***Improve Werahouse Productivity: using Spatial Clustering (Parte 2)***

1. ***Due livelli di Ottimizzazione***

Nella parte 1 quello che si è cercato di fare è di stimare la picking route walking distance totale per un set di orders, usando:

1. **Werahouse Mapping**: Collegare ciascuna order line con le picking location coordinate (x,y) nel magazzino;
2. **Calcolo della Distanza**: funzione che calcola la walking distance da 2 picking location

Inoltre, nella parte 1, si è decio anche di adottare un approccio semplice per le seguenti:

1. **Picking Route Design**: data la possibilità di scegliere tra diverse picking locations, il werahouse picker sceglierà sempre di andare alla più vicina (Next Closest Location Strategy);
2. **Order** **Waving**: orders sono ordinati e raggruppati in waves in base al tempo di ricezione (TimeStamp).

Come si riesce a migliorare?

Sicuramente esistono algoritmi più complessi che ci permettono di migliorare, ma esistono soluzioni semplici che ottimizzano il nostro algoritmo.

1. ***Order Waves usando Picking Locations Clustering***

Gli ordini a riga singola hanno il vantaggio di trovarsi in un'unica location di stoccaggio; il raggruppamento di più ordini a riga singola per cluster può garantire che il picker rimanga in una zona delimitata e, quindi, non faccia troppi spostamenti.

Dove si trovano gli ordini a riga singola?

Immagine che contiene testo

Descrizione generata automaticamente

*Figura 1: Order Lines DataFrame*

**Funzione**: Calcolo del numero di ordini a riga singola per location di stoccaggio (%)

**Codice Python:**

*# Creare un Pivot Table Orderlines a secondo della location*

**def** lines\_locations(df\_orderlines, filename):

*# Groupby*

df\_count = pd.DataFrame(df\_orderlines.groupby(['Alley\_Number', 'Cellule'])['Coord'].count()).reset\_index()

df\_pivot = pd.pivot\_table(df\_count, values='Coord', index=['Cellule'],

columns=['Alley\_Number'], aggfunc=np.sum).fillna(0)

*# Totale*

lines\_total = df\_pivot.sum().sum()

df\_pivot = df\_pivot/lines\_total

df\_pivot.to\_excel(filename)

**return** df\_pivot

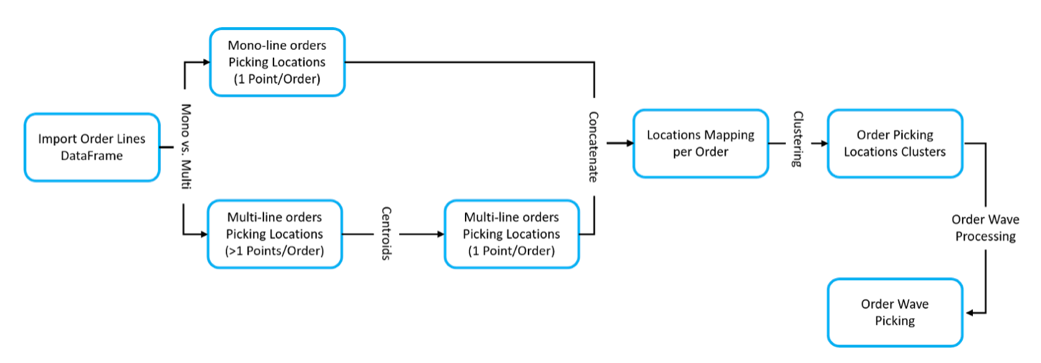
Immagine che contiene tavolo

Descrizione generata automaticamente

*Figura 2: Distribuzione di righe di ordini a riga singola per posizione di stoccaggio: 5.000 righe di ordine (%)*

**Nota:** si prende come riferimento l’esempio di distribuzione sopra

* **Scope**: 5.000 righe d'ordine per 23 corridoi;
* **Ordini a riga singola:** 49% degli ordini situati nei vicoli A11, A10 e A09.
  1. ***Picking locations clustering using Scipy***



*Figura 3: Order Lines Processing for Order Wave Picking using Clustering by Picking Location*

**Idea**: Picking Location Clustering

Raggruppamento delle picking locations per clusters per ridurre la walking distance per ciascun picking route. (Esempio: la massima walking distance fra 2 locations è <15)

Il clustering spaziale è il task di raggruppare insieme un set di punti in modo che gli oggetti nello stesso cluster siano più simili tra loro rispetto agli oggetti in altri cluster.

Immagine che contiene testo, termometro, dispositivo

Descrizione generata automaticamente

*Figura 4: Esempio di tre Picking Location Clusters*

Qui, la metrica di somiglianza sarà la walking distance da una posizione all'altra.

