CVRP Business Analytics

El Amrani Rachid
Buccelli Giorgia Rosalia

Luglio 2022

1 Introduzione

Nelle seguenti pagine verrà trattato il **capacitated vehicle routing problem (CVRP)**. In particolare il problema verrà affrontato prima con l'approccio della **programmazione lineare**, poi si cercherà una soluzione migliore con il **simulated annealing**. Verranno mostrati dei plot per verificare i risultati e inoltre verranno comparate le performance. I codici utilizzati saranno in appendice

2 Approccio con programmazione lineare

Prima di tutto, un po' di notazioni, chiamiamo

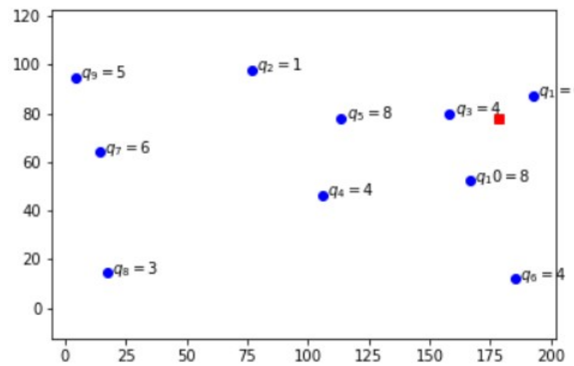
- n il numero di clienti
- N l'insieme dei clienti $N = \{1, 2, ..., n\}$
- V l'insieme dei vertici (o nodi), con $V = \{0\} \cup N$
- A l'insieme di archi $A = \{(i, j) \in V^2 : i \neq j\}$
- c_{ij} il costo di trasporto del tratto $(i, j) \in A$
- Q è la capacità del veicolo. Assumiamo capacità uguali
- q_i la quantità che deve essere consegnata al cliente $i \in N$
- u_i è una variabile ausiliaria derivante dalla *Miller-Tucker-Zemlin (MTZ) formulation*

Dunque la formulazione del problema è la seguente:

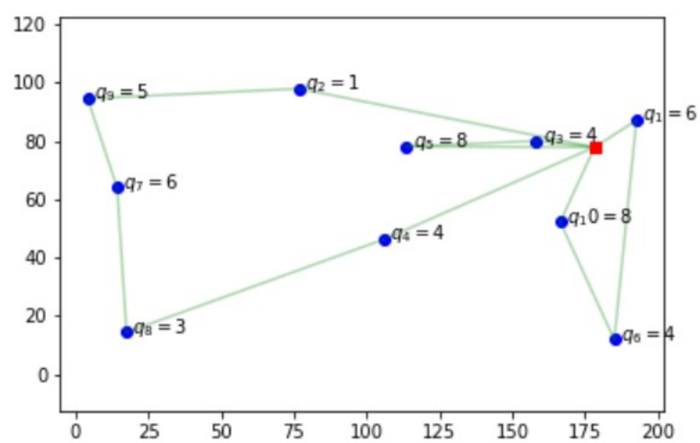
$$\begin{aligned}
\min \quad & \sum_{i,j \in A} c_{ij} x_{ij} & & (1)\\
\text{s.t.} \quad & \sum_{j \in V, j \neq i} x_{ij} = 1 & i \in N & \quad (2)\\
& \sum_{i \in V, i \neq j} x_{ij} = 1 & j \in N & \quad (3)\\
& \text{if } x_{ij} = 1 \;\Rightarrow\; u_i + q_j = u_j & i, j \in A : j \neq 0, i \neq 0 & \quad (4)\\
& q_i \leq u_i \leq Q & i \in N & \quad (5)\\
& x_{ij} \in \{0, 1\} & i, j \in A & \quad (6)
\end{aligned}$$

In (1) viene mostrata la funzione obiettivo, ovvero il costo totale del percorso. I vincoli (2) e (3) sono stati inseriti per garantire che nei nodi il veicolo passi una volta sola.
Il vincolo (4) è stato inserito per evitare i *subtour*.
Il vincolo (5) rappresenta il vincolo di capacità e il vincolo (6) definisce le variabili x_{ij} come variabili binarie. In particolare se $x_{ij} = 1$ allora il percorso (i, j) sarà considerato nella soluzione, viceversa il percorso non sarà considerato.

Le locazioni delle città vengono generate in modo casuale



Una volta risolto il problema di programmazione lineare attraverso il pacchetto python *docplex* (che usa un Branch and Bound), viene fornita la soluzione



Che ha come valore della funzione di costo: 726.249

3 Simulated annealing

Il **simulated annealing** è un algoritmo di **ottimizzazione globale**.
È stato sviluppato originariamente come metodo di simulazione della tempra (annealing) dei solidi. L'annealing è il processo con il quale un solido, viene portato allo stato fluido attraverso alte temperature, dopodichè viene riportato di nuovo allo stato solido o cristallino, a temperature basse, controllando e riducendo gradualmente la temperatura. Ad alte temperature, gli atomi nel sistema si trovano in uno stato molto disordinato e quindi l'energia del sistema è elevata. Per portare questi atomi nella configurazione cristallina corretta (quindi a minore energia), deve quindi essere abbassata la temperatura del sistema. Questo procedimento va fatto però lentamente in quanto si rischia di creare dei difetti nella struttura cristallina del solido. È importante dunque procedere ad un graduale raffreddamento del sistema.

Il sistema si dice essere in *equilibrio termico* alla temperatura T se la probabilità $P(E_i)$ di uno stato avente energia E_i è governata dalla *distribuzione di Boltzmann.*

$$\mathbb{P}(E_i) = \frac{\exp\left(-\frac{E_i}{k_B T}\right)}{\sum_j \exp\left(-\frac{E_j}{k_B T}\right)}$$

L'analogia con l'ottimizzazione globale è presto detta:

- l'energia che viene minimizzata rappresenta la funzione obiettivo e gli atomi della struttura cristallina rappresentano i vari valori della funzione.

