ELABORATO DI RICERCA OPERATIVA

Algoritmo euristico per i problemi di localizzazione P-Mediana e Clustering

Professore:
Maurizio Boccia

Studenti:
Francesco Scognamiglio M63001364
Felice Micillo M63001377

1.

Definizione e formulazione dei problemi di P-Mediana e Clustering

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

DEFINIZIONE DEL PROBLEMA DI P-MEDIANA

Il problema di P-Mediana è un problema di localizzazione il cui obiettivo è di minimizzare la somma dei costi di afferenza, localizzando in modo opportuno un numero *P* di centri di servizio.

Il problema può essere rappresentato su un grafo *G(V,A)* :

- *V* è l'insieme dei vertici, potenziali centri e clienti.
- *E* è l'insieme degli archi che collegano un nodo *i* ad un nodo *j* caratterizzato da un costo (costo di afferenza)

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

FORMULAZIONE

- Dati del problema:
  - c_{ij} costo di afferenza in i della domanda generata da j
- Variabili decisionali:
  - $x_{ij} = 1$, se il cliente j afferisce al centro di servizio i, 0 altrimenti
  - $y_i = 1$, se in i è localizzato un centro di servizio, 0 altrimenti

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

FORMULAZIONE

$$\min \sum_{i \in I, j \in J} c_{ij} x_{ij}$$ ← Funzione obiettivo

$$\sum_{i \in I} y_i = p$$ ← Numero di centri da aprire

$$\sum_{i \in I} x_{ij} = 1 \qquad \forall j \in J$$ ← Ogni domanda deve essere soddisfatta

$$x_{ij} \leq y_i \qquad \forall i \in \mathrm{I}, \forall j \in J$$

$$x_{ij} \geq 0 \qquad \forall i \in \mathrm{I}, \forall j \in J$$ ← Variabili upper bounds

$$y_i \in \{0,1\} \qquad \forall i \in I$$

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

DEFINIZIONE DEL PROBLEMA DI CLUSTERING

Il problema di Clustering consiste nel raggruppare un insieme di oggetti con cui bisogna relazionarsi in futuro per qualche utilizzo oppure per facilitare la ricerca degli oggetti stessi nel momento del bisogno.

Nel problema di clustering è fondamentare tenere conto della distanza che vi è tra i punti che compongono un determinato cluster, questa distanza misura la dissimilarità che vi è tra gli oggetti.

Gli oggetti che compongono i cluster possono essere rappresentati su un grafo $G(V,A)$ dove V è l'insieme degli oggetti mentre A è l'insieme degli archi che collegano i nodi tra loro.

L'obiettivo del problema di Clustering è quello di trovare la partizione di V in k cluster, con k prefissato, che minimizzi la somma dei pesi degli archi incidenti in nodi appartenenti ad una stessa classe.

Un cluster può avere un oggetto rappresentativo per tutti gli altri oggetti dello stesso cluster e viene chiamato centroide. Un centroide è posto al centro rispetto a tutti gli altri oggetti. L'insieme dei centroidi può essere paragonato alle P-Mediane di un problema di P-Mediana.

Per cui la formulazione del problema di Clustering è molto simile alla formulazione di un problema di P-Mediana.

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

FORMULAZIONE

$$\min \sum_{i,j \in V} d_{ij} x_{ij}$$

← Funzione obiettivo

$$\sum_{i \in V} y_i = p$$

← Numero di centroidi da trovare

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V$$

← Ogni elemento deve afferire ad un centroide

$$x_{ij} \leq y_j \qquad \forall i,j \in V$$

$$x_{ij} \geq 0 \qquad \forall i,j \in V$$

← Un elemento i può afferire ad uno specifico centroide solo se questo è attivo

$$y_j \in \{0,1\} \qquad \forall j \in V$$

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

2.