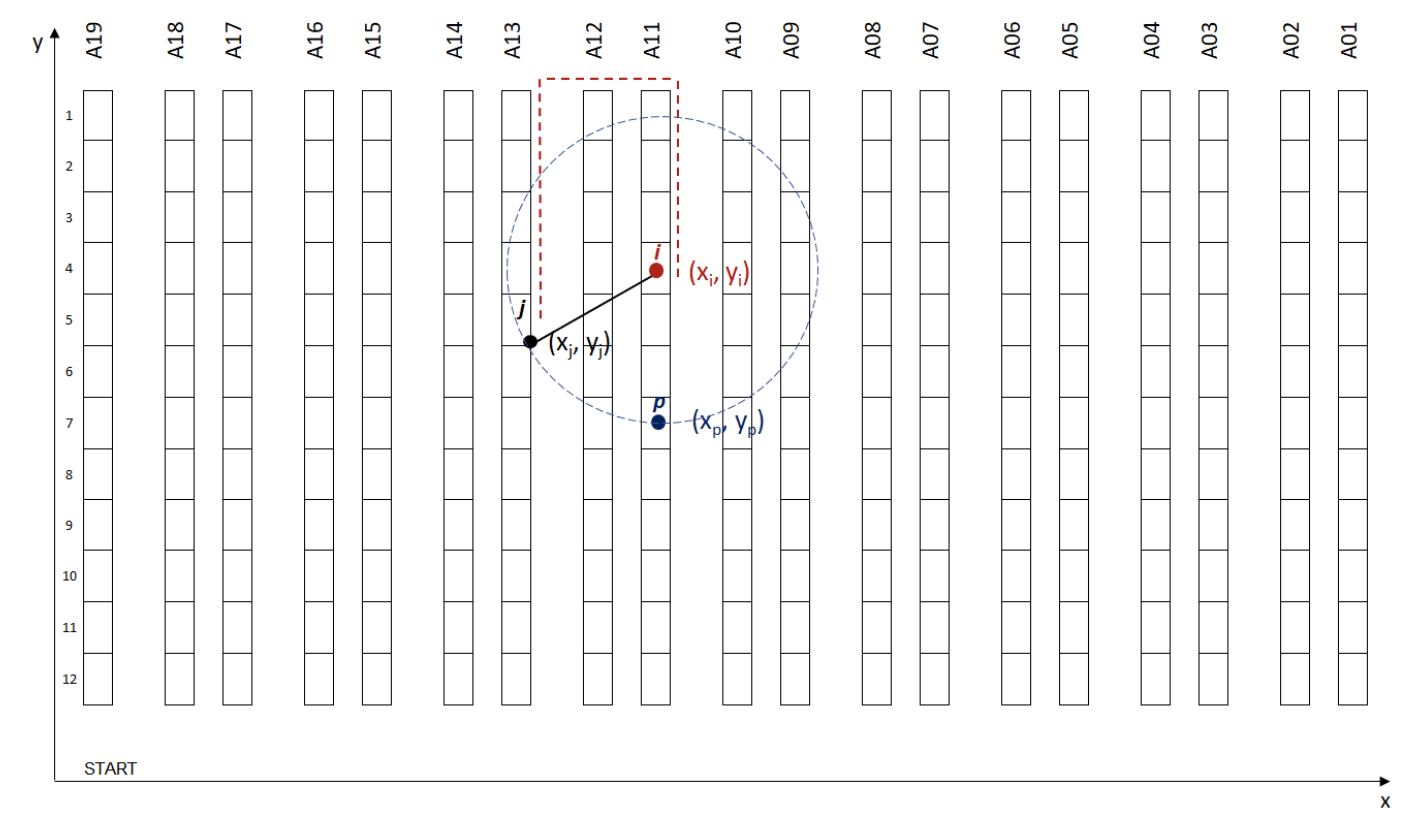
Ad esempio, si vogliono raggruppare le locations assicurandosi che la massima walking distance tra due locations sia di 10 m.

*Euclidian Distance vs. Walking Distance*

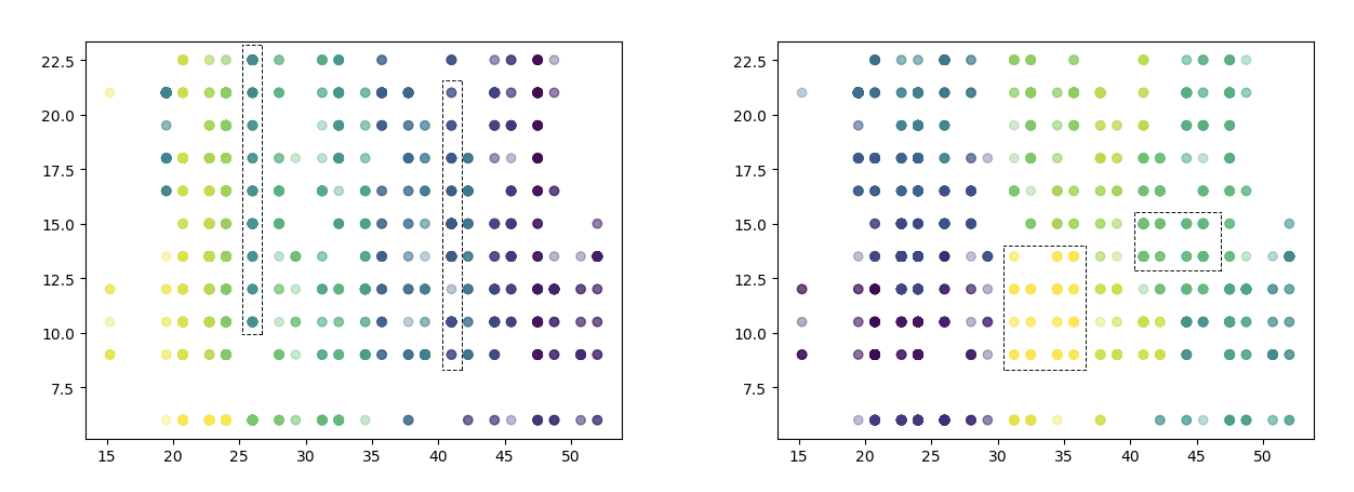
Per il modello specifico definito, non si possono usare metodi di raggruppamento convenzionali utilizzando la distanza euclidea. In effetti, la walking distance (usando la distance\_picking function) è diversa dalla distanza euclidea.