- La temperatura T assume un significato più astratto e rappresenta il parametro con il quale il ricercatore decide che direzione far prendere alla simulazione.

- La distribuzione di Boltzmann viene usata per verificare se accettare o meno una perturbazione

Ad alte temperature l'algoritmo si comporta più o meno come una random search. La ricerca salta da un punto all'altro dello spazio delle soluzioni individuando le aree in cui è più probabile individuare l'ottimo globale. A basse temperature l'SA è simile ai metodi steepest descent

3.1 Algoritmo

In particolare, l'algoritmo funziona nel seguente modo:

Step 0:
Vengono inizializzate le variabili e viene generata una soluzione iniziale $\mathbf{x}^{(0)}$.

Step 1: Per $k = 1, \ldots 500$
Data una soluzione $\mathbf{x}^{(k)}$ viene generata una soluzione perturbata $\mathbf{x'}^{(k)}$. Dopodiché viene calcolata la *differenza di energia* ΔE e viene accettata la nuova simulazione con probabilità

$$\min\{1, P(\Delta E)\}$$

Con

$$P(\Delta E) = \exp\left(-\frac{\Delta E}{k_B T}\right) \qquad \text{e} \qquad \Delta E = E(\mathbf{x'}) - E(\mathbf{x})$$

Dunque se la perturbazione è effettivamente migliore, questa viene accettata con probabilità 1, se non lo è, viene accettata con una certa probabilità che via via diventa sempre più piccola.

Step 3:
Avviene adesso la **fase di annealing**, ovvero viene ridotta la temperatura moltiplicandola per una costante r detta *cooling ratio*

$$T_{\text{new}} = r \cdot T_{\text{old}}$$

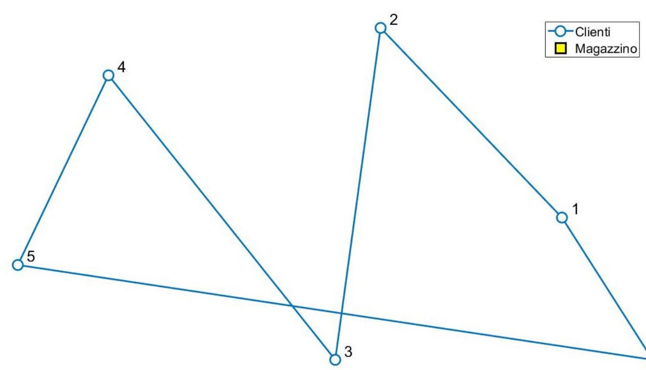
In questa simulazione è stato usato un cooling ratio pari a 0.997.
Successivamente si controllano che non siano verificati i criteri di interruzione e si torna allo **step 1**.

Criteri di interruzione:
L'algoritmo si interrompe quando la temperatura raggiunge il valore 1

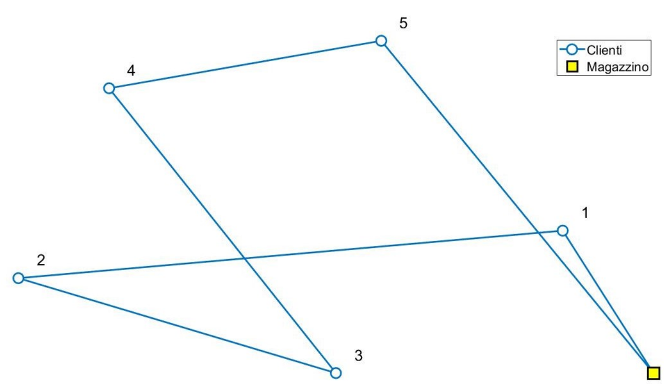
3.2 Metodi di perturbazione

La soluzione perturbata viene generata in 3 modi diversi, ad ogni iterazione, scelti manualmente

Swap
Un modo in cui si può perturbare una soluzione è sicuramente lo scambio fra due clienti di una *route*, in particolare dato un percorso, supponiamo di avere una soluzione del tipo

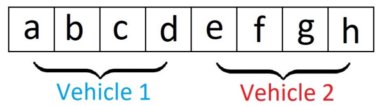


Casualmente, indifferentemente da quale veicolo sono stati assegnati, vengono scelti due clienti (ad esempio 2 e 5) e viene fatto uno scambio nell'ordine del percorso si arriva dunque alla perturbazione

