avessimo i due archi non consecutivi (1,2) e (3,4), allora il 2-opt controllerebbe la somma di (1,3) e (2,4) (`distanza_finale`), e se fosse minore del costo di partenza (`distanza_iniziale`) salverebbe il tour composto da questi nuovi archi. A questo punto potremmo definire `arco_new1` e `arco_new2`, i quali collegherebbero, rispettivamente, il cliente 1 con il 3 e il cliente 2 con il 4.

Il tour ottimale sarà quindi il risultato di diversi *flip* applicati opportunamente alla strada originale.

In ultimo, esattamente come nel caso del metodo costruttivo, calcoliamo il costo totale (`costo_tot_new`).

Applicazione e risultati

Per la visualizzazione dei risultati sono stati utilizzati due dataset presi al seguente link http://vrp.galgos.inf.puc-rio.br/index.php/en/ :

- Set A (Augerat, 1995), instance A-n32-k5, da 32 nodi;
- Set A (Augerat, 1995), instance A-n80-k10, da 80 nodi.

entrambi aventi capacità massima pari a 100. All'interno di ciascun dataset possiamo trovare le variabili di seguito riportate:

- `cmax`: capacità identica per tutti i veicoli;
- `distance`: matrice simmetrica delle distanze;
- `demand`: vettore colonna con i valori di domanda per ogni cliente;
- `coord`: matrice con le coordinate euclidee di ogni cliente.

I seguenti risultati sono stati ottenuti utilizzando il codice MATLAB che si trova in appendice.
Per il dataset con 32 nodi, al termine dell'algoritmo, abbiamo ottenuto 5 strade differenti:

1. [1 15 19 9 12 5 29 25 1]
2. [1 30 10 23 16 11 26 6 21 1]
3. [1 7 3 4 24 28 1]
4. [1 18 20 32 22 14 8 27 1]
5. [1 2 13 17 31 1]

Nella seguente tabella vengono riportati i costi risultanti da entrambi i metodi:

	costo metodo costruttivo	**costo metodo iterativo**
strada 1	231.1290	210.0549
strada 2	256.7940	238.1507
strada 3	179.4865	176.1747
strada 4	160.5585	160.5585
strada 5	78.9902	78.9902
totale	**906.96**	**863.93**

Mentre per il dataset da 80 nodi, abbiamo ottenuto 10 strade diverse:

1. [1 34 16 56 57 70 66 36 27 48 20 58 21 76 32 1]
2. [1 25 7 31 79 9 17 44 69 62 28 45 1]
3. [1 24 3 38 35 12 64 11 1]
4. [1 53 29 80 49 19 15 72 1]
5. [1 65 47 42 26 60 18 6 78 52 1]
6. [1 10 55 73 71 39 4 54 1]
7. [1 77 51 33 5 23 46 59 1]
8. [1 40 61 30 75 13 63 1]
9. [1 14 43 37 68 67 74 50 1]
10. [1 2 8 22 41 1]

Nella seguente tabella vengono riportati i costi risultanti da entrambi i metodi:

	costo metodo costruttivo	**costo metodo iterativo**
strada 1	413.7594	297.7286
strada 2	285.8253	249.4652
strada 3	210.0161	191.3636
strada 4	199.7242	192.4229
strada 5	226.3642	208.3812
strada 6	260.2485	260.2485
strada 7	208.3472	199.1938
strada 8	169.9954	156.0194
strada 9	111.0145	98.6279
strada 10	85.7610	85.7610
totale	**2171.06**	**1939.21**

Come si evince dalle tabelle il costo totale diminuisce dopo aver applicato il metodo iterativo. I costi riportati sul sito corrispondono a 784 per il dataset da 32 nodi e a 1763 per quello da 80 nodi. Rispetto ai risultati da noi ottenuti si evidenzia un 10% in meno. Inoltre, è possibile che i costi ottenuti dai due metodi per alcune strade siano uguali. In questo caso il metodo costruttivo fornisce già la configurazione ottimale, ciò accade per le strade 2 e 3 del primo dataset e per le strade 6 e 10 del secondo dataset.