Come si evince dalla *Figura 5,* la distanza Euclidea fra i (xi,Yi) e i 2 punti: p (x\_p,y\_p) e j (x\_j,y\_j) sono uguali. Invece, se si compare il picker Walking distance, p (x\_p,y\_p) è più vicino.

Per questo modello, Picker Walking Distance è la metrica specifica che si va ad usare e con l’obiettivo di ridurre. Pertanto, l'algoritmo di clustering dovrebbe utilizzare la una distance\_walking personalizzata per prestazioni migliori.

*Figura 5: Esempio di Walking vs. Euclidian Distance*

**Esempio**: Locations Clustering entro 25 m di distanza (5.000 righe d'ordine)



*Figura 6: A sinistra: Clustering usando la Walking Distanza / A destra: Clustering usando la Euclidian Distance*

L'esempio a sinistra che utilizza la Walking Distance è il raggruppamento di locations all'interno della stessa corsia riducendo la pciking route distance; mentre l'esempio a destra può raggruppare posizioni che coprono più corridoi.

**Funzione**: Clusters for Single Line Orders using Walking Distance

Per un set di ordini, le righe estraggono ordini a righe singole (df\_orderlines) e creano clusters di storage locations entro una distanza (dist\_method) usando una funzione di distanza custom(dist\_method).

Si espone di seguito il **Codice Python**, implementato usando le funzioni *ward* e *fcluster* di Scipy per creare clusters di Picking Location usando una metrica *distance\_func* (walking distance).

**Codice Python:**

**from** scipy.cluster.vq **import** kmeans2, whiten

**from** scipy.spatial.distance **import** pdist

**from** scipy.cluster.hierarchy **import** ward, fcluster

**from** calculate\_distance **import** dist\_func

**from** pre\_processing **import** mono\_order

**def** clustering\_loc(df, distance\_threshold, dist\_method, orders\_number,

''' Step 1: Clustering + Mapping'''

wave\_start, clust\_start, df\_type):

*# 1. Create Clusters*

list\_coord, list\_OrderNumber, clust\_id, df = cluster\_wave(df, distance\_threshold, 'custom', clust\_start, df\_type)

clust\_idmax = max(clust\_id) *# Last Cluster ID*

*# 2. Mapping Order lines with clusters*

dict\_map, dict\_omap, df, Wave\_max = lines\_mapping\_clst(df, list\_coord, list\_OrderNumber, clust\_id, orders\_number, wave\_start)

**return** dict\_map, dict\_omap, df, Wave\_max, clust\_idmax

**def** cluster\_wave(df, distance\_threshold, dist\_method, clust\_start, df\_type):

'''Step 2: Creo waves by clusters'''

*# Creo colonne per Clustering*

**if** df\_type == 'df\_mono':

df['Coord\_Cluster'] = df['Coord']

*# Mapping points*

df\_map = pd.DataFrame(df.groupby(['OrderNumber', 'Coord\_Cluster'])['SKU'].count()).reset\_index() *# Here we use Coord Cluster*

list\_coord, list\_OrderNumber = np.stack(df\_map.Coord\_Cluster.apply(**lambda** t: literal\_eval(t)).values), df\_map.OrderNumber.values

*# Cluster picking locations*

clust\_id = cluster\_locations(list\_coord, distance\_threshold, dist\_method, clust\_start)

clust\_id = [(i + clust\_start) **for** i **in** clust\_id]

*# List\_coord*

list\_coord = np.stack(list\_coord)

**return** list\_coord, list\_OrderNumber, clust\_id, df

**def** cluster\_locations(list\_coord, distance\_threshold, dist\_method, clust\_start):

''' Step 3: Creo clusters per la location (x,y)'''

*# Creo la linkage matrix*

**if** dist\_method == 'euclidian':

Z = ward(pdist(np.stack(list\_coord)))

**else**:

Z = ward(pdist(np.stack(list\_coord), metric = distance\_picking\_cluster))

*# Single cluster array*

fclust1 = fcluster(Z, t = distance\_threshold, criterion = 'distance')

**return** fclust1

**Funzione**: Single Line Orders Mapping with ClusterID

Per un insieme di ordini, le righe estraggono gli ordini a righe singole (df), l'ID del cluster e il numero degli ordini, questa funzione consente di mappare il dataframe con l'ID del cluster per la creazione della wave.

**Codice Python:**

**def** lines\_mapping\_clst(df, list\_coord, list\_OrderNumber, clust\_id, orders\_number, wave\_start):

'''Step 4: Mapping Order lines mapping with clustering '''

*# Dictionnary for mapping by cluster*

dict\_map = dict(zip(list\_OrderNumber, clust\_id))

*# Dataframe mapping*

df['ClusterID'] = df['OrderNumber'].map(dict\_map)

*# Order by ID and mapping*

df = df.sort\_values(['ClusterID','OrderNumber'], ascending = True)

list\_orders = list(df.OrderNumber.unique())

*# Dictionnary for order mapping*

dict\_omap = dict(zip(list\_orders, [i **for** i **in** range(1, len(list\_orders))]))

*# Order ID mapping*

df['OrderID'] = df['OrderNumber'].map(dict\_omap)

*# Create Waves: Increment when reaching orders\_number or changing cluster*

df['WaveID'] = wave\_start + ((df.OrderID%orders\_number == 0) | (df.ClusterID.diff() != 0)).shift(1).fillna(0).cumsum()

wave\_max = df.WaveID.max()