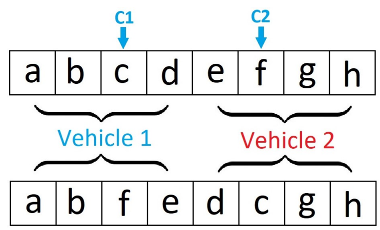


Ribaltamento

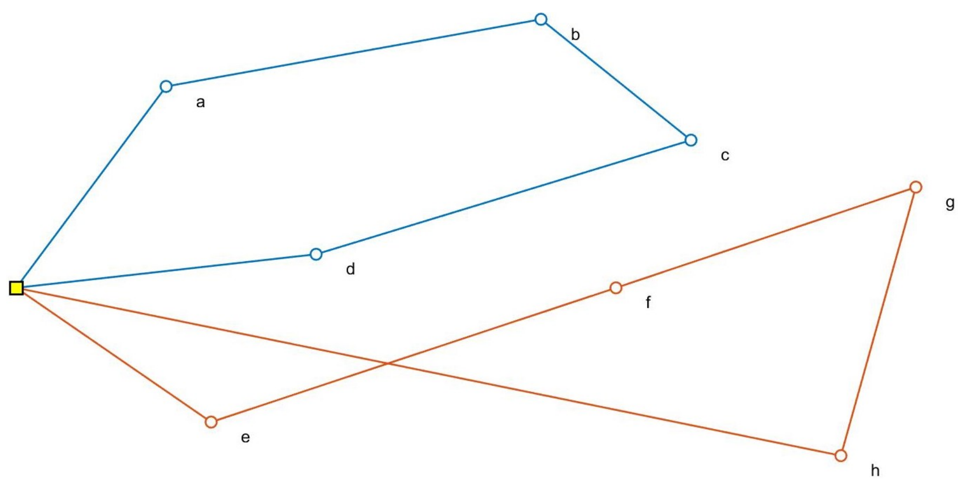
Un altro modo in cui può essere generata una perturbazione è quello del **ribaltamento**.
Supponiamo che la soluzione corrente **x** consista in un vettore di clienti, ad esempio



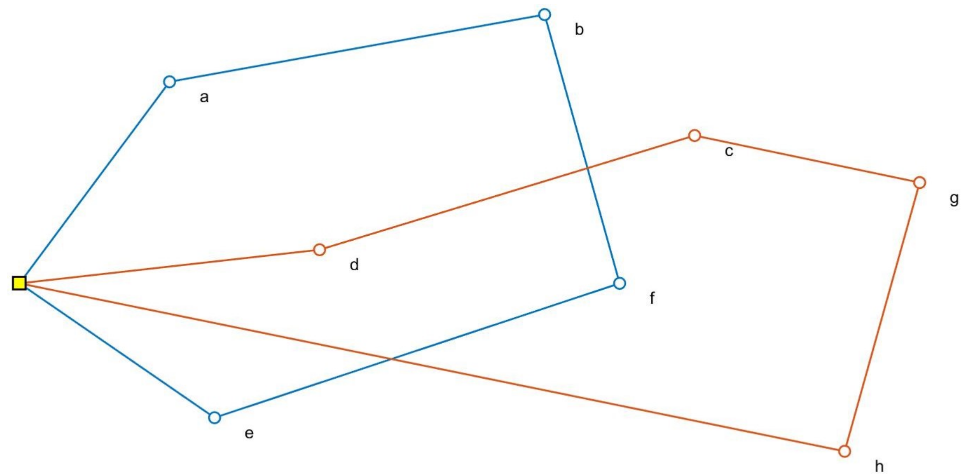
Vengono scelti 2 clienti casuali c1 e c2, e successivamente la porzione di vettore tra c1 e c2 viene ribaltata.



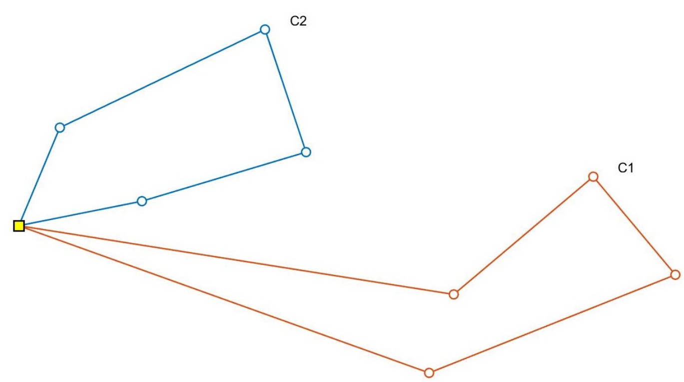
Dal punto di vista geometrico, data la soluzione iniziale

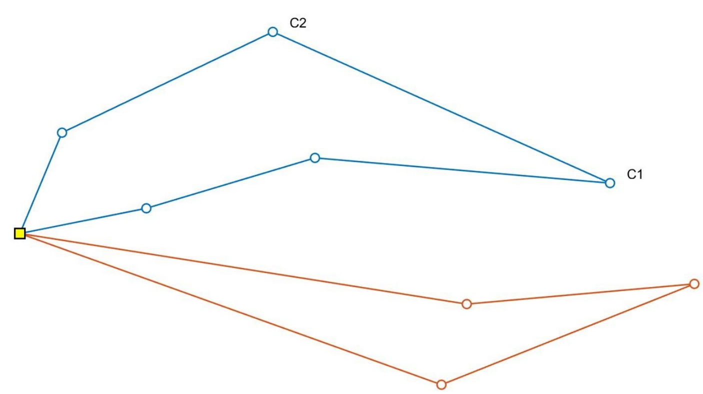


Si genera quindi la perturbazione



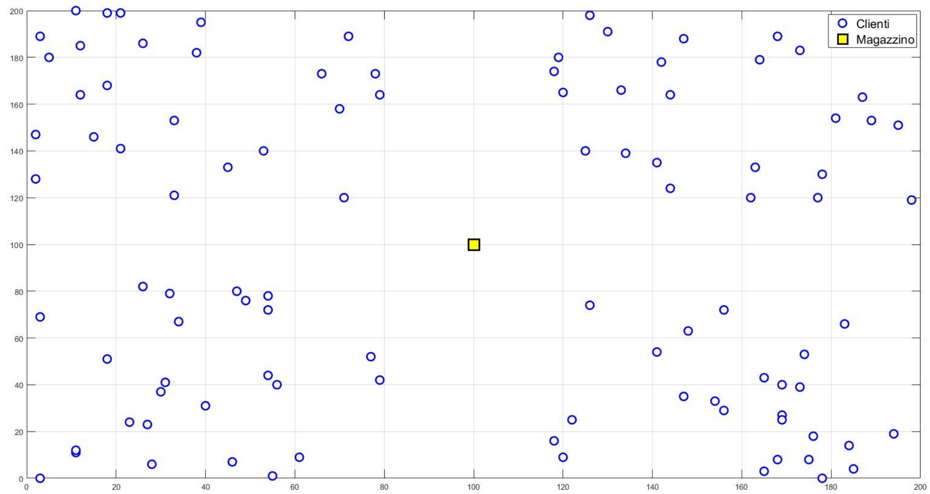
Inserimento
Un ulteriore modo usato per generare una perturbazione di una data soluzione è quello dell'**inserimento**. Vengono casualmente scelti due clienti, che indichiamo con c1 e c2, successivamente c1 viene inserito come cliente successivo a c2.