Descrizione delle euristiche di ricerca locale

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

EURISTICA DI RICERCA LOCALE

Gli algoritmi di ricerca locale sono delle euristiche migliorative, dove si parte da una soluzione di innesco e si cerca di migliorarla effettuando delle mosse.

Nella fase di inizializzazione si prende in considerazione una soluzione di innesco, solitamente casuale.

Successivamente, si definisce un intorno alla soluzione corrente, attuando una mossa. L'intorno è costituito da un insieme di soluzioni ammissibili.

Tra le soluzioni presenti nell'intorno della soluzione attuale, tramite un criterio di valutazione, si sceglie una soluzione migliore dell'attuale. Nel caso in cui non vi siano soluzioni migliorative allora l'algoritmo si arresta.

Se vi è una soluzione migliore, la soluzione attuale viene sostituita con quest'ultima e si ripetono gli step precedenti finché l'algoritmo non termina.

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

CONSIDERAZIONI SUGLI ALGORITMI EURISTICI

Gli algoritmi euristici ottengono una soluzione approssimata rispetto alla soluzione ottima.

Per migliorare il trade-off che vi è tra la qualità della soluzione e il tempo di calcolo possiamo effettuare:

- Approccio multistart
- Scelta sulla dimensione degli intorni
- Scelta della soluzione corrente
- Diversificazione e intensificazione

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

3.

Descrizione e implementazione dell'Algoritmo di Teitz and Bart per la risoluzione dei problemi di P-Mediana e Clustering

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

ALGORITMO DI TEITZ AND BART

L'algoritmo di Teitz e Bart è un algoritmo migliorativo basato sul concetto di saving dovuto alla sostituzione di un centro aperto con un altro.

Sia S l'insieme dei p centri aperti ed S' l'insieme dei centri non aperti

- Al passo iniziale, l'algoritmo sceglie (anche a caso) i *P* centri di servizio da aprire
- Alla generica iterazione, per ogni coppia i ϵ S e j ϵ S' calcola il saving:

  $s_{ij} \leq z(s) - z(S - \{i\} \cup \{j\})$
- Effettua la sostituzione cui corrisponde il saving più alto; se tutti i saving sono non positivi l'algoritmo si arresta.

Se tutti i saving inerenti ad un elemento j sono negativi, non verrà considerato per il calcolo dei savig all'interazione successiva, quindi non potrà mai essere parte della soluzione finale.

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

IMPLEMENTAZIONE DELL'ALGORITMO

Per implementare l'algoritmo di Teitz and Bart abbiamo optato per il potente linguaggio di programmazione Python.

Il datasheet su cui viene eseguito l'algoritmo è definito in un file json, dove ogni elemento è caratterizzato dal nome di una città («city») e dalle sue coordinate («lat», «lng»).

Dal datasheet viene creata un dizionario che contiene tutte le città come chiave e le coordinate come valori chiamato *cities_coords* e una lista di nomi di tutte le città presenti nel datasheet chiamata *cities_names*.

Dal dizionario e dalla lista ottenute, generiamo una matrice delle distanze che consiste sempre in un dizionario con due chiavi corrispondenti alle città su cui si è calcolata la distanza e come valore la distanza calcolata, la matrice risultate è chiamata *distance_matrix*.

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

IMPLEMENTAZIONE DELL'ALGORITMO

Prima di passare all'implementazione dell'algoritmo di Teitz and Bart, abbiamo implementato la funzione obiettivo che ha come argomenti la matrice delle distanze, la lista dei nomi delle città e la lista delle mediane («distance_matrix», «cities_names», «medians»).

La funzione non fa altro che calcolare la somma di tutti i costi (distanze) minimi tra ogni città che non è mediana e ogni mediana.