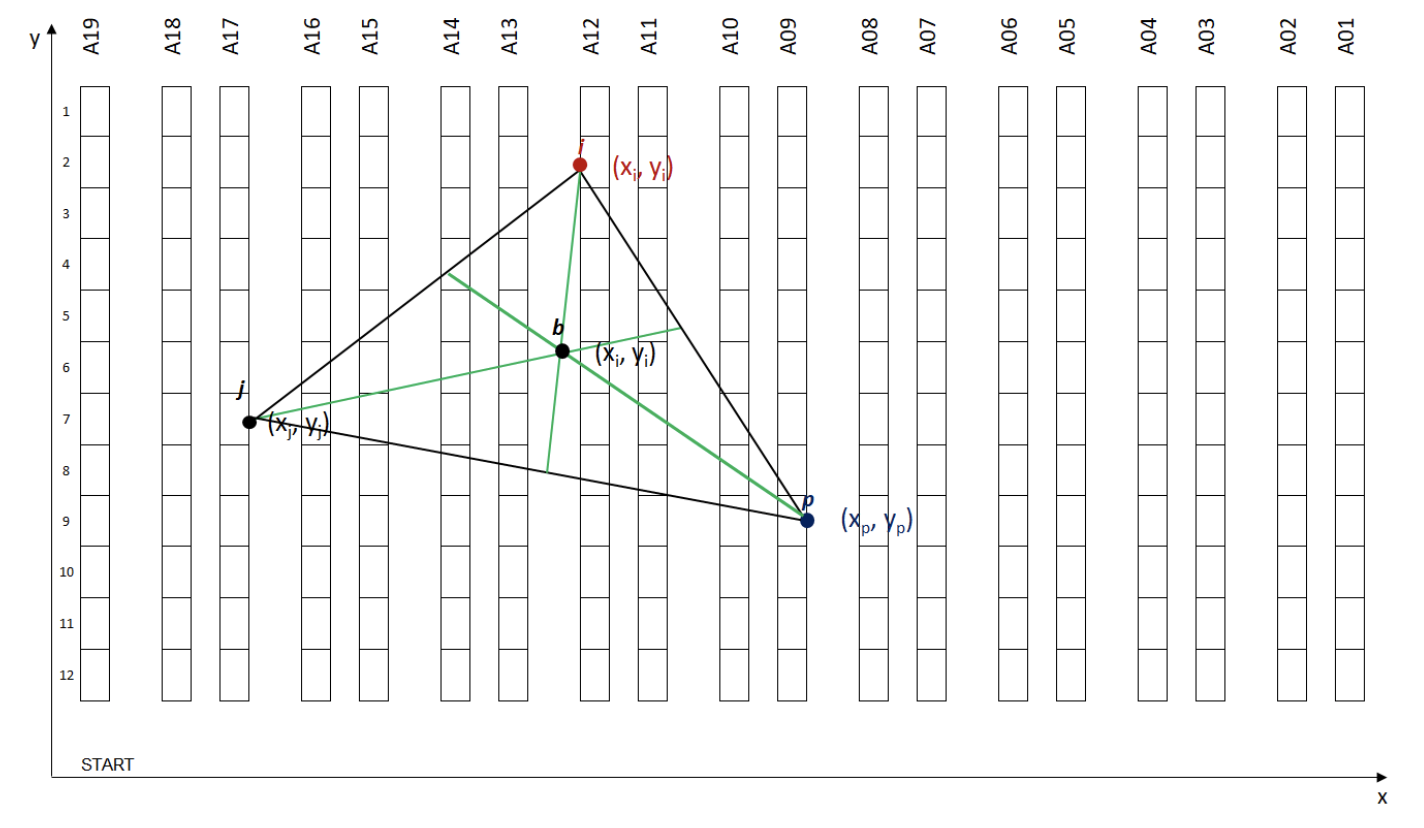
**return** dict\_map, dict\_omap, df, wave\_max

* 1. ***Picking locations clustering for Multi-line Orders***

**Funzione:** Centroide per ogni multi-line order

A differenza degli ordini a riga singola, gli ordini a più righe (multi-line) possono coprire diverse picking locations. Tuttavia, possiamo applicare la stessa metodologia applicata ai centroidi delle storage locations.

**Esempio**: Ordine con 3 linee coperto da 3 differenti picking locations



*Figura 7: Centroide di tree Picking Locations*

**Codice Python:**

**def** centroid(list\_in):

'''Centroid function'''

x, y = [p[0] **for** p **in** list\_in], [p[1] **for** p **in** list\_in]

centroid = [round(sum(x) / len(list\_in),2), round(sum(y) / len(list\_in), 2)]

**return** centroid

**def** centroid\_mapping(df\_multi):

'''Mapping Centroids'''

*# Mapping multi*

df\_multi['Coord'] = df\_multi['Coord'].apply(literal\_eval)

*# Raggrupo le coordinate per ordine*

df\_group = pd.DataFrame(df\_multi.groupby(['OrderNumber'])['Coord'].apply(list)).reset\_index()

*# Calcolo i Centroid*

df\_group['Coord\_Centroid'] = df\_group['Coord'].apply(centroid)

*# Definisco un dizionario per il mapping*

list\_order, list\_coord = list(df\_group.OrderNumber.values), list(df\_group.Coord\_Centroid.values)

dict\_coord = dict(zip(list\_order, list\_coord))

*# Final mapping*

df\_multi['Coord\_Cluster'] = df\_multi['OrderNumber'].map(dict\_coord).astype(str)

df\_multi['Coord'] = df\_multi['Coord'].astype(str)