3.3 Risultati

L'algoritmo è stato fatto girare su dei dati generati in modo casuale ma comunque distinguibili in 4 insiemi



Si comincia dunque dalla soluzione iniziale generata casualmente

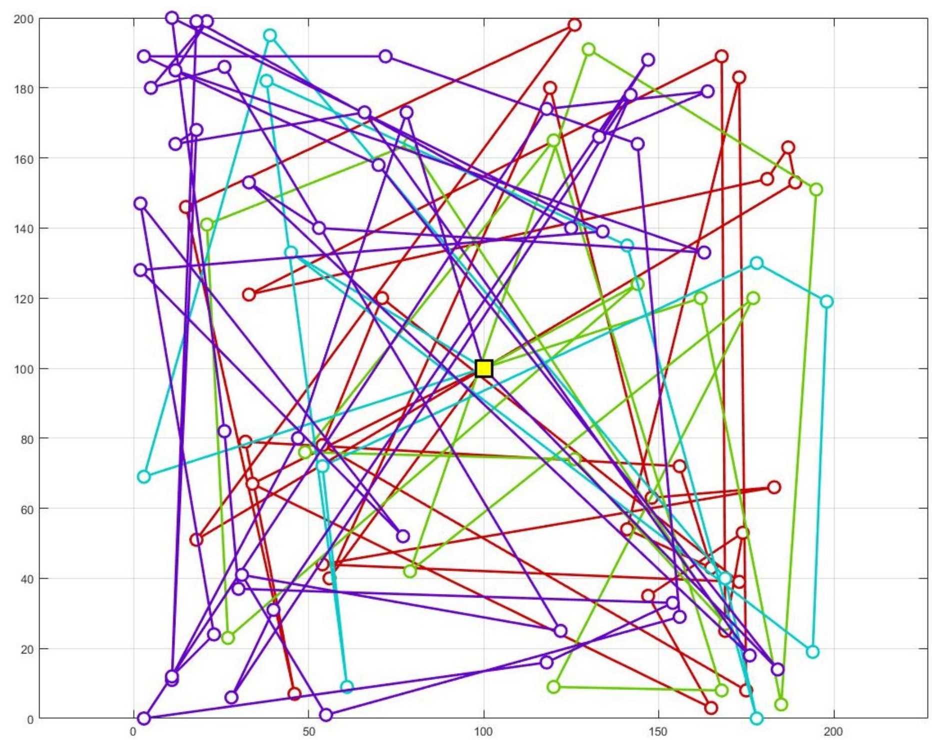


Figure 1: Costo iniziale: 1.0749e+04; Temperatura iniziale $T = 500$

A destra sarà mostrato il valore della funzione di costo ad ogni iterazione e a sinistra una rappresentazione grafica dei percorsi che faranno i veicoli.

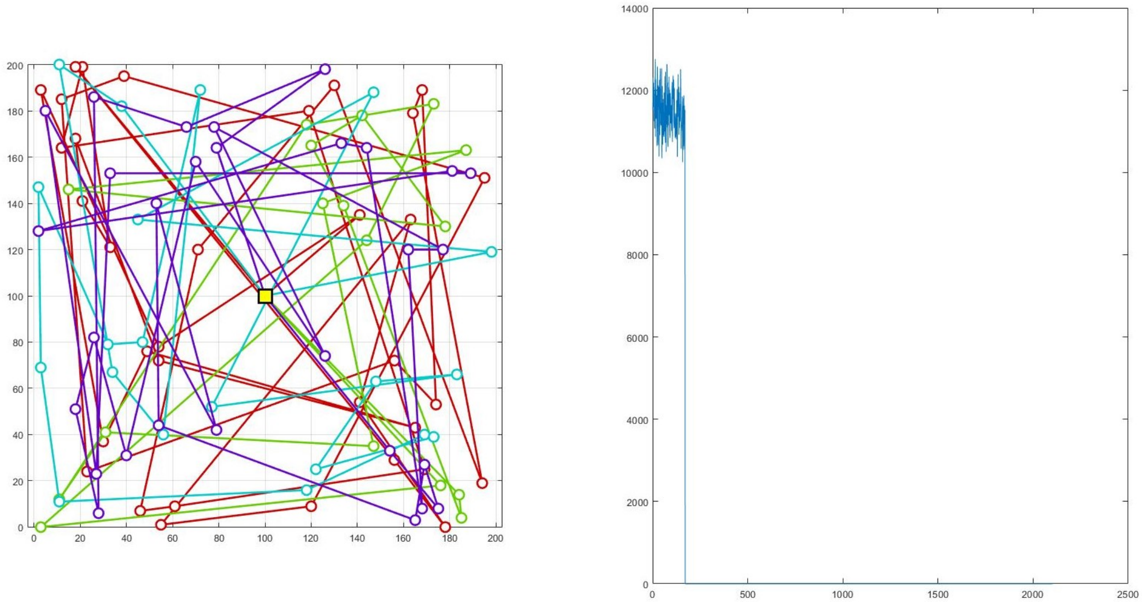


Figure 2: Iteration 171: BestCost = 9788.1023: CurrentCost = 9788.1023 T = 299.1177

