Implementazione e testing di semplici euristiche per il problema CVRP - Business Analytics

Giuseppe Intilla 297641, Francesco Pappone 299913

Agosto 2022

1 Algoritmi costruttivi

1.1 Clarke-Wright

Il primo algoritmo costruttivo implementato è il Clarke-Wright. Utilizza il criterio dei saving per unire le route in modo sequenziale. Definiamo il saving tra due nodi i e j come: $s(i,j) = c(i,0) + c(0,j) - c(i,j)$, dove c rappresenta il costo. A ogni iterazione cerchiamo i nodi che hanno il massimo saving e, se possibile, uniamo le due route che li contengono.

1.1.1 Implementazione

A livello operativo implementiamo in Matlab una classe route che gestisce le operazioni necessarie tramite diversi metodi: merge fra due route, aggiunta di un nodo, calcolo del costo e del budget occupato. La struttura dati base della classe route è un array numerico contenente i suoi nodi. Implementiamo, poi, la nostra funzione principale che prende in input la capacità massima, la matrice dei costi e il vettore di domande. Definiamo, inoltre, il vettore routes, che viene inizializzato con tutte le route contenenti i singoli nodi, e il vettore opt, inizialmente vuoto e che alla fine sarà composto dalle route ottimali. La parte centrale di tale funzione è costituita da due while loop annidati. Quello esterno itera finchè il vettore routes non è vuoto, quello interno invece si ferma quando viene effettuato un merge tra due route.

1.1.2 Pseudocodice

while routes≠null **do**
  $opt \leftarrow get\_opt(routes)$
  $delete(opt, routes)$
  while $k \neq 0$ **do**
    $i, j \leftarrow max(savings)$
    $savings(i,j) \leftarrow 0$
    $candidate \leftarrow merge(route(i), route(j))$

```
        if budget(candidate) ≤ capacity then
            delete([route(i), route(j)], routes)
            add(candidate, routes)
            k ← 0
        end if
    end while
end while
```

1.2 Variazione

Il limite dell'algoritmo appena descritto è il fatto che non possiamo controllare il numero di route generate. Per questo motivo implementiamo una variazione parallela del Clarke-Wright che forza l'algoritmo ad avere il numero voluto di route. Sia n il numero di route desiderato, scegliamo n nodi seed che useremo come seme per le nostre route. Il k-esimo seed viene scelto nel modo seguente:

$$s_k = \arg\max_i \min\{c(i,0), c(i,s_1), \ldots, c(i,s_{i-1})\} \tag{1}$$

Partiamo definendo delle route dai seed scelti e le estendiamo in modo sequenziale. Per ogni route aggiungiamo nodi usando il criterio dei savings finchè possibile senza sforare il vincolo di capacità.

1.2.1 Implementazione

Implementiamo adesso la nostra funzione principale che prende in input la capacità massima, il numero di veicoli, la matrice dei costi e il vettore di domande. Definiamo il vettore routes, che viene inizializzato con tutti i nodi non seed, e il vettore opt, inizializzato con i seed. La parte centrale della funzione è costituita da un for loop e un while loop annidati. Il for loop esterno itera sul numero di veicoli, il while loop interno invece si ferma quando la route in analisi non è più ottimizzabile a causa del vincolo di capacità.

1.2.2 Pseudocodice

```
for i ≤ n do
    while opt(i) ≠ is_opt(routes) do
        j ← max(savings[opt(i)])
        candidate ← add_node(opt(i), j)
        savings(j, opt(i)) ← 0
        if budget(candidate) ≤ capacity then
            delete(route(j), routes)
            add(candidate, opt)
            modify(savings)
        end if
    end while
end for
```

1.3 Esperimenti computazionali

Per validare gli algoritmi proposti facciamo dei test su alcune delle istanze del portale CVRPLIB. I risultati sono riassunti nella tabella seguente.

Istance	Customers	Capacity	Vehicles	Optimal	Clarke-Wright	Parallel
A-n32-k5	31	100	5	784	982	987
A-n80-k10	79	100	10	1763	2066	2253
M-n200-k17	199	200	17	1275	2244	1832
Golden_1	239	550	9	5623*	6641	6755

Tabella 1: Risultati dei due algoritmi applicati a 4 istanze di test.
* valore sub-ottimale

2 Algoritmi iterativi

2.1 Sweep clustering e 2-opt-based local search con simulated annealing

Il seguente metodo si basa sulla clusterizzazione attraverso il metodo sweep per la costruzione di una serie di problemi di TSP. Ognuno dei problemi di TSP, inizializzato con una route dipendente dall'ordine di ingresso nel cluster dei nodi, viene quindi affrontato attraverso una local search sulla 2-opt neighborhood di ogni singola route, con una ricerca sulla neighborhood effettuata attraverso simulated annealing.