**return** df\_multi

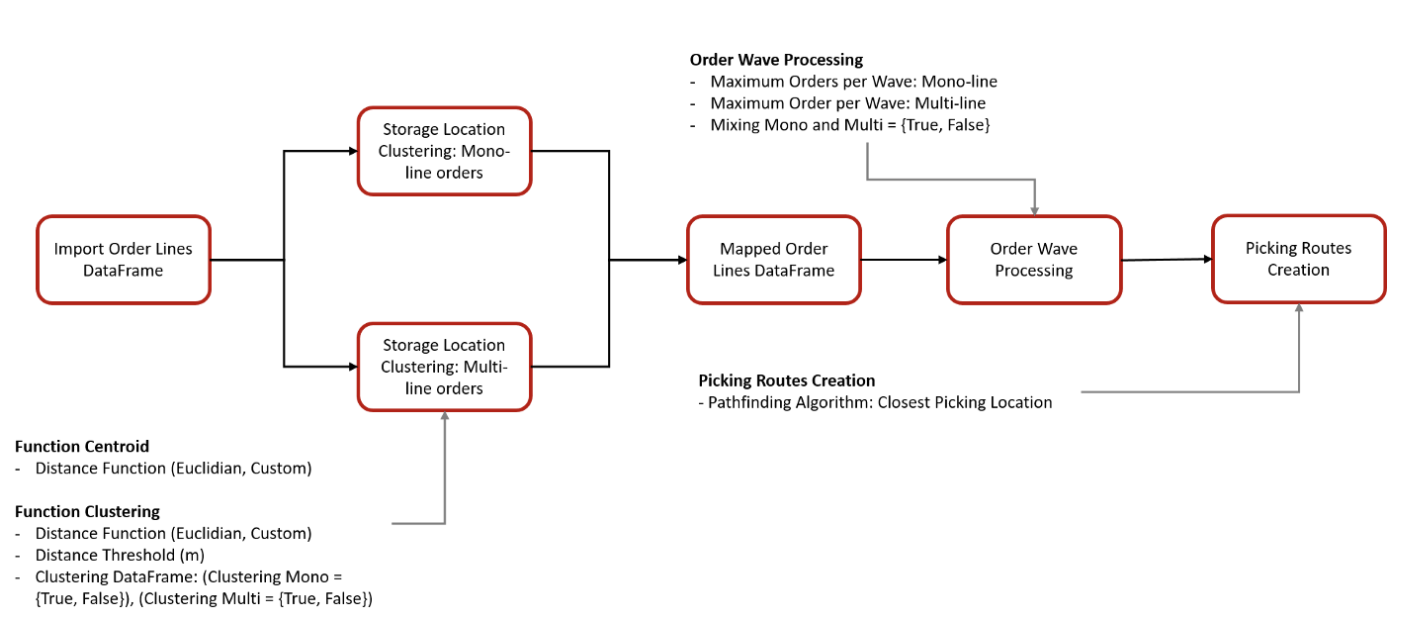
Dopo aver utilizzato questa funzione, si torna alla situazione degli ordini a riga singola con un singolo punto (x, y) per ordine.

Tornando a quel contesto, si può, quindi, applicare il clustering a questi punti cercando di raggruppare l’oridne per zona geografica con la condizione di massima distanza.

1. ***Model Simulation***

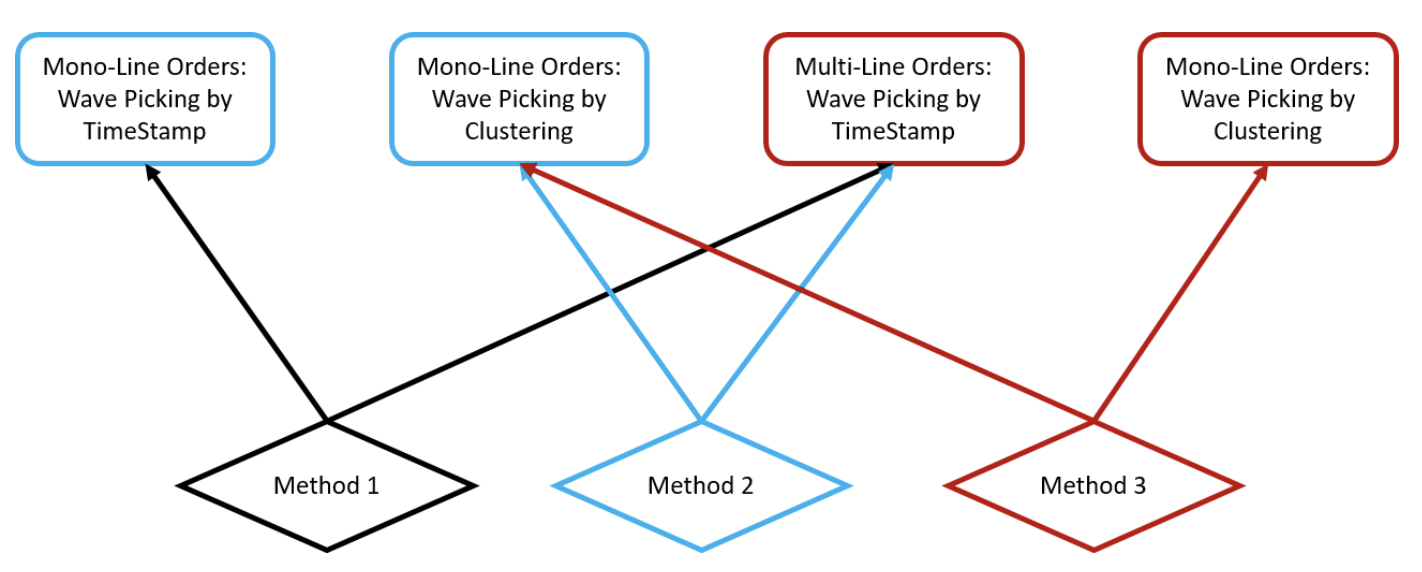
Per riassumere, la costruzione del modello, riportata nel grafico sottostante, si hanno diversi steps prima di scegliere la creazione di percorsi utilizzando l'elaborazione delle onde.