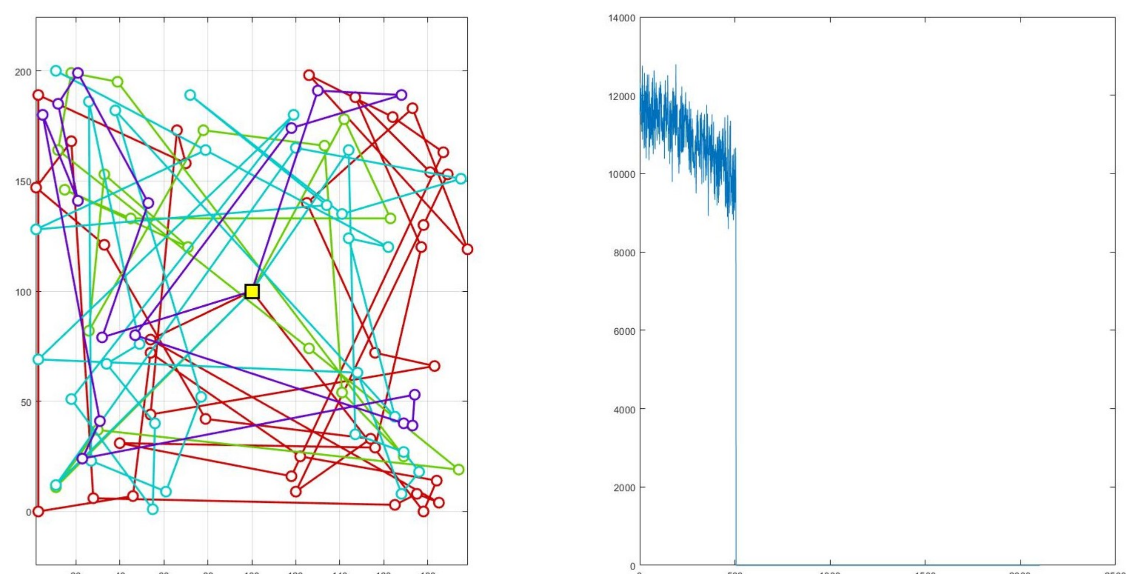


Figure 3: Iteration 506: BestCost = 8368.9642: CurrentCost = 8368.9642 T = 109.3252

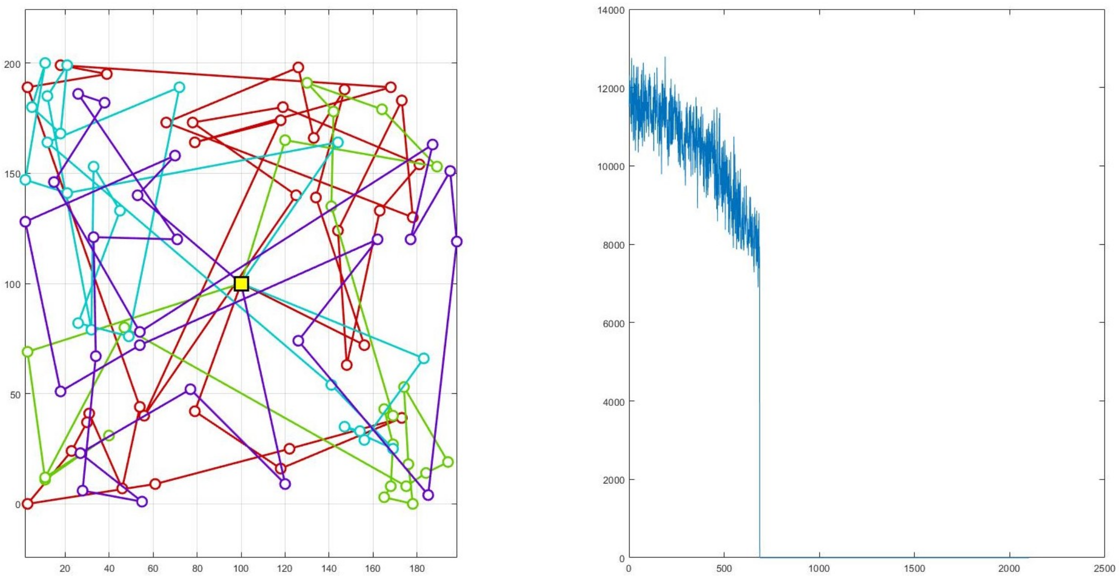


Figure 4: Iteration 687: BestCost = 5887.8592: CurrentCost = 5887.8592 T = 63.4664