2.2 Implementazione

La procedura è implementata associando ad ognuno degli step fondamentali dell'algoritmo una funzione. L'intero processo può essere svolto fornendo un'inizializzazione alle capacità, alla domanda e alle coordinate dei nodi ed inserendo tali informazioni come argomenti di una singola funzione.

2.2.1 Sweep clustering

Per costruire i singoli problemi di TSP su cui impiegare l'algoritmo di local search, produciamo una clusterizzazione dei nodi in funzione della loro posizione. In particolare calcoliamo, per ogni nodo, le coordinate polari. Viene quindi inizializzata una semiretta centrata nel deposito, di angolo θ_k, con $theta_0$ inizializzato casualmente, e con update $\Delta\theta_k = \theta_{k+1} - \theta_k = \alpha$ parametro scelto a priori. Se un nodo ha distanza dalla retta minore di un parametro δ, e se la capacità rimanente al cluster corrente è sufficiente, il nodo viene aggiunto al cluster. Altrimenti, si procede con il cluster successivo fino al loro esaurimento. E' utile notare come i cluster non esauriscano necessariamente la capacità a loro disposizione. Per permettere ai cluster di utilizzare più capacità possibile,

abbiamo implementato lo sweep in modo da ripetere un numero prefissato di round di acquisizione per ogni cluster, così da poter accedere a nodi lasciati indietro da precedenti acquisizioni.

2.2.2 Implementazione

L'algoritmo calcola, per prima cosa, le coordinate associate ai nodi e inizializza l'angolo della semiretta. Un ciclo for itera sui veicoli, a cui saranno associati univocamente dei cluster. Un ciclo while, all'interno del precedente, fa un update dell'angolo e contiene le condizioni per l'accettabilità di un nodo. Se un nodo viene accettato, viene sottratto dalla lista dei nodi accettabili e viene aggiornata la capacità residua del veicolo. Quando viene incontrato un nodo con capacità troppo elevata per l'acquisizione, il ciclo while interno viene terminato, e inizia quello associato al successivo veicolo. Una volta terminati i veicoli, l'operazione ricomincia per un numero fissato di round, mantenendo soltanto i nodi non ancora accettati. L'algoritmo di Sweep è organizzato come una funzione i cui argomenti sono: una matrice contenente le informazioni sui customers, l'informazione sulle capcità, lo stepsize dell'update α, la distanza minima per la collisione tra la retta e il nodo, il vettore delle distanze dei clienti dal deposito. La funzione viene chiamata all'inizio del procedimento. I cluster così prodotti vengono trasformati in tour di inizializzazione dei singoli TSP, il tour viene definito a partire dall'ordine di acquisizione da parte del cluster.

2.2.3 2-opt neigborhood

L'operazione di local-search sui singoli TSP viene effettuata su una neighborhood composta dai tour risultanti dall'applicazione di tutti i possibili 2-swap su quello corrente. In particolare, gli swap avvengono tra coppie di edge non adiacienti, e in modo che la feasibility della route risultante sia sempre conservata.

2.2.4 Implementazione

La costruzione della neighborhood tramite 2-opt è operata da una funzione il cui argomento è il tour di cui costruire la neighborhood. Per semplificare l'implementazione, i tour sono codificati come lista di nodi raggiunti, in modo che gli estremi della lista siano sempre il deposito (così da mantenere la feasibility). Per alcune operazioni, è più agevole una rappresentazione espansa, nella forma di lista di edge (coppie di nodi). Tale implementazione non è intesa per rappresentare un grafo direzionato, ma per ogni link è implicita l'esistenza dell'intera edge associata. La funzione produce una lista di coppie di edge non adiacienti. Per ogni elemento di questa lista, viene effettuato lo scambio in modo da preservare la feasibility. In particolare, sia $(v_1, \ldots, v_{k1}, \ldots, v_{k2}, \ldots, v_n)$ un tour, con $(k1, l1)$ e $(k2, l2)$ coppie di edges non adiacienti. Il 2-swap viene costruito invertendo l'ordine (questa inversione modifica anche la topologia del grafo, non solo il verso di un'ipotetico grafo direzionato costruito seguendo l'ordine dei no-

di) dei nodi tra v_{k1+1} e v_{k2}. L'output della funzione è una lista di routes (nella codifica compressa, ovvero come sola lista di nodi).