Ad ogni passaggio, abbiamo un set di parametri che possono essere regolati per migliorare le prestazioni:



*Figura 8: Model Costruction with Parameters*

* 1. ***Confronto fra 3 metodi di Wave Processing***



*Figura 9: 3 Metodi per Wave Processing*

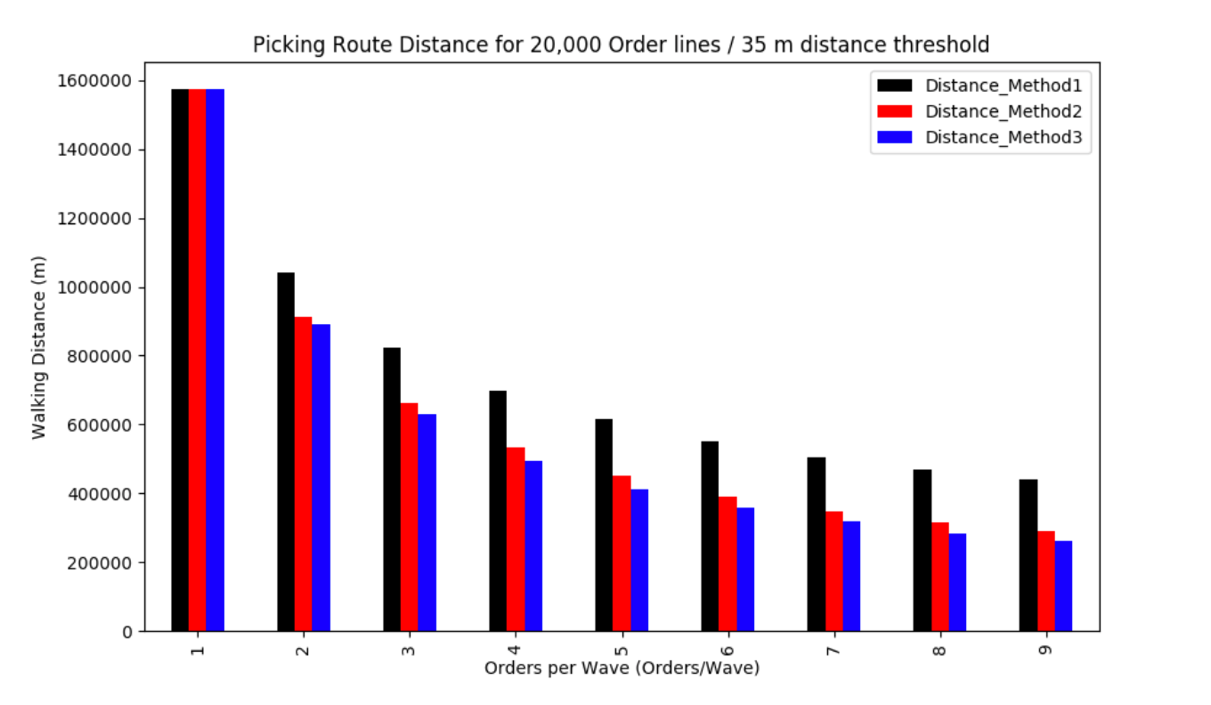
Quello che si va a fare è valutare l’impatto dell’elaborazione Order Wave by clusters di picking locations sulla walking distance totale.

Questo viene fatto testando tre diversi metodi:

* **Metodo 1**: non si applica il clustering (es. Scenario iniziale);
* **Metodo 2**: si applica il clustering solo agli ordini a Riga Singola;
* **Metodo 3**: si applica il clustering agli ordini a Riga Singola e ai Centroidi degli ordini multilinea.

Scenario della Simulation:

* **Order Lines**: 20,000 Lines;
* **Distance Threshold**: Distanza Massima fra 2 picking locations (distance\_threshold = 35 m);
* **Orders per Wave**: orders\_number in [1,9].