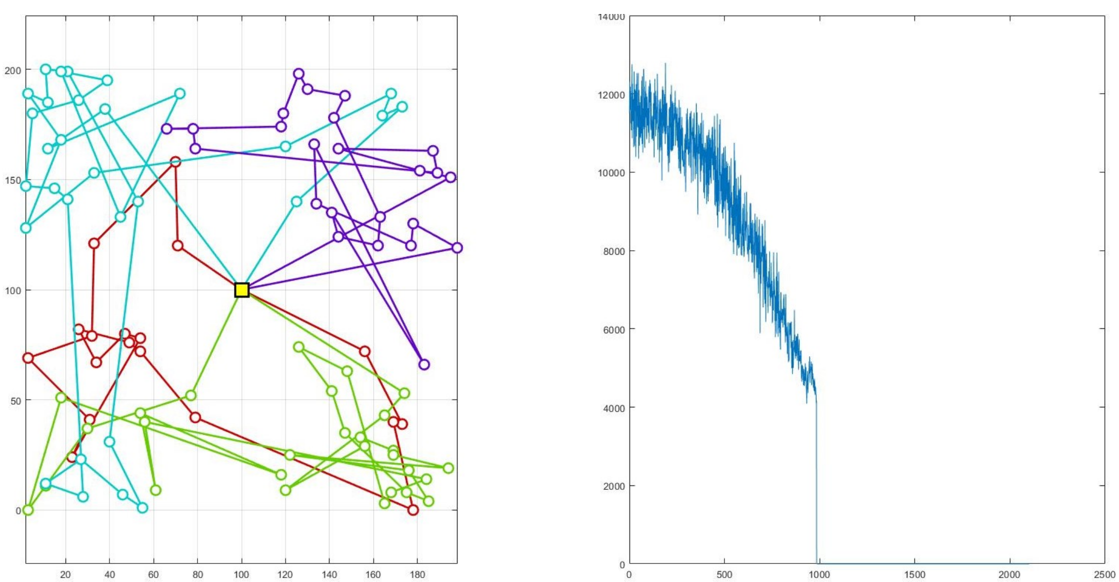


Figure 5: Iteration 984: BestCost = 4086.1214: CurrentCost = 4086.1214 T = 26.0019

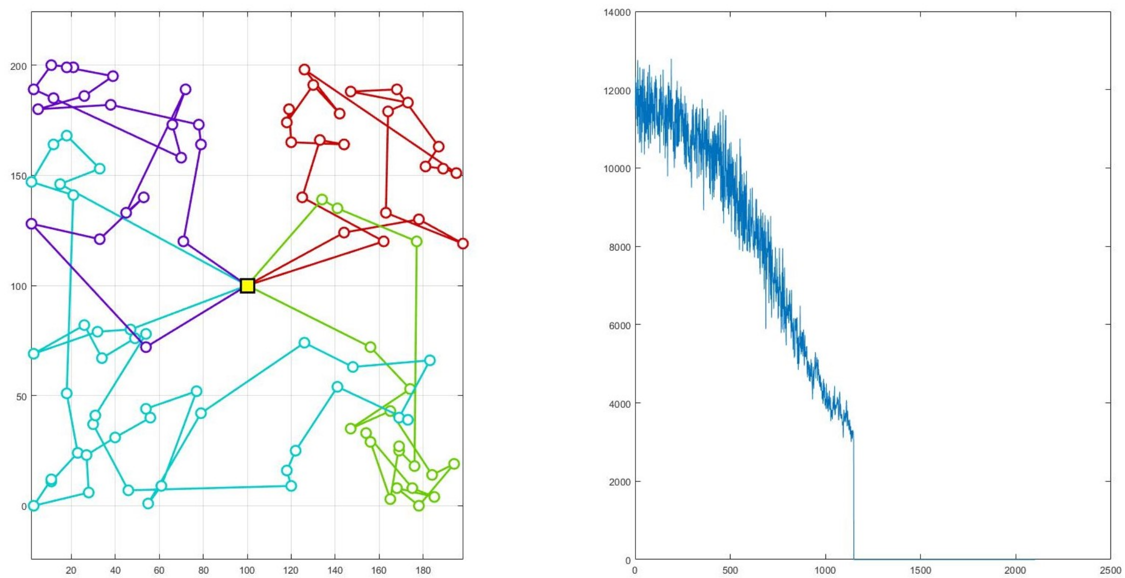


Figure 6: Iteration 1149: BestCost = 2921.1909: CurrentCost = 2921.1909 T = 15.8382

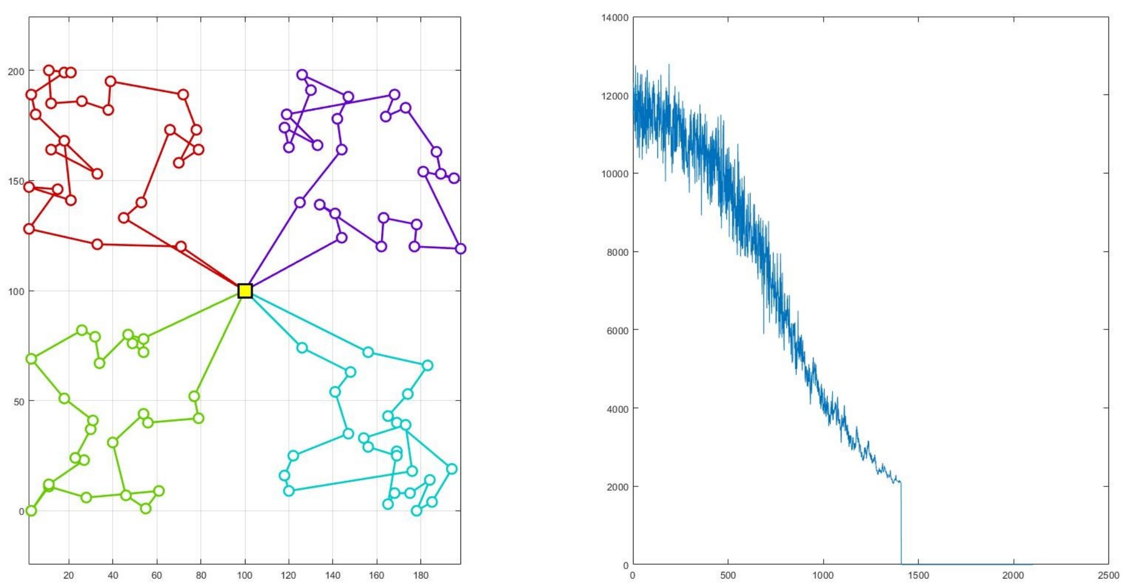


Figure 7: Iteration 1409: BestCost = 2049.5876: CurrentCost = 2049.5876 T = 7.2518