2.2.5 Simulated annealing

La ricerca sulla neighborhood così costruita avviene attraverso un criterio di simulated annealing: sia C_{old} il costo associato alla corrente configurazione. Estraiamo un tour uniformemente dalla neighborhood, con costo associato C_{new}. la probabilità p di accettare la proposta vale dunque

$$p = \min(1, \frac{\exp(-C_{new} - C_{old})}{T(k)})$$

dove $T(k) = 1 - \frac{k+1}{L}$, k counter delle iterazioni, L numero massimo di iterazioni (parametro scelto a priori). L'iterazione si ferma quando la differenza di costo tra una scelta e la successiva scende sotto una certa soglia λ, anche questa scelta a priori, oppure se si esaurisce il numero di iterazioni.

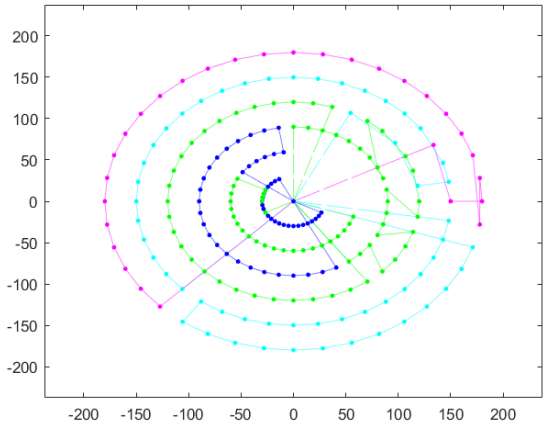
2.2.6 Implementazione

La funzione che implementa la local search viene chiamata successivamente al clustering e all'inizializzazione dei tour sui singoli cluster. Gli argomenti sono il tour iniziale, il limite di iterazioni, la threshold λ e la matrice delle distanze. Le operazione descritte nel paragrafo precedente avvengono all'interno di un ciclo while, che viene interrotto qualora si raggiunga il limite d'iterazioni o qualora si ottenga una soluzione molto vicina, rispetto al costo, a quella precedente. Anche qui, i tour sono rappresentati nella loro codifica più compressa, e vengono espansi (tramite un'apposita funzione) solo per il calcolo dei costi. Tutte le funzioni descritte fino ad ora vengono chiamate, nella giusta sequenza, da una funzione i cui argomenti sono le posizioni dei nodi, la domanda, il numero dei veicoli e la capacità per veicolo. All'interno di questa funzione vengono definiti tutti i parametri del problema, vengono prodotti ed inizializzati i cluster, e quindi migliorati attraverso il simulated annealing. Al termine, l'output della funzione è un cell array contenente, come elementi, i singoli tour, sotto forma di liste di nodi. Viene inoltre eseguito un plot delle route, con ogni route riprodotta con un colore random. Una seconda implementazione di questa funzione permette di inserire un cell array contenente soluzioni già prodotte con altri metodi, e di utilizzare la local search sui TSP così prodotti, migliorando le configurazioni con il simulated annealing.

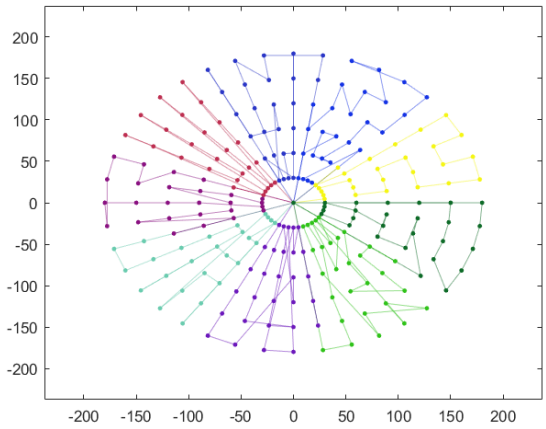
2.3 Esperimenti computazionali

Istance	Customers	Capacity	Vehicles	Optimal	Sweep init	CW init	Parallel init
A-n32-k5	31	100	5	784	1090	927	983
A-n80-k10	79	100	10	1763	2397	1960	2239
M-n200-k17	199	200	17	1275	2914	1777	1843
Golden_1	239	550	9	5623*	9739	6001	6732

Tabella 2: Risultati per gli esperimenti numerici di local search, rispettivamente, per l'inizializzazione tramite sweep-clustering, tramite soluzione del CW e della sua variante.
* valore sub-ottimale



(a) Soluzione trovata con inizializzazione da soluzione CW



(b) Soluzione trovata con inizializazzione Sweep

Figura 1: I plot rappresentano soluzioni al quarto esperimento proposto. I colori sono associati ai singoli veicoli.