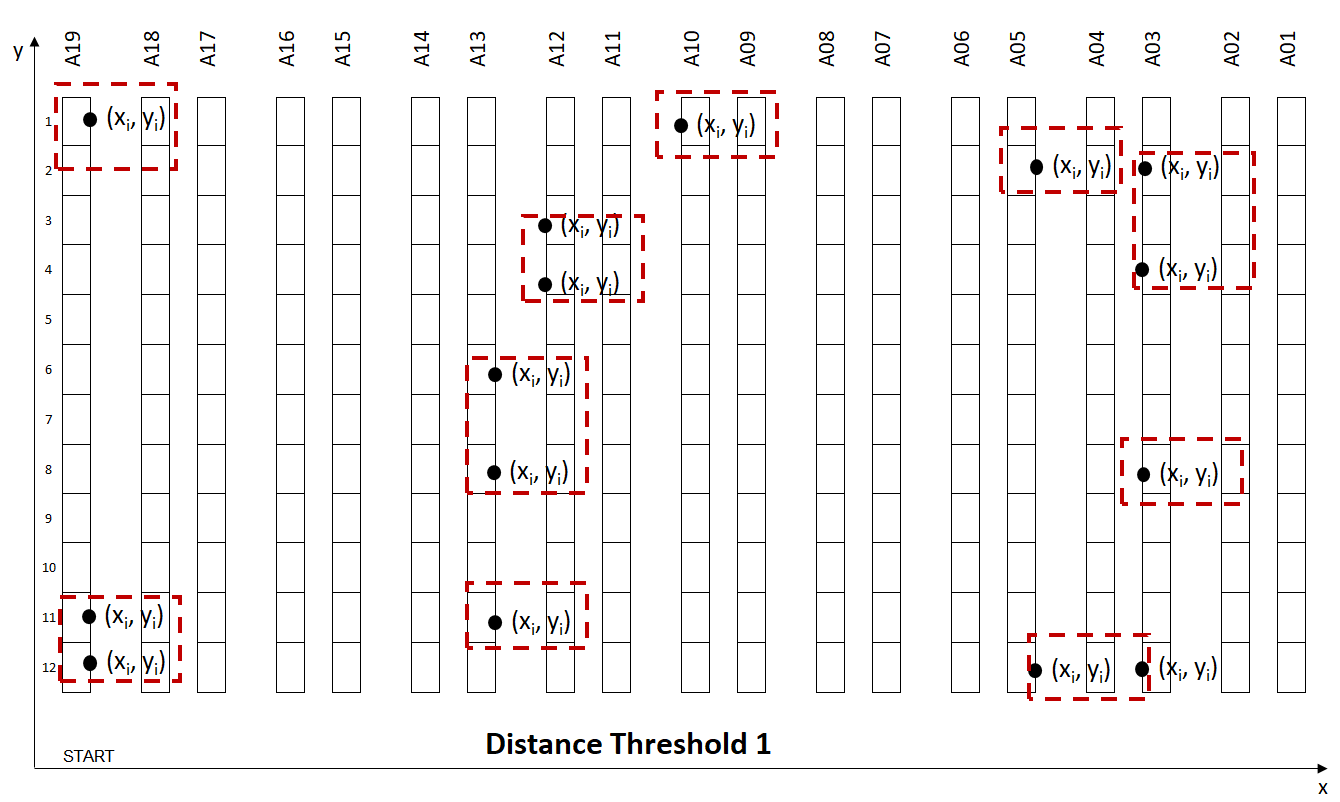


*Figura 10: Test 1:20.000 Order Lines/35 m distance Threshold*

Risultati:

* **Best** **Performace**: Metodo 3 per 9 ordini/Wave con riduzione dell'83% della walking distance.
* **Metodo 2 vs. Metodo 1**: Clustering per ordini mono-linea riduce la walking distance del 34%;
* **Metodo 3 vs. Metodo 2**: Clustering per ordini mono-linea riduce la walking distance del 10%.
  1. ***Tuning Distance Threshold for Clustering***

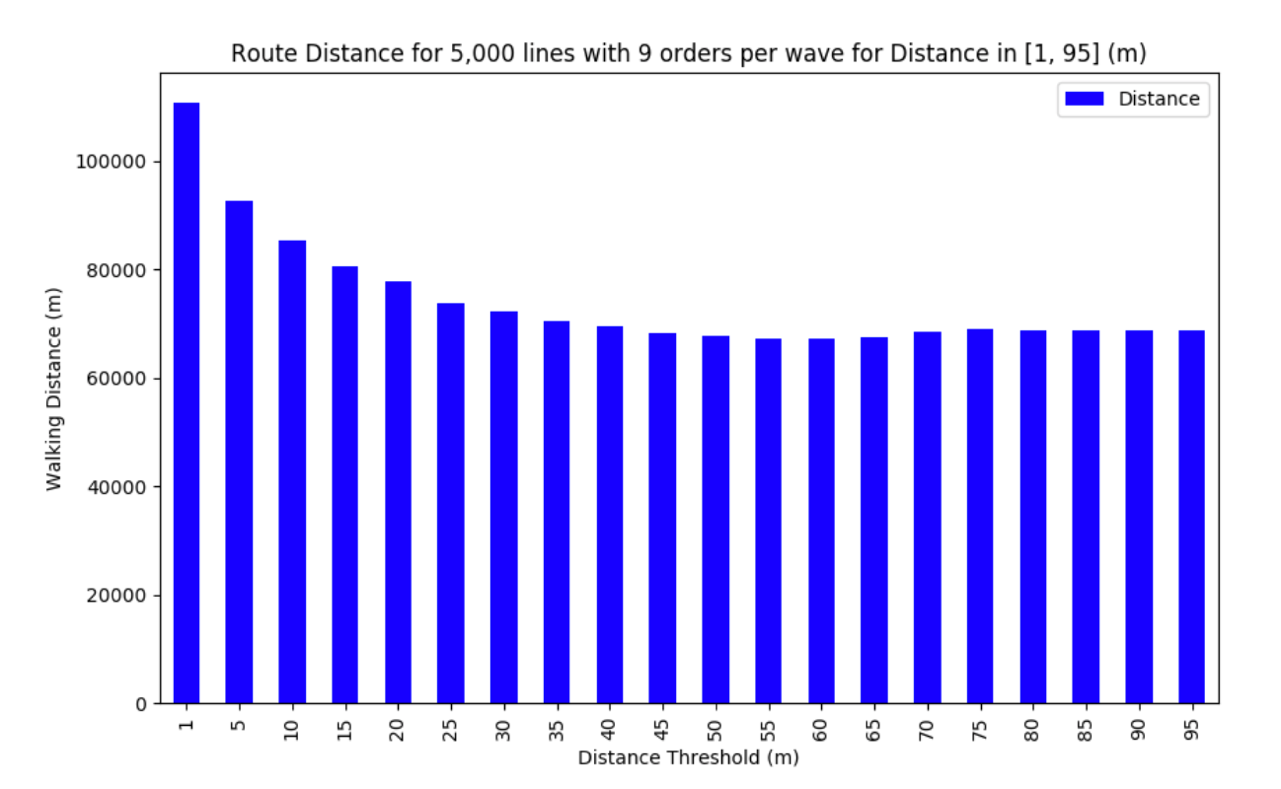
La prima ipotesi che viene fatta è che il Metodo 3 è il miglior scenario (20.000 order lines, 35 m Distance Threshold). Quindi, si va, di seguito, ad osservare la Distance Threshold sulla Walking Distance totale.