Ci restituisce il valore della funzione obiettivo in una variabile *z*.

```
def fun_obj(cities_names, distance_matrix, medians):
    min_distances = []
    for i in cities_names:
        distances = []
        if not(i in medians):
            for j in medians:
                distances.append(distance_matrix[i, j])
            min_distances.append(min(distances))
    res = sum(min_distances)
    return res
```

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

IMPLEMENTAZIONE DELL'ALGORITMO

Dopo aver implementato la funzione obiettivo, abbiamo implementato l'algoritmo di Teitz and Bart.

1.

```
def tb_heuristic(cities_names, distance_matrix, p):

    random_medians = random.sample(cities_names, p)
    z = fun_obj(cities_names, distance_matrix, random_medians)

    print("Soluzione Iniziale => Z: " + str(z) + " P-Mediane: " + str(random_medians))
```

2.

```
medians = random_medians.copy()
bad_medians = []
end_loop = False
k = 0
while not end_loop:
    savings = {}
    tmp_bad_medians = []
    max_val = -1
    cont_negative = 0
    for j in cities_names:
        if not (j in medians) and not (j in bad_medians):
            cont = 0
            for i in medians:
                tmp_median = medians.copy()
                tmp_median[tmp_median.index(i)] = j
                savings[(i, j)] = z - fun_obj(cities_names, distance_matrix, tmp_median)
                if savings[(i, j)] < 0:
                    cont_negative = cont_negative + 1
                    cont = cont + 1
                else:
                    if savings[(i, j)] > max_val:
                        max_val = savings[(i, j)]
                        best_i = i
                        best_j = j
            if cont == p:
                tmp_bad_medians.append(j)
    bad_medians = tmp_bad_medians.copy()
```

3.

```
if cont_negative != len(savings):
    medians[medians.index(best_i)] = best_j
    z = fun_obj(cities_names, distance_matrix, medians)
    print("Soluzione " + str(k) + " => Z: " + str(z) + " P-Mediane: " + str(medians))
    k = k + 1
else:
    print("Soluzione Ottima all'iterazione " + str(k-1) + " => Z: " + str(z) + " P-Mediane: "
+ str(medians))
    end_loop = True
    return z, medians
```

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

IMPLEMENTAZIONE DELL'ALGORITMO

Per realizzare i cluster dalla soluzione dell'algoritmo di Teitz and Bart, abbiamo implementato una funzione di assegnamento, dove, in base alla distanza minima da tutti i centroidi crea una matrice di assegnamento.

```
def assegnamento(distance_matrix, cities_names, centroids):

    cluster_matrix = {}
    for i in cities_names:
        if not(i in centroids):
            min_val = 100000000
            for j in centroids:
                if distance_matrix[(i, j)] < min_val:
                    min_val = distance_matrix[(i, j)]
                    centroid = j
            cluster_matrix[(i, centroid)] = distance_matrix[(i, centroid)]
```

La funzione ci restituisce la matrice dei cluster chiamata *cluster_matrix*.

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

IMPLEMENTAZIONE DELL'ALGORITMO

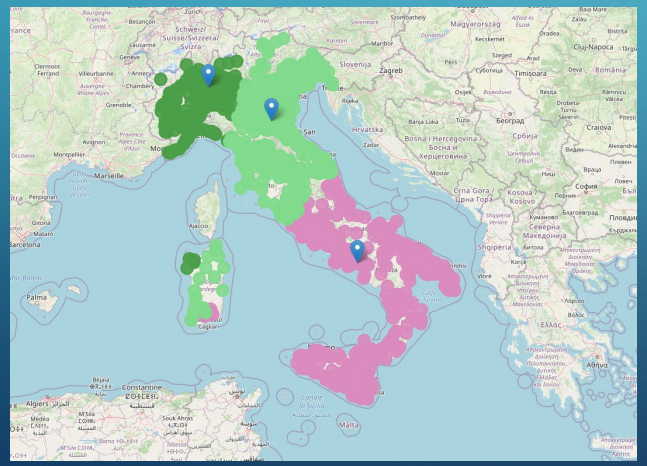
Per rappresentare i cluster, abbiamo implementato una funzione che, tramite la libreria *folium*, genera una mappa su cui verranno disegnati i cluster identificati da colori differenti.