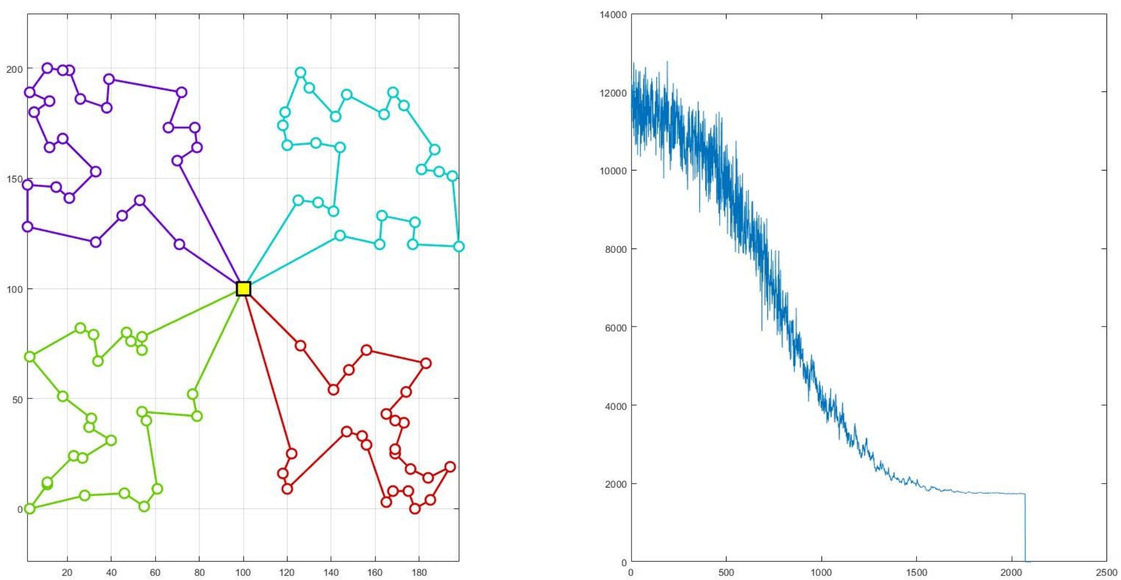


Figure 8: Iteration 2069: BestCost = 1732.4682: CurrentCost = 1738.7664 T = 0.99828

4 Appendice

In questa sezione sono stati riportati i codici utilizzati

4.1 Codice python per LP

```
import numpy as np
import matplotlib.pyplot as plt
from docplex.mp.model import Model

rnd = np.random
rnd.seed(0)

n = 10
Q = 20
N = [i for i in range(1, n+1)]
V = [0] + N
q = {i: rnd.randint(1, 10) for i in N}

loc_x = rnd.rand(len(V))*200
loc_y = rnd.rand(len(V))*100

plt.scatter(loc_x[1:], loc_y[1:], c='b')
for i in N:
plt.annotate('$q_%d=%d$' % (i, q[i]), (loc_x[i]+2, loc_y[i]))
plt.plot(loc_x[0], loc_y[0], c='r', marker='s')
plt.axis('equal')

A = [(i, j) for i in V for j in V if i != j]
c = {(i, j): np.hypot(loc_x[i]-loc_x[j], loc_y[i]-loc_y[j]) for i, j in A}

mdl = Model('CVRP')

x = mdl.binary_var_dict(A, name='x')
u = mdl.continuous_var_dict(N, ub=Q, name='u')

mdl.minimize(mdl.sum(c[i, j]*x[i, j] for i, j in A))
mdl.add_constraints(mdl.sum(x[i, j] for j in V if j != i) == 1 for i in N)
mdl.add_constraints(mdl.sum(x[i, j] for i in V if i != j) == 1 for j in N)
mdl.add_indicator_constraints(mdl.indicator_constraint(x[i, j], u[i]+q[j] == u[j]) for i, j i
mdl.add_constraints(u[i] >= q[i] for i in N)
mdl.parameters.timelimit = 15
solution = mdl.solve(log_output=True)

print(solution)

active_arcs = [a for a in A if x[a].solution_value > 0.9]

plt.scatter(loc_x[1:], loc_y[1:], c='b')
for i in N:
    plt.annotate('$q_%d=%d$' % (i, q[i]), (loc_x[i]+2, loc_y[i]))
for i, j in active_arcs:
    plt.plot([loc_x[i], loc_x[j]], [loc_y[i], loc_y[j]], c='g', alpha=0.3)
plt.plot(loc_x[0], loc_y[0], c='r', marker='s')
plt.axis('equal')
```