*Figura 11: Differenti distance threshold per Picking Location Clustering*

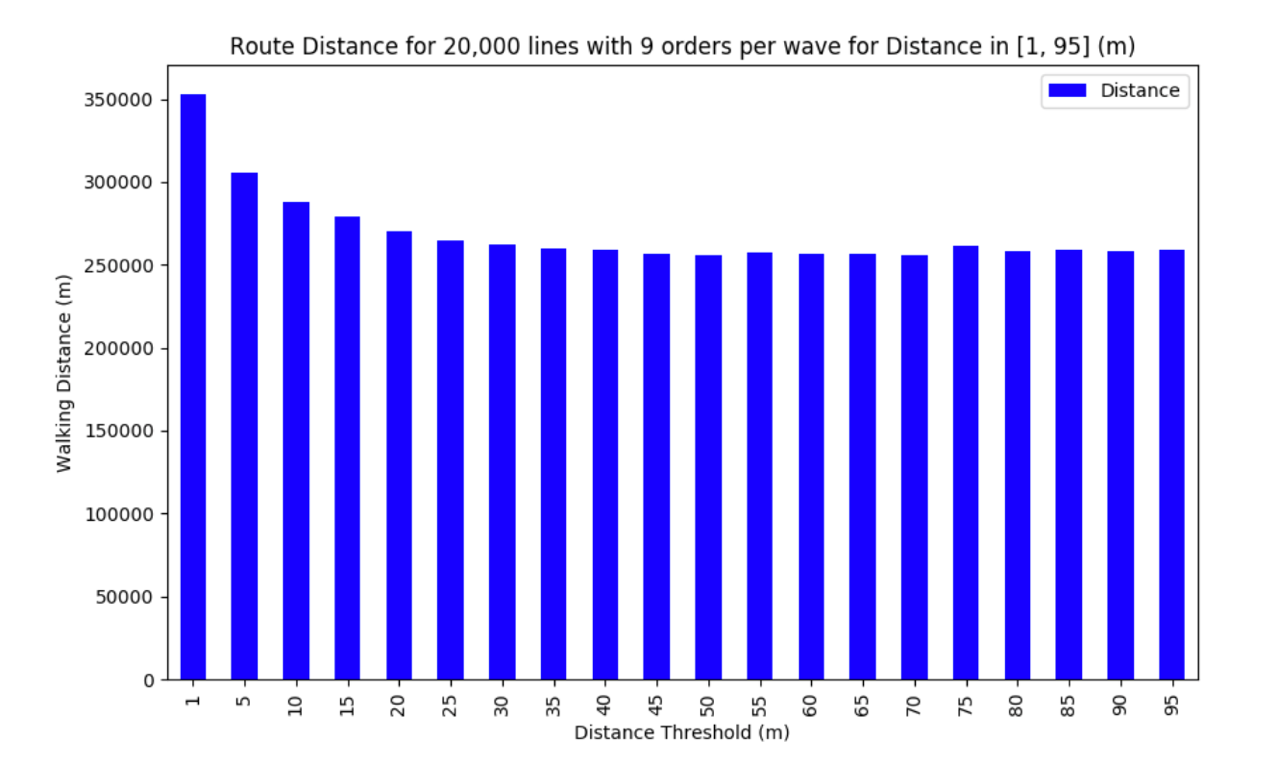
Quando il trade-off fra Walking Distance per due locations e la Wave Size è caratterizzato da:

* **Low** **Distance**: La walking distance fra due locations è bassa ma si hanno meno ordini per wave (più waves);
* **High** **Distance**: La walking distance fra due locations è alta ma si hanno molti ordini per wave (meno waves).



*Figura 12: Risultati per 5.000 linee raggruppate in Waves di 9 ordini con una distance threshold in [1,95]*

Si può osservare un minimo locale, per Distance\_Threshold = 60 m, dove la distanza è ridotta del 39% rispetto alla Distance\_Threshold = 1 m.



*Figura 12: Risultati per 20.000 linee raggruppate in Waves di 9 ordini con una distance threshold in [1,95]*

Si può osservare un minimo locale, per Distance\_Threshold = 50 m, dove la distanza è ridotta del 27% rispetto alla Distance\_Threshold = 1 m.

1. ***Prossimi Steps***

Il prossimo passo è la creazione del **Picking Route**, cioè per una lista di Picking Locations da coprire, come si può trovare il percorso migliore riducendo al minimo la walking distance?