

Τεχνικές Εξόρυξης Δεδομένων

Εργασία 2

- Νεαμονίτης Ευάγγελος, Α.Μ.: 1115201400123
- Ορφανίδης Ελευθέριος, Α.Μ.: 1115201400133

Αρχεία:

1. Erwthma1.ipynb
2. Erwthma2_A1.ipynb
3. Erwthma2_A2.ipynb
4. Erwthma3.ipynb
5. Οι 3 φάκελοι :
 - a. erwthma1
 - b. erwthma2_A1
 - c. erwthma2_A2

Περιέχουν τα αρχεία html που δημηουργήθηκαν από την εκτέλεση των αρχείων κώδικα.

Προεπεξεργασία δεδομένων:

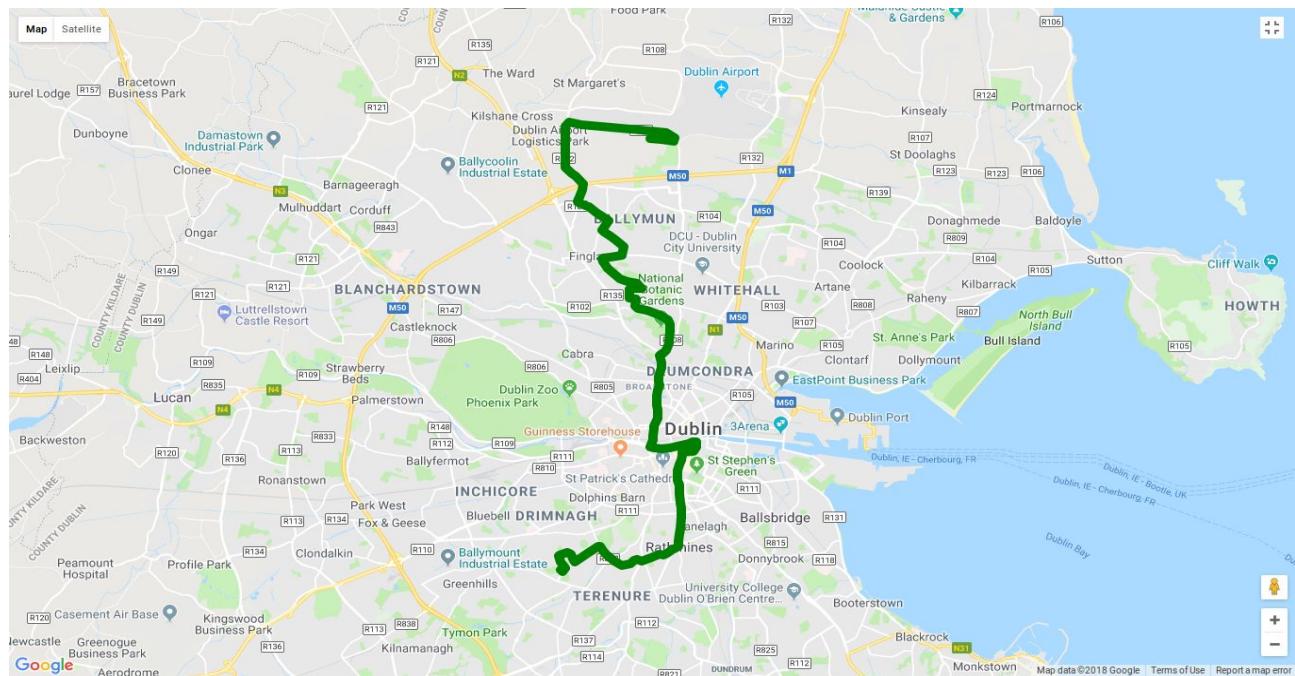
- Διαβάζουμε τα αρχεία με την συνάρτηση pd.read_csv()
- Αποθηκεύουμε στις μεταβλητές Trainnr και Testnr τα Trajectories κάθε στοιχείου των train_set και test_set αντίστοιχα, αφού έχουμε αφαιρέσει πρώτα τον χρόνο από την ακολουθία των σημείων της τροχιάς. Οπότε κάθε στοιχείο των μεταβλητών είναι της μορφής: [[lon0,lat0],.... [lonX,latX]]
- Αποθηκεύουμε στην μεταβλητή Ids όλα τα JourneyPatternId του train_set για δική μας ευκολία.

Ερώτημα 1:

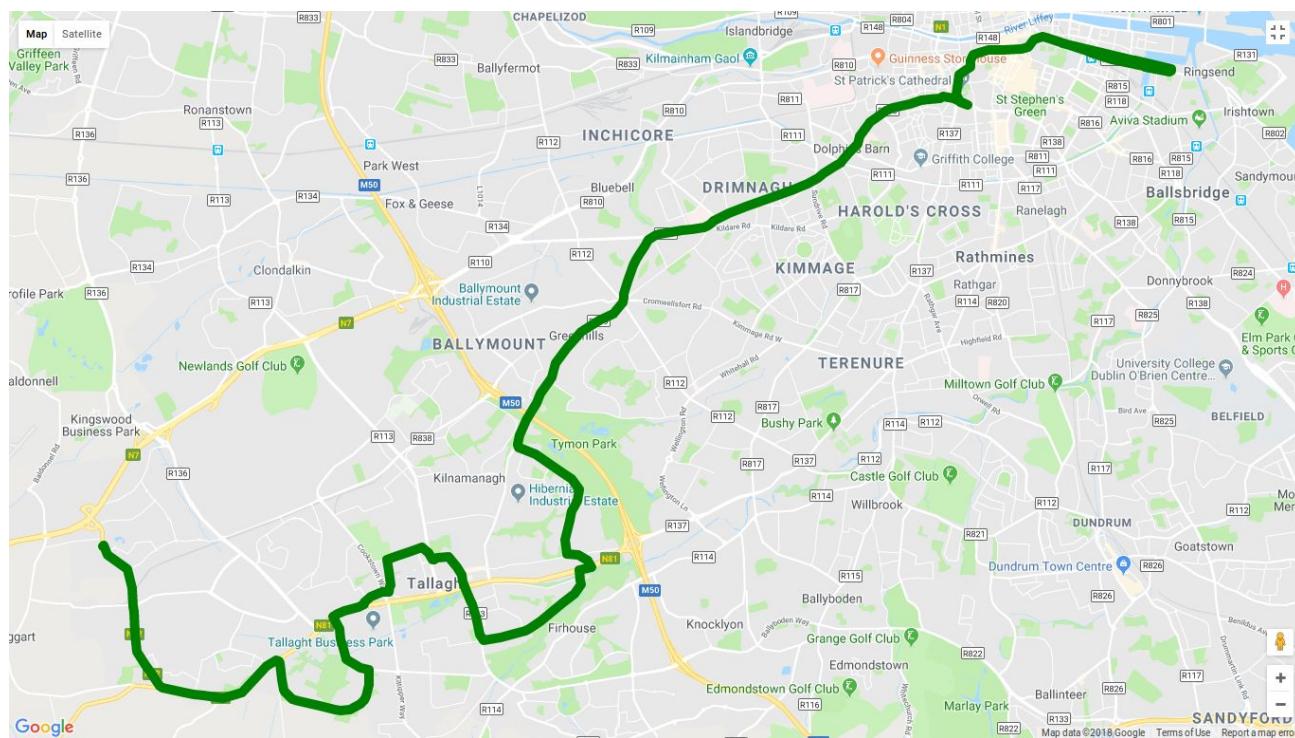
Τα αρχεία εξόδου για το ερώτημα αυτό βρίσκονται στον φάκελο erwthma1.

Οι διαδρομές που οπτικοποιήσαμε είναι οι παρακάτω:

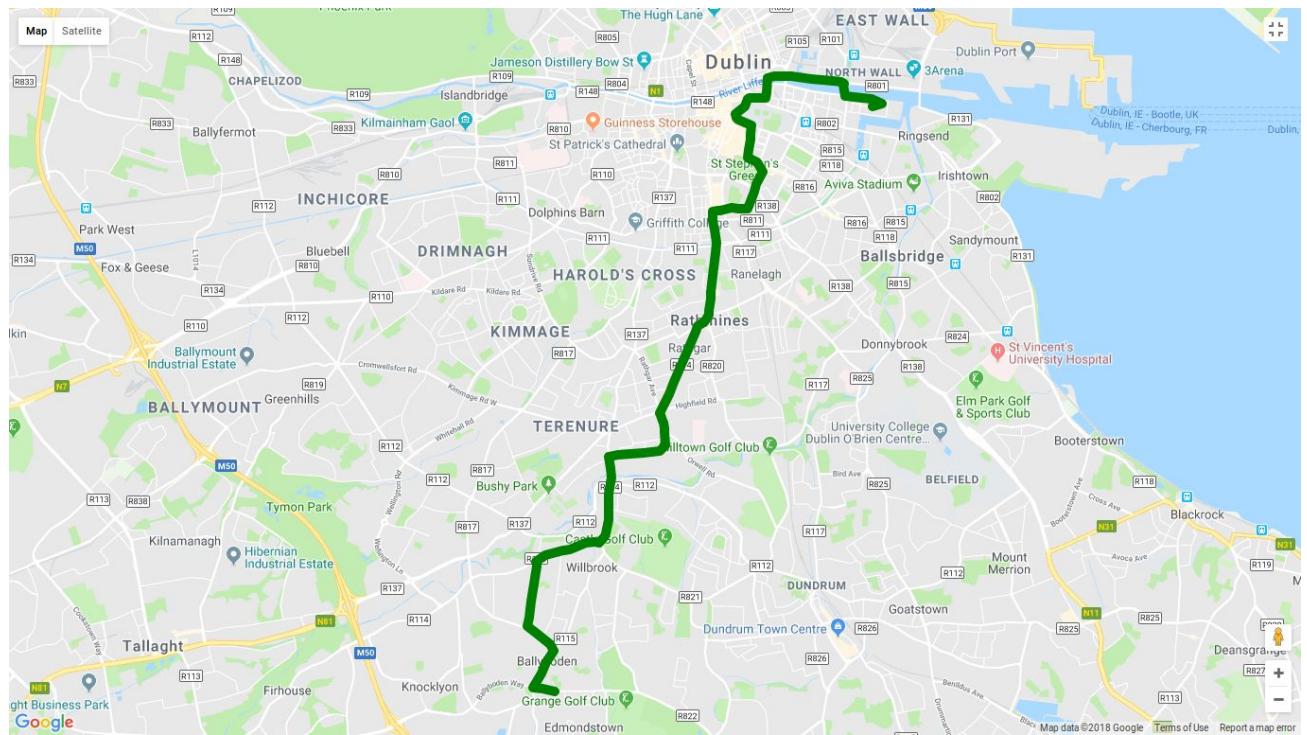
083A1001:



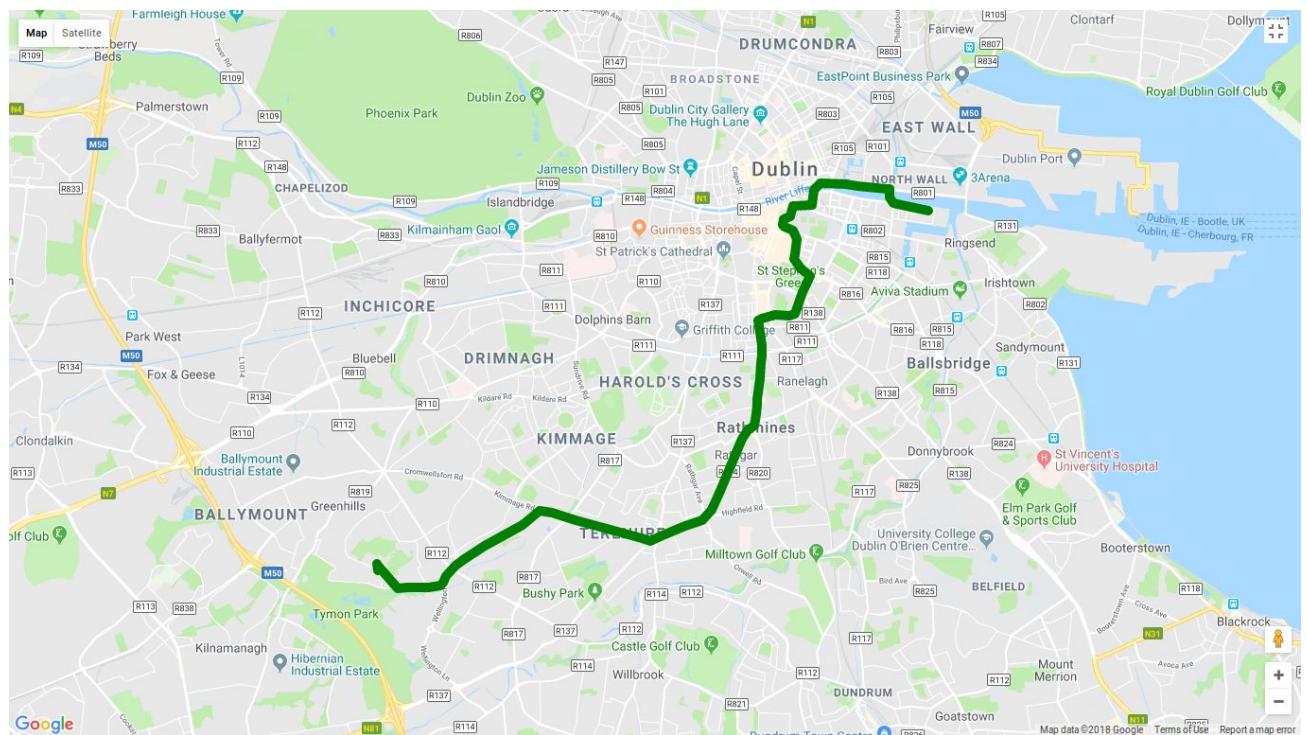
077A0001:



015B1002:



015A1001:



Ερώτημα Α1

Η υλοποίηση του αλγορίθμου DTW που χρησιμοποιήσαμε είναι η fast dtw που μειώνει την υψηλή πολυπλοκότητα του αλγορίθμου σε $O(N)$. Η υλοποίηση προέρχεται από τον παρακάτω σύνδεσμο :

<https://pypi.org/project/fastdtw/>

και η εντολή εγκατάστασης:

pip install fastdtw:

Η υλοποίηση μας της haversine distance είναι από τον παρακάτω σύνδεσμο:

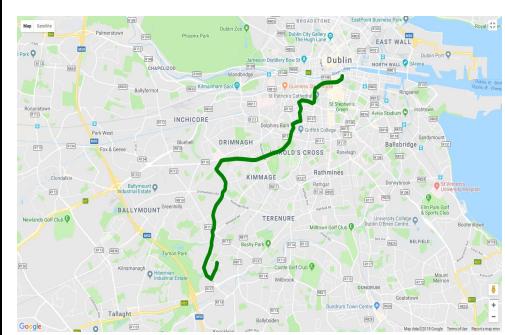
<https://stackoverflow.com/questions/4913349/haversine-formula-in-python-bearing-and-distance-between-two-gps-points>

Η μόνη αλλαγή που κάναμε στην haversine distance είναι στα ορίσματα καθώς έπαιρνε (lon1, lat1, lon2, lat2) ενώ η υλοποίηση της fastdtw έστελνε τα ορίσματα ως A,B όπου A=(lon1,lat1) και B=(lon2,lat2).

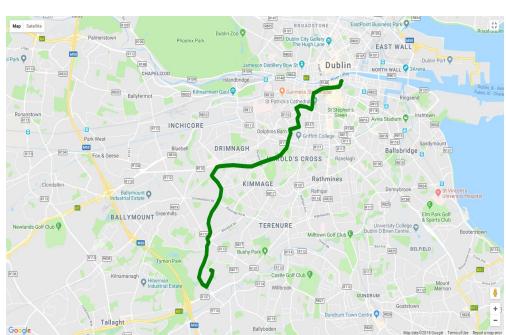
Το ΔΤ που παρουσιάζεται είναι ο χρόνος υπολογισμού και των 5 κοντινότερων γειτόνων του κάθε Test Trip.

Τελικά τα αποτελέσματα που πήραμε για το test_set_a1.csv είναι τα παρακάτω:

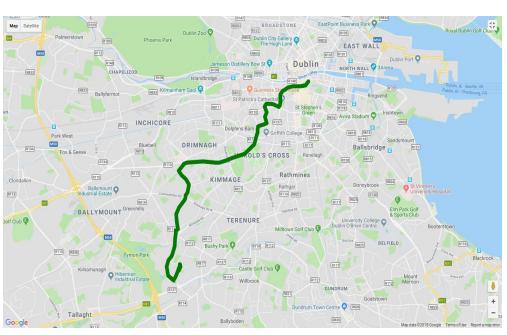
Test Trip 1



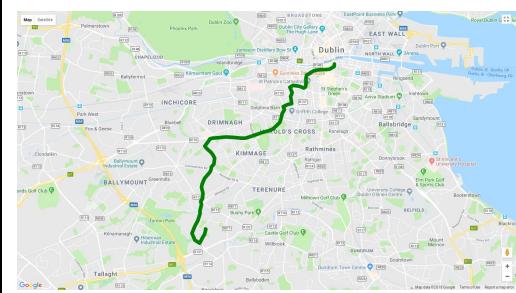
Test Trip 1
 $\Delta t = 69.78 \text{ sec}$



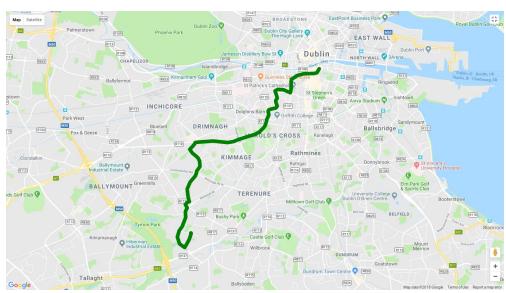
Neighbor 1
JP_ID: 01500001
DTW: 3.79km



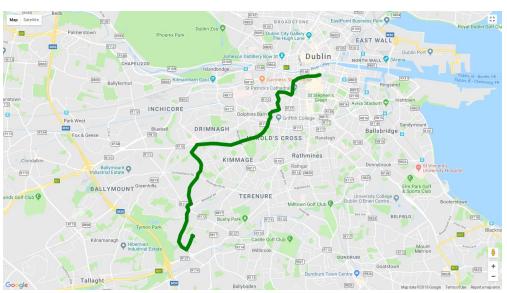
Neighbor 2
JP_ID: 01500001
DTW: 3.99km



Neighbor 3
JP_ID: 01500001
DTW: 0.0km

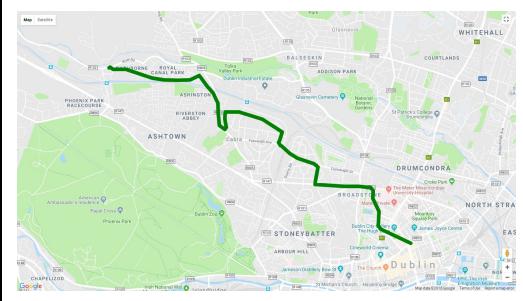


Neighbor 4
JP_ID: 01500001
DTW: 3.51km

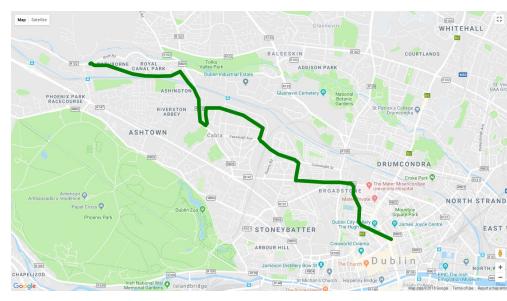


Neighbor 5
JP_ID: 01500001
DTW: 4.11km

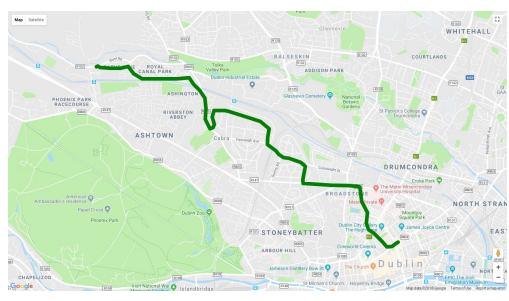
Test Trip 2



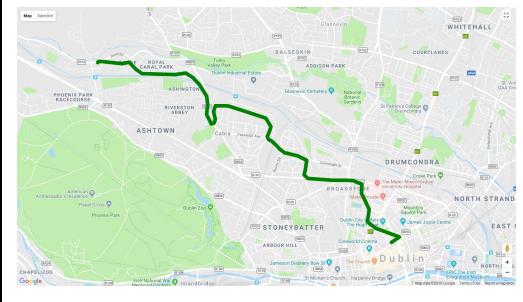
Test Trip 2
 $\Delta t = 50.44 \text{ sec}$



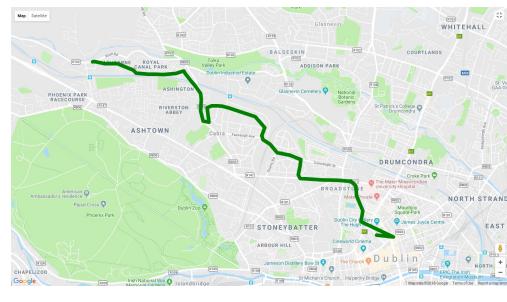
Neighbor 1
JP_ID: 01200001
DTW: 0.0km



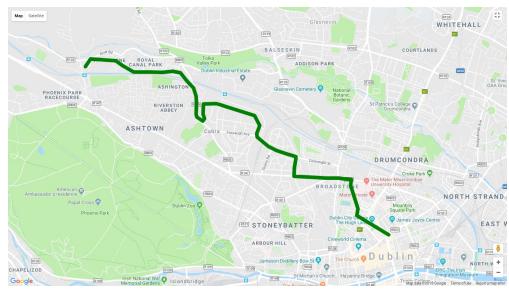
Neighbor 2
JP_ID: 01200001
DTW: 3.48km



Neighbor 3
JP_ID: 01200001
DTW: 3.37 km

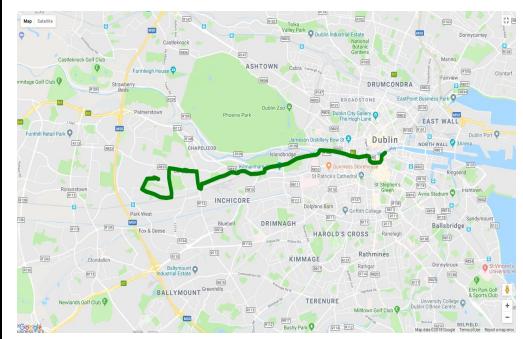


Neighbor 4
JP_ID: 01200001
DTW: 2.79km

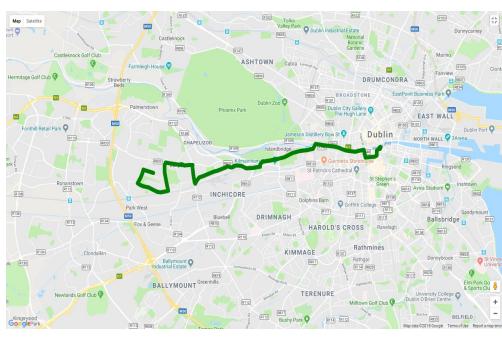


Neighbor 5
JP_ID: 01200001
DTW: 3.88km

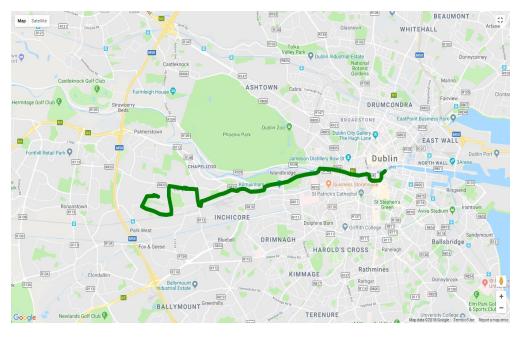
Test Trip 3



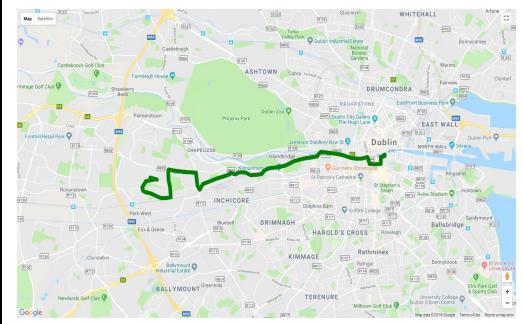
Test Trip 3
 $\Delta t = 66.27 \text{ sec}$



Neighbor 1
JP_ID: 00790001
DTW: 4.70m



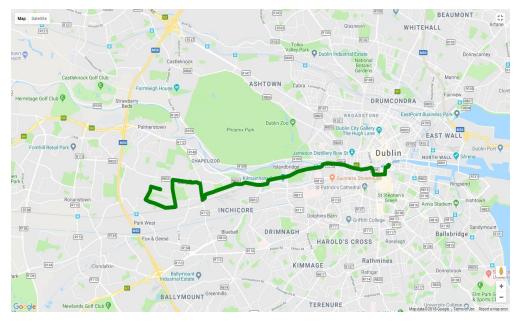
Neighbor 2
JP_ID: 00790001
DTW: 0.0km



Neighbor 3
JP_ID: 00790001
DTW: 4.87km

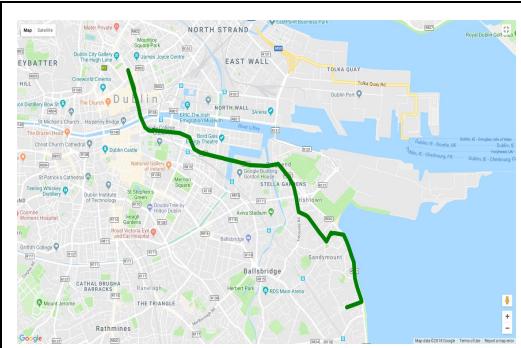


Neighbor 4
JP_ID: 00790001
DTW: 4.80km

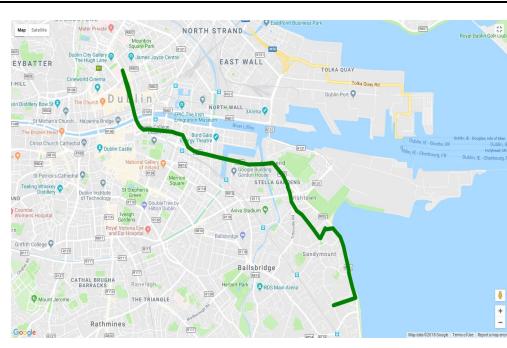


Neighbor 5
JP_ID: 00790001
DTW: 4.77km

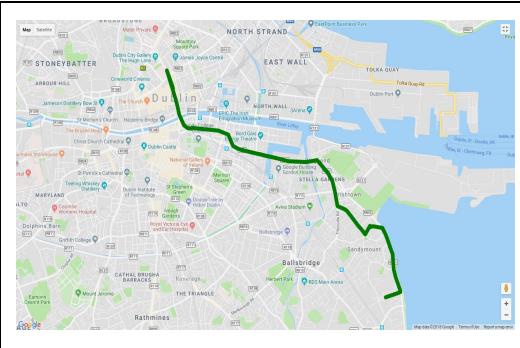
Test Trip 4



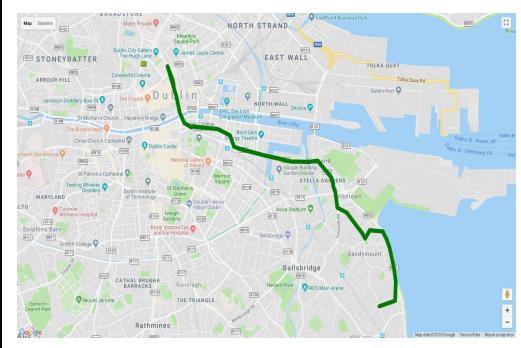
Test Trip 4
 $\Delta t = 55.02 \text{ sec}$



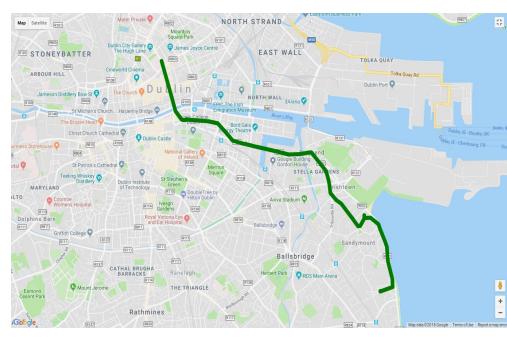
Neighbor 1
JP_ID: 00010002
DTW: 2.85km



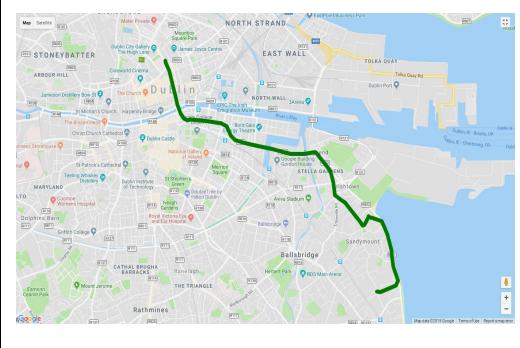
Neighbor 2
JP_ID: 00010002
DTW: 0.0km



Neighbor 3
JP_ID: 00010002
DTW: 2.46km

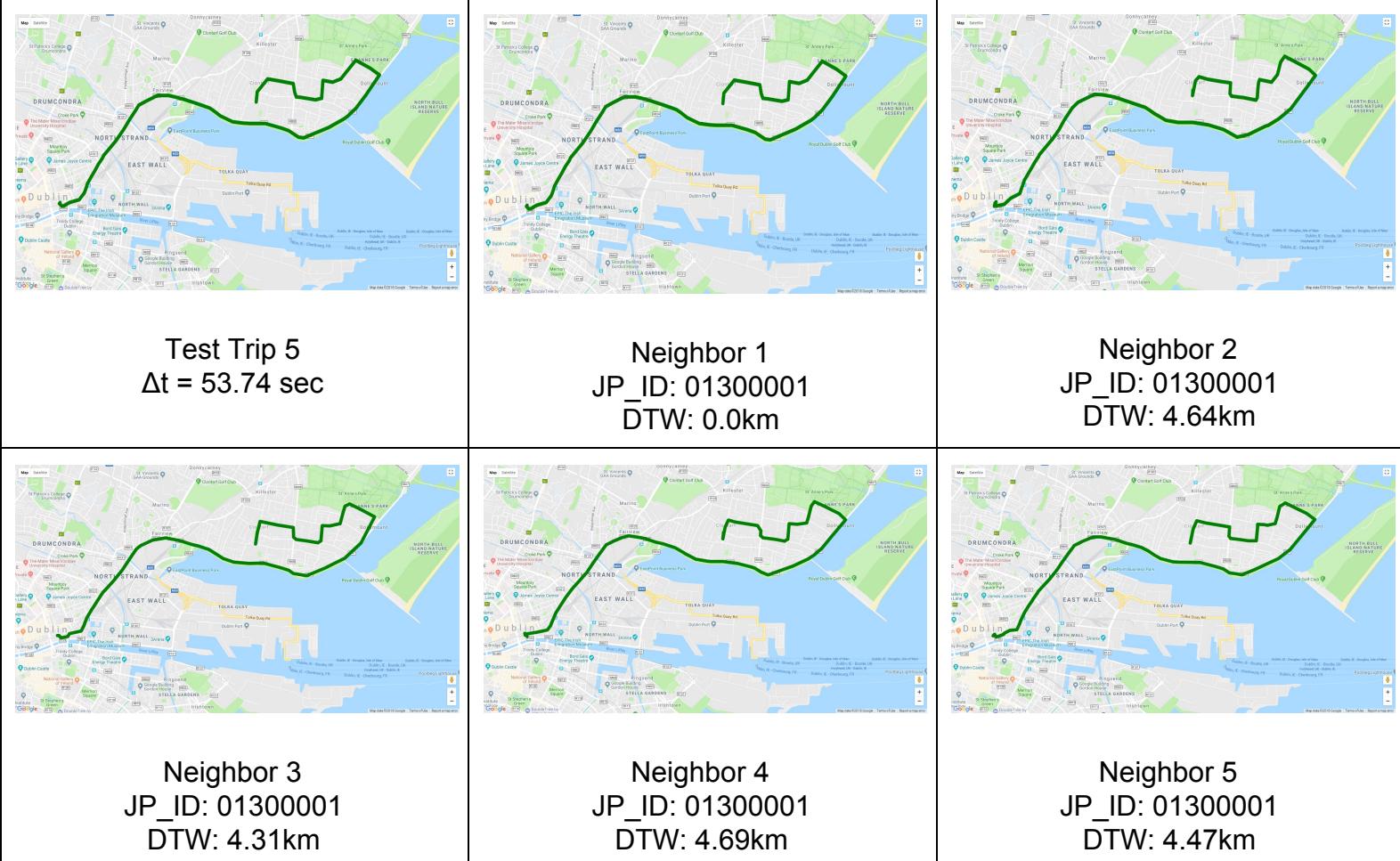


Neighbor 4
JP_ID: 00010002
DTW: 3.21km



Neighbor 5
JP_ID: 00010002
DTW: 3.46km

Test Trip 5



Ερώτημα A2

Η υλοποίηση του αλγορίθμου LCSS είναι τον παρακάτω σύνδεσμο με αλλαγές ώστε να ταιριάζει στα δεδομένα μας:

<https://gis.stackexchange.com/questions/131719/longest-common-subsequence-for-trajectory-matching-in-python>

Τα κομμάτια που χρησιμοποιήθηκαν από τον παραπάνω σύνδεσμο είναι τα:

- Computing the length of the LCS
- Reading out an LCS

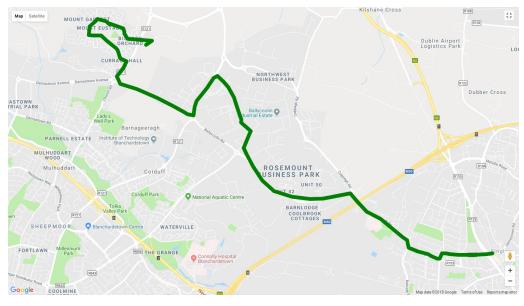
Οι αλλαγές που κάναμε είναι οι εξής:

-Στη δημιουργία του πίνακα, δυο σημεία θεωρούνται κοινά αν απέχουν με βάση την haversine distance 200 μέτρα

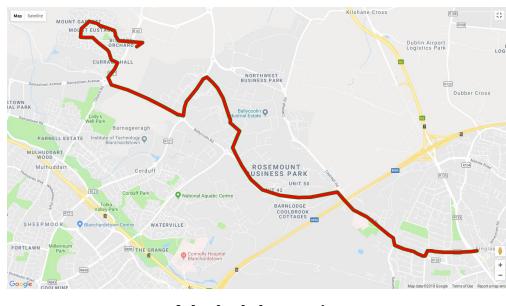
-Και μετατρέξαμε τον τύπο που επιστρέφει η backtrack από string σε list ώστε να παίρνουμε το path.

Τελικά τα αποτελέσματα που πήραμε για το test_set_a2.csv είναι τα παρακάτω:

Test trip 1



Test Trip 1
 $\Delta t = 388.48$ sec



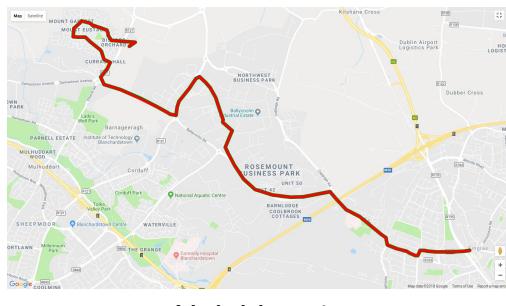
Neighbor 1
JP_ID: 040D1002
#Matching Points: 78



Neighbor 2
JP_ID: 040D1002
#Matching Points: 76



Neighbor 3
JP_ID: 040D1002
#Matching Points: 75

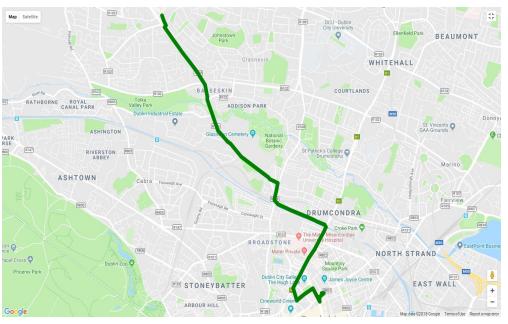


Neighbor 4
JP_ID: 040D1002
#Matching Points: 78

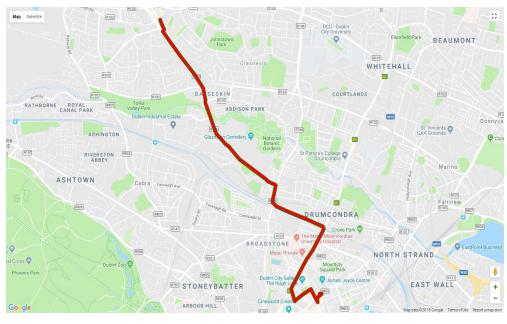


Neighbor 5
JP_ID: 040D1002
#Matching Points: 76

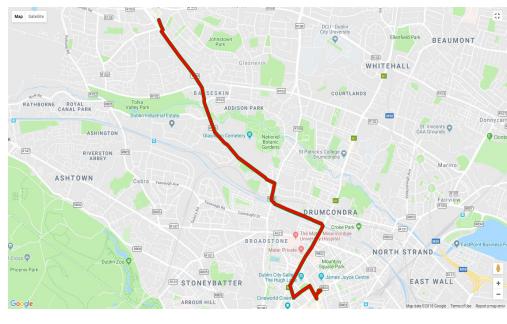
Test Trip 2



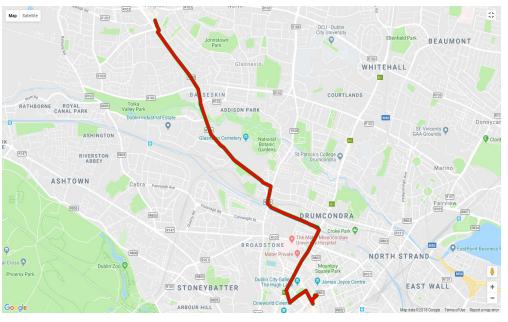
Test Trip 2
 $\Delta t = 403.15$ sec



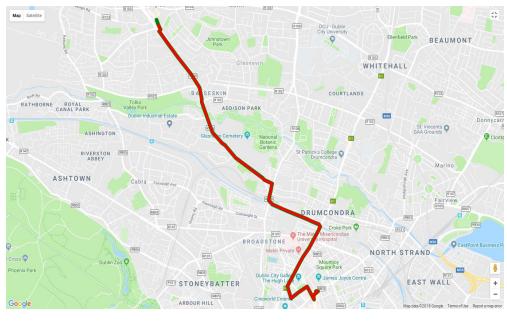
Neighbor 1
JP_ID: 040D1002
#Matching Points: 73



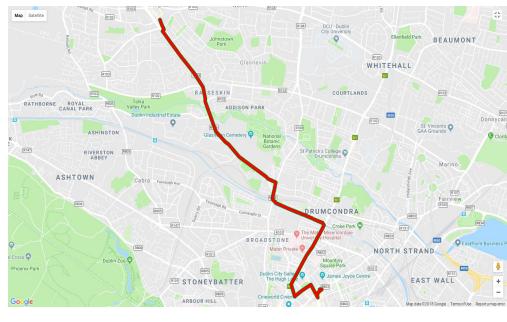
Neighbor 2
JP_ID: 040D1002
#Matching Points: 78



Neighbor 3
JP_ID: 040D1002
#Matching Point: 74

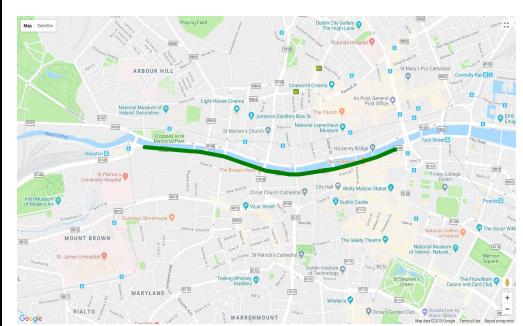


Neighbor 4
JP_ID: 040D1002
#Matching Points: 75

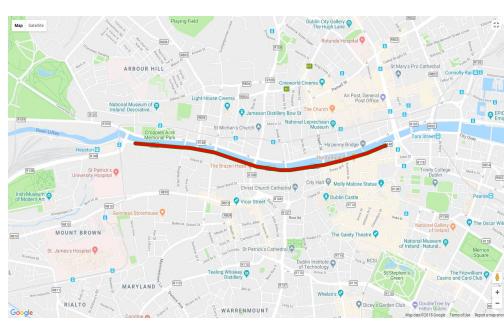


Neighbor 5
JP_ID: 040D1002
#Matching Points: 82

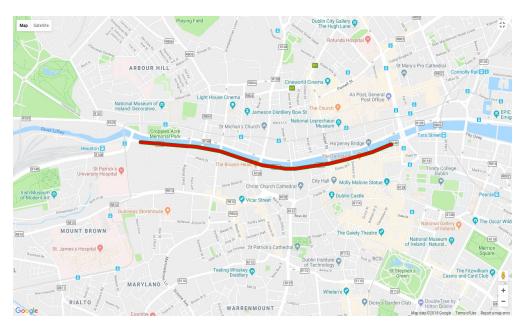
Test Trip 3



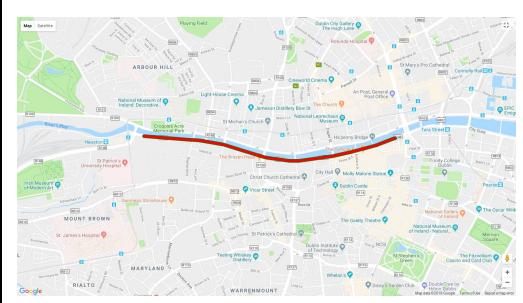
Test Trip 3
 $\Delta t = 200.93$ sec



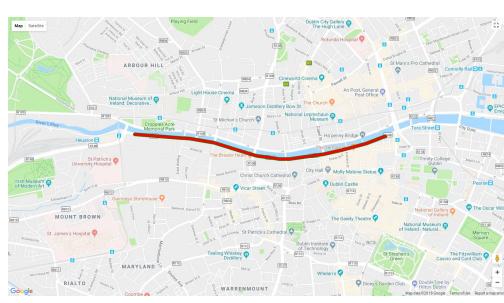
Neighbor 1
JP_ID: 01451001
#Matching Points: 40



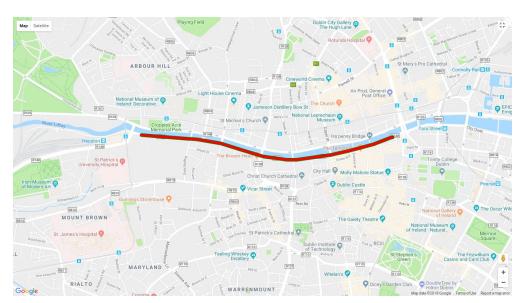
Neighbor 2
JP_ID: 00790001
#Matching Points: 40



Neighbor 3
JP_ID: 067X0001
#Matching Point: 40

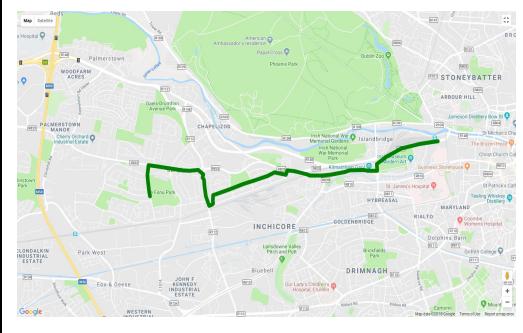


Neighbor 4
JP_ID: 079A0001
#Matching Points: 40



Neighbor 5
JP_ID: 01451008
#Matching Points: 40

Test Trip 4



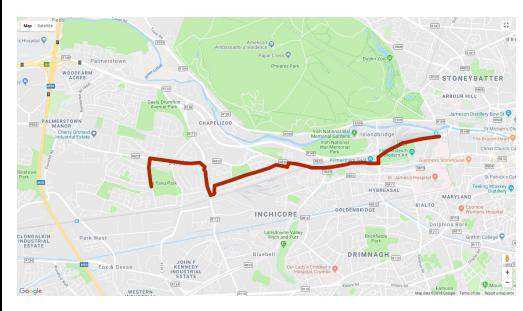
Test Trip 4
 $\Delta t = 300.68 \text{ sec}$



Neighbor 1
JP_ID: 00790001
#Matching Points: 59



Neighbor 2
JP_ID: 00790001
#Matching Points: 59



Neighbor 3
JP_ID: 00790001
#Matching Point: 59