4.2 Codici Matlab per il simulated annealing

```matlab
clc;
clear;
close all;

T0 = 500 ; % initial temperature
r = 0.997 ; % temperature damping rateU/il fattore di raffreddamento
Ts = 1 ; % stopping temperature
iter = 500; % n. di iterazioni

rng('default');
rng(08082022);

model = initModel();
% PlotCities(model)


flag = 0;

% initialization
while(1)
route = randomSol(model);
if(isFeasible(route,model))
    break;
end
end

% plotSolution(route,model); % Plot starting point


set(gcf,'unit','normalized','position',[0,0.35,1,0.7]);

cost = calculateCost(route,model);
T = T0;

cnt = 1;
minCost = cost;
minRoute = route;

maxIterate = 2100;
costArray = zeros(maxIterate,1);

% SA
while(T > Ts)
    for k = 1:iter
    mode = randi([1 3]);
    newRoute = createNeibor(route,model,mode);
    newCost = calculateCost(newRoute,model);
    delta = newCost - cost;

    if(delta < 0)
        cost = newCost;
        route = newRoute;
    else
        p=exp(-delta/T);
        if rand() <= p
             cost = newCost;
             route = newRoute;

        end
```

```
    end
    end

    costArray(cnt) = cost;
    if cost<minCost
        minCost = cost;
        minRoute = route;
        flag = 1;
    end

    T = T*r; %  annealing
    disp(['Iteration␣' num2str(cnt) ':␣BestCost␣=␣' num2str(minCost) ':␣CurrentCost␣=␣' num2s
    cnt = cnt+1;

   figure(1);
   if(flag == 1)
   plotSolution(minRoute,model);
   flag = 0;
   end
%  figure(2);
   subplot(1,2,2)
   plot(costArray);

   pause(0.0001);
end
```

```
function res = randomSol(model)
res = randperm(model.city+model.veh-1);
end
```

```
function model = initModel()

city = 100 ; % city number except depot
veh = 4 ; % vehicle number
model.city = city;
model.veh = veh;

xmin=0;
xmax=200;

ymin=0;
ymax=200;



maps = zeros(city+veh,city+veh);

x=randi([xmin xmax],1,city);
y=randi([ymin ymax],1,city);

x0 = 100;
y0 = 100;
model.x0 = x0;
model.y0 = y0;


% for showing multiple vehicles
offset = 18;
x1=randi([xmin xmax/2-offset],1,city/4);
x2=randi([xmin xmax/2-offset],1,city/4);
x3=randi([xmax/2+offset xmax],1,city/4);
x4=randi([xmax/2+offset xmax],1,city/4);

y1=randi([ymin ymax/2-offset],1,city/4);
y2=randi([ymin ymax/2-offset],1,city/4);
y3=randi([ymax/2+offset ymax],1,city/4);
y4=randi([ymax/2+offset ymax],1,city/4);
x = [x1 x2 x3 x4];
y = [y1 y3 y2 y4];


for k = 1:veh
x = [x x0];
y = [y y0];
end

model.x = x;
model.y = y;

n = city+veh;

for i = 1:n
   for j = i:n
      maps(j,i) = sqrt((x(i)-x(j))^2+(y(i)-y(j))^2);
      maps(i,j) = maps(j,i);
   end
end

model.maps = maps;

end
```

```
    function res = isFeasible(route,model)
    len = length(route);
    if route(1)>model.city||route(len)>model.city
        res = 0; return
    end

    for i = 2:len
        if route(i)>model.city && route(i-1)>model.city  % not use all vehicles
            res = 0; return
        end
    end

    city = model.city;
    veh = model.veh;

    oneRoute = find(route>city);
    From = [0 oneRoute]+1;
    To = [oneRoute city+veh]-1;
    routes = cell(veh,1);

for i = 1:veh
       routes{i} = route(From(i):To(i));
       if length(routes{i})<5 || length(routes{i})>50 %capacity costrain
           % we assume every veihicle has 50 capacit unit
           res = 0;
           return
       end
end

    res = 1;
end
```

```
    function newRoute = createNeibor(route,model,mode)
while(1)
    switch mode
        case 1
            newRoute = Swap(route);
        case 2
            % Do Reversion
            newRoute=Reversion(route);

        case 3
            % Do Insertion
            newRoute=Insertion(route);
    end

    if(isFeasible(newRoute,model))
        break;
    end
end
end

function newRoute = Swap(route)
    n = numel(route);

    i = randsample(n,2);
    i1 = i(1);
    i2 = i(2);
    newRoute = route;
    newRoute([i1 i2]) = route([i2 i1]);
end

function newRoute = Reversion(route)
    n=numel(route);

    i=randsample(n,2);
    i1=min(i(1),i(2));
    i2=max(i(1),i(2));

    newRoute=route;
    newRoute(i1:i2)=route(i2:-1:i1);

end


function newRoute = Insertion(route)
    n=numel(route);

    i=randsample(n,2);
    i1=i(1);
    i2=i(2);

    if i1<i2
        newRoute=[route(1:i1-1) route(i1+1:i2) route(i1) route(i2+1:n)];
    else
        newRoute=[route(1:i2) route(i1) route(i2+1:i1-1) route(i1+1:n)];
    end

end
```

```
    function PlotCities(model)
    plot(model.x, model.y, 'bo', 'LineWidth',2,...
            'MarkerSize',10,...
            'MarkerFaceColor','white');
    hold on
    plot(model.x0, model.y0, 'ks',...
        'LineWidth',2,...
        'MarkerSize',18,...
        'MarkerFaceColor','yellow');
    grid on
    legend('Clienti', 'Magazzino', 'FontSize', 15)
end
```

```
    function plotSolution(route,model)
city = model.city;
veh = model.veh;
x0 = model.x0;
y0 = model.y0;
x = model.x;
y = model.y;

oneRoute = find(route>city);
From = [0 oneRoute]+1;
To = [oneRoute city+veh]-1;
routes = cell(veh,1);
subplot(1,2,1);

for i = 1:veh
       routes{i} = route(From(i):To(i));
end

colors=hsv(veh);

for r = 1:veh
    X = [x0 x(routes{r}) x0];
    Y = [y0 y(routes{r}) y0];
    color = 0.8*colors(r,:);

     plot(X,Y,'-o',...
            'Color',color,...
            'LineWidth',2,...
            'MarkerSize',10,...
            'MarkerFaceColor','white');
     hold on;
end

plot(x0,y0,'ks',...
        'LineWidth',2,...
        'MarkerSize',18,...
        'MarkerFaceColor','yellow');

  hold off;
  grid on;
  axis equal;


end
```