Appendice

Metodo costruttivo

```
% scelta del dataset in base al numero di nodi
% all'intero ci sono: la matrice delle distanze (distance)
%                     il vettore delle richieste (demand)
%                     le coordinate sulla mappa (coord)
%                     la capacita' massima (cmax)

% load('dataset80.mat')  % dataset con 80 nodi
load('dataset32.mat')    % dateset con 32 nodi

nnode = size(distance);
nnode = nnode(1); % numero di nodi sulla mappa
strada = cell(nnode,1); % lista delle strade che verranno create, ...
    inizializziamo con il numero massimo di nodi e in seguito ...
    capiremo quante ne verranno effettivamente create
assegnamento = zeros(nnode,1); % vettore che indica a quale strada ...
    e' stato assegnato il nodo i-esimo

for i=1:nnode % ciclo che dice che ogni strada inizia con il nodo 1 ...
    (deposito)
    strada{i}=1;
end

% implementazione del METODO COSTRUTTIVO dei saving

saving = zeros(nnode,nnode); % matrice che conterra' i "saving" dei nodi
%... ed essendo simmetrica si puo' fare triangolare
for i=2:nnode
    for j = i+1:nnode
        saving(i,j)=distance(1,i)+distance(1,j)-distance(i,j);
    end
```

```
end
nstrada = 0; % inizializzazione
m = 10e5; % indica il saving piu' grande trovato finora

% ciclo while che assegna tutti i nodi a una determinata strada in ...
    base a 3 diversi criteri di assegnamento, va avanti finche' tutte ...
    le strade non sono state assegnate, controllando la variabile ...
    "assegnamento"

while sum(assegnamento≠0)<nnode-1 % -1 perche' il deposito non va ...
    assegnato
    M =  max(saving(saving<m)); % trovo il massimo saving minore del ...
        precedente
    [i,j]=find(saving == M);     % trovo i nodi che hanno quell' ...
        esatto saving
    nconflitti = size(i,1);      % ATTENZIONE: i nodi con lo stesso ...
        saving posso essere di piu'! Tengo conto del loro numero con ...
        questa variabile
    for t = 1:nconflitti
        % primo criterio di assegnamento: se nessuno dei due nodi i ...
            e j del saving e' stata assegnato e il vincolo di ...
            capacita' e' rispettato, allora li assegno entrambi a una ...
            nuova strada (nstrada+1)
        if assegnamento(i(t))==0 && assegnamento(j(t))==0
            if demand(i(t)) + demand(j(t)) ≤ cmax
                assegnamento(i(t)) = nstrada+1;
                assegnamento(j(t)) = nstrada+1;
                strada{assegnamento(i(t))} = ...
                    [strada{assegnamento(i(t))},i(t),j(t)];
                capacity(assegnamento(i(t))) = ...
                    demand(i(t))+demand(j(t)); % aggiornando la ...
                    capacita' residua della strada
                nstrada = nstrada+1;
            end
```

```
        % secondo criterio: se il nodo di arrivo j non e' stato ...
            ancora assegnato, lo inserisco nella strada del nodo i, ...
            nel caso in cui il vincolo di capacita' sia rispettato
        elseif  assegnamento(j(t))==0
            if capacity(assegnamento(i(t))) + demand(j(t)) ≤ cmax
                strada{assegnamento(i(t))} = ...
                    [strada{assegnamento(i(t))},j(t)];
                capacity(assegnamento(i(t))) = ...
                    capacity(assegnamento(i(t))) + demand(j(t));
                assegnamento(j(t)) = assegnamento(i(t));
            end
        % terzo criterio: uguale al secondo criterio ma fatto per il ...
            nodo di partenza i
        elseif assegnamento(i(t))==0
            if capacity(assegnamento(j(t))) + demand(i(t)) ≤ cmax
                strada{assegnamento(j(t))} = ...
                    [strada{assegnamento(j(t))},i(t)];
                capacity(assegnamento(j(t))) = ...
                    capacity(assegnamento(j(t))) + demand(i(t));
                assegnamento(i(t)) = assegnamento(j(t));
            end
        end
    end
    m = M; % aggiorno il nuovo valore di saving
end
% calcolo il costo finale con il metodo costruttivo dei saving
costo = zeros(nstrada,1);
for i=1:nstrada
    strada{i}=[strada{i},1]; % aggiungo il deposito come strada finale
    for j = 1: size(strada{i},2)-1
        costo(i) = costo(i) + distance(strada{i}(j),strada{i}(j+1)); ...
            % sommo i costi di ogni arco
    end
end
costo_tot = sum(costo);
```

Metodo Iterativo

```
% implementazione del METODO ITERATIVO 2 opt
% controlliamo se nuove coppie di nodi nella stessa strada posso ...
    portare a una diminuzione del costo totale

for j = 1:nstrada % per ogni strada creata
minimo_old = 0;
minimo_new = 20;
    while minimo_new ≠ 0
        minimo_old = 0;
        minimo_new = 20;
        lunghezza = length(strada{j})-1; % numero di nodi visitati ...
            dalla strada j
        % salvo gli archi (coppie di nodi successivi) della strada j
        for i = 1: lunghezza
            arco{j}{i} = [strada{j}(i) strada{j}(i+1)];
        end
        % inizio confronto di nuovi archi all'interno della strada j
        for i = 1:(lunghezza-2)
            if i == 1
                last_check = lunghezza-1;
            else
                last_check = lunghezza;
            end
             for k = (i+2):last_check
                % valuto la distanza iniziale, sommando i pesi di ...
                    due archi non consecutivi, e poi la distanza ...
                    finale che ottengo facendo uno scambio dei nodi
                distanza_iniziale = ...
                    distance(arco{j}{i}(1),arco{j}{i}(2)) + ...
                    distance(arco{j}{k}(1),arco{j}{k}(2));
                distanza_finale = ...
                    distance(arco{j}{i}(1),arco{j}{k}(1)) + ...
                    distance(arco{j}{i}(2),arco{j}{k}(2));
```

```
                %controllo se ho migliorato la situazione
                minimo_new = min(minimo_old, ...
                    distanza_finale-distanza_iniziale);
                % se e' migliorata allora aggiorno gli archi
                if minimo_new ≠ minimo_old
                    minimo_old = minimo_new;
                    arco_old1 = [arco{j}{i}(1) arco{j}{i}(2)];
                    arco_old2 = [arco{j}{k}(1) arco{j}{k}(2)];
                    arco_new1 = [arco{j}{i}(1) arco{j}{k}(1)];
                    arco_new2 = [arco{j}{i}(2) arco{j}{k}(2)];
                end
             end
        end
        % aggiorno la strada con l'arco migliore trovato per ridurre ...
            il costo totale
        begin_flip = find(arco_new1(1) == strada{j})+1;
        end_flip = find(arco_new2(2) == strada{j})-1;
        strada{j}(begin_flip:end_flip) = ...
            flip(strada{j}(begin_flip:end_flip));
    end
end

% calcolo il costo finale ottenuto con il metodo iterativo 2 opt
costo_new = zeros(nstrada,1);
for i=1:nstrada
    for j = 1: size(strada{i},2)-1
        costo_new(i) = costo_new(i) + ...
            distance(strada{i}(j),strada{i}(j+1));
    end
end
costo_tot_new = sum(costo_new);
```