1.

```
def map_generator(cluster_matrix, centroids, cities_coords):

    lat, lng = cities_coords[centroids[0]]
    colors_array = []
    for i in range(len(centroids)):
        colors_array.append([random.random(), random.random(), random.random()])
    map_clusters = folium.Map(location=[lat, lng], zoom_start=4)
    cont = 0
```

Esempio:



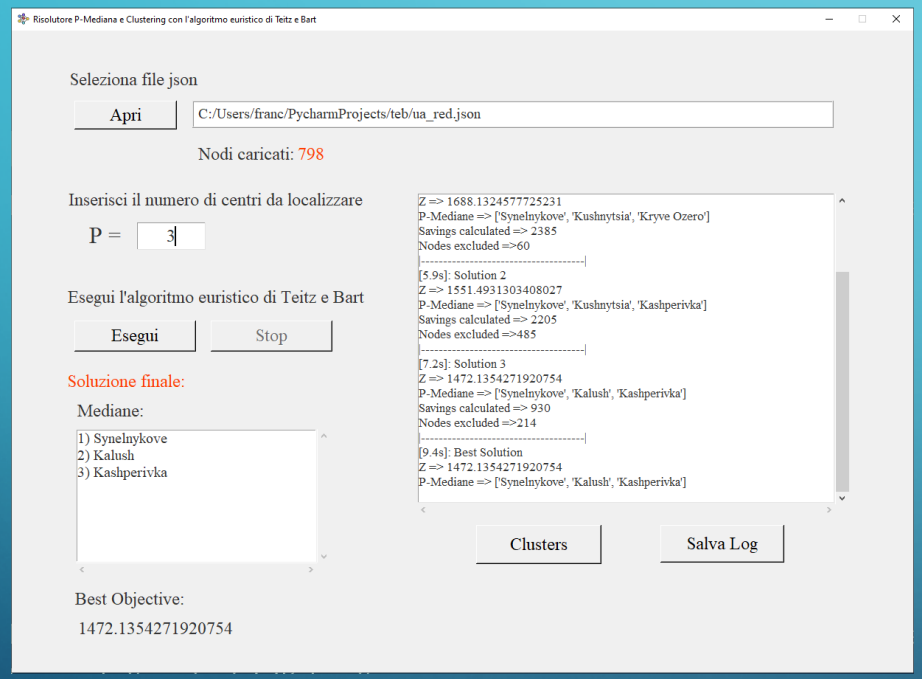
2.

```
for k in centroids:
    for i, j in cluster_matrix:
        if j == k:
            lat, lng = cities_coords[i]
            folium.vector_layers.CircleMarker(
                [lat, lng],
                radius=10,
                tooltip='Nodo: ' + str(i) + ' Lat: ' + str(lat) + ' Lng: ' + str(lng),
                color=colors.to_hex(colors_array[cont]),
                fill=True,
                fill_color=colors.to_hex(colors_array[cont]),
                fill_opacity=0.9
            ).add_to(map_clusters)
    cont = cont + 1
    lat, lng = cities_coords[k]
    folium.vector_layers.Marker(
        [lat, lng],
        radius=10,
        tooltip='Centroide: ' + str(k) + ' Lat: ' + str(lat) + ' Lng: ' + str(lng)
    ).add_to(map_clusters)
return map_clusters
```

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

IMPLEMENTAZIONE DELL'ALGORITMO

Inoltre, per migliorare l'usabilità del software realizzato, abbiamo implementato un'interfaccia grafica.



L'interfaccia grafica non influisce in alcun modo sui tempi di calcolo dell'algoritmo in quanto, quest'ultimo, viene eseguito su un thread dedicato.

Dopo l'esecuzione dell'algoritmo, possiamo visualizzare i cluster oppure salvare direttamente sul pc il log con le iterazioni dell'algoritmo.

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

4.

Implementazione del metodo ottimo per la risoluzione dei problemi di P-Mediana e di Clustering

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

IMPLEMENTAZIONE DELLA SOLUZIONE OTTIMA

Per il calcolo della soluzione ottima, abbiamo utilizzato una libreria di python gurobi.

Per la risoluzione del problema di mediana abbiamo utilizzato gli stessi datasheet dell'algoritmo euristico, per cui abbiamo utilizzato le stesse funzioni per estrarre i dati necessari.

Abbiamo inizializzato il modello e le variabili («Xvars» e «Yvars»)

```
import gurobipy as gp
from gurobipy import GRB

Mod = gp.Model("P-Median")

Xvars = Mod.addVars(distance_matrix.keys(), obj=distance_matrix, vtype=GRB.CONTINUOUS, name="x")
Yvars = Mod.addVars(cities_names, vtype=GRB.BINARY, name="y")
```

IMPLEMENTAZIONE DELLA SOLUZIONE OTTIMA

Successivamente abbiamo definito i vincoli in base alla formulazione del problema:

- Vincolo sul numero di centri da aprire:

```
Constr_1 = Mod.addConstr(Yvars.sum() == p)
```

- Vincolo di assegnamento:

```
Constr_2 = Mod.addConstrs(Xvars.sum('*',j) == 1 for j in cities_names)
```

- Vincolo di upper bound variabile:

```
Constr_3 = Mod.addConstrs(Xvars[(i,j)] <= Yvars[i] for i,j in distance_matrix)
```

- Vincolo di interezza:

```
Constr_4 = Mod.addConstrs(Xvars[(i,j)] >= 0 for i,j in distance_matrix)
```

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

IMPLEMENTAZIONE DELLA SOLUZIONE OTTIMA

Dopo l'inizializzazione dei vincoli del problema, abbiamo avviato la procedura di risoluzione:

```
Mod.optimize()
```

Una volta risolto il problema tramite il risolutore, abbiamo prelevato i valori associati alle variabili del problema:

```
Yvals = Mod.getAttr('x', Yvars)
Xvals = Mod.getAttr('x', Xvars)

medians = []
for i in Yvals.items():
    if i[1] == 1:
        medians.append(i[0])
z = 0
for i, j in Xvals.items():
    if j == 1:
        z = z + distance_matrix[i]

print("Best Solution: " + str(z))
print("Best Medians: ", medians)
```

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

5.

Confronto teorico delle soluzioni ottenute

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

CONSIDERAZIONI SUI RISULTATI OTTENUTI

Sia I un esempio o istanza di un dato problema P e siano:

- EUR(I): valore della soluzione fornita dall'algoritmo euristico;

- OPT(I): valore della soluzione ottima fornita da un algoritmo esatto.

La qualità della soluzione prodotta dall'euristica è data da:

$$gap = \frac{|OPT(I) - EUR(I)|}{|OPT(I)|} \times 100$$

Francesco Scognamiglio M63001364 | Felice Micillo M63001377

CONSIDERAZIONI SUI RISULTATI OTTENUTI

Test 1:

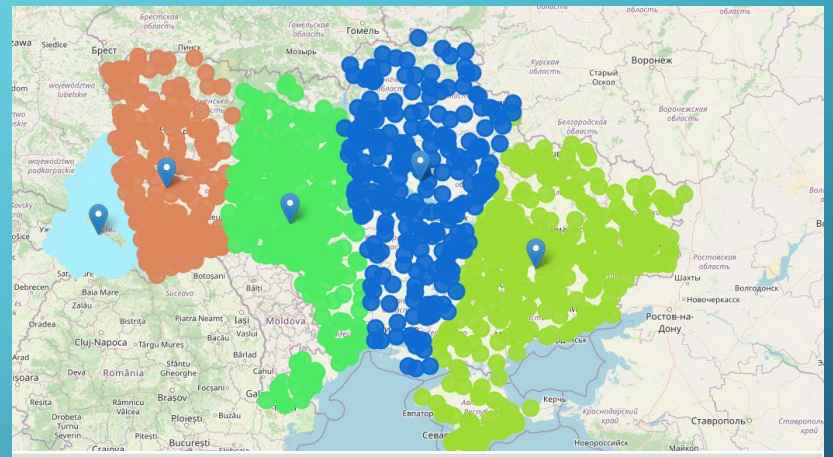
Nodi analizzati: 1370 | P = 5

OPT(I) = 1966,95 | EUR(I) = 2099,90

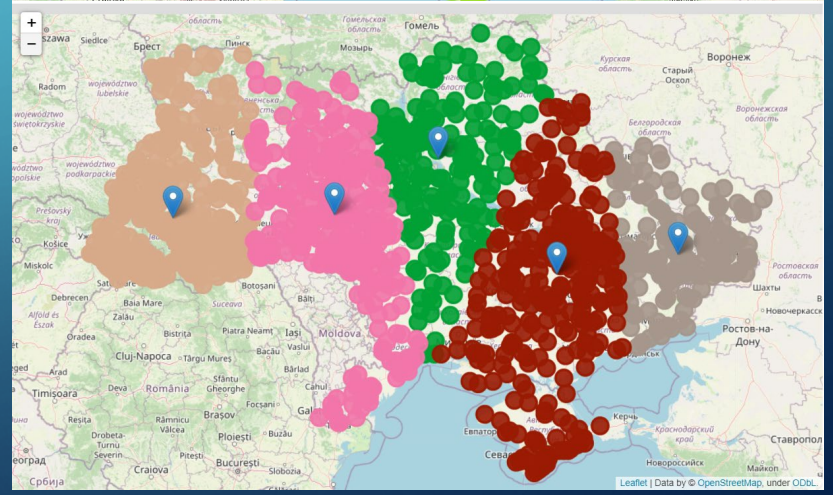
gap = 6,75%

T(OPT) = 200,53s | T(EUR) = 9,4s

Clusters (EUR):



Clusters (OPT):



Francesco Scognamiglio M63001364 | Felice Micillo M63001377

CONSIDERAZIONI SUI RISULTATI OTTENUTI

Test 2:

Nodi analizzati: 798 | P = 5

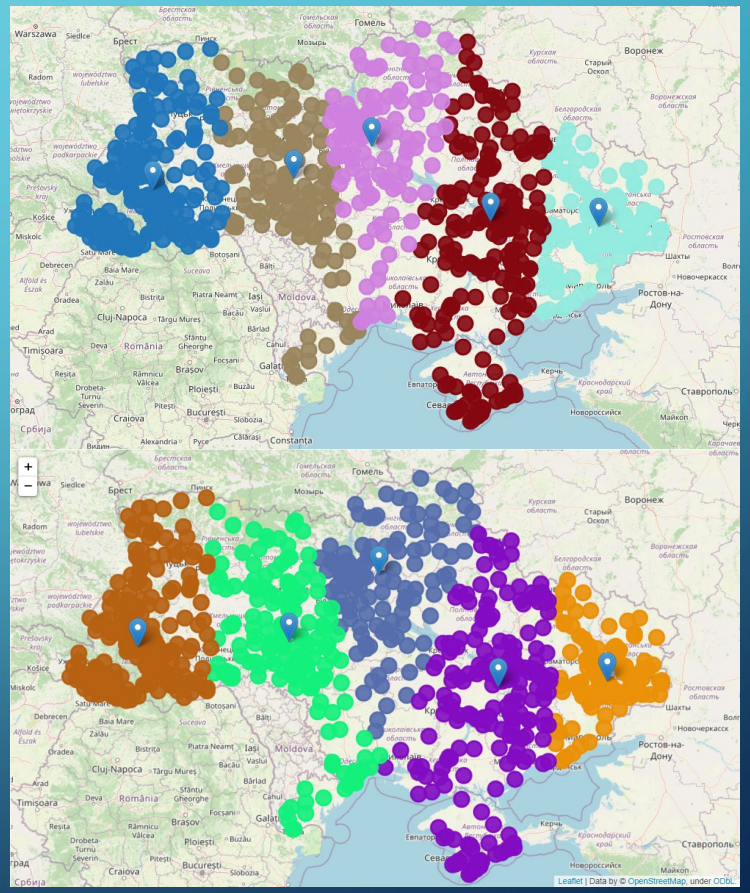
OPT(I) = 1139,59 | EUR(I) = 1152,07

gap = 1,09%

T(OPT) = 119,34s | T(EUR) = 24,2s

Clusters (EUR):

Clusters (OPT):



Francesco Scognamiglio M63001364 | Felice Micillo M63001377

CONSIDERAZIONI SUI RISULTATI OTTENUTI

Test 3:

Nodi analizzati: 1311 | P = 6

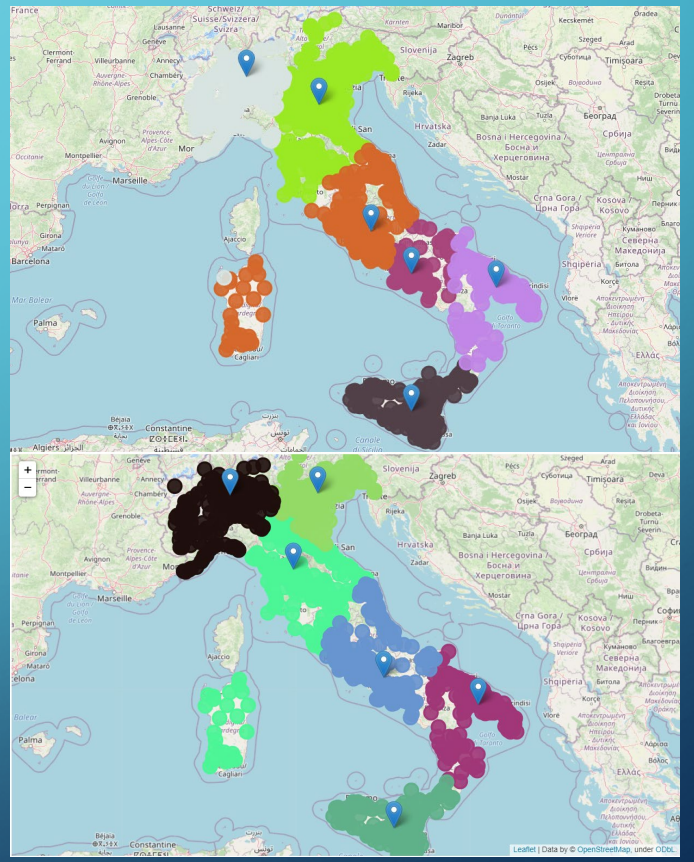
OPT(I) = 1250,72 | EUR(I) = 1267,06

gap = 1,3%

T(OPT) = 143,19s | T(EUR) = 80,4s

Clusters (EUR):

Clusters (OPT):



Francesco Scognamiglio M63001364 | Felice Micillo M63001377

CONSIDERAZIONI SUI RISULTATI OTTENUTI

Di risultati ottenuti dai test effettuati, possiamo dedurre che con un numero ristretto di nodi, il gap tra l'euristica e la soluzione ottima scende drasticamente. Non è da sottovalutare l'importanza della scelta della variabile P: se vogliamo localizzare un numero elevato di P-Mediane il tempo di esecuzione è maggiore sia per il risolutore, sia per l'algoritmo euristico.

Francesco Scognamiglio M63001364 | Felice Micillo M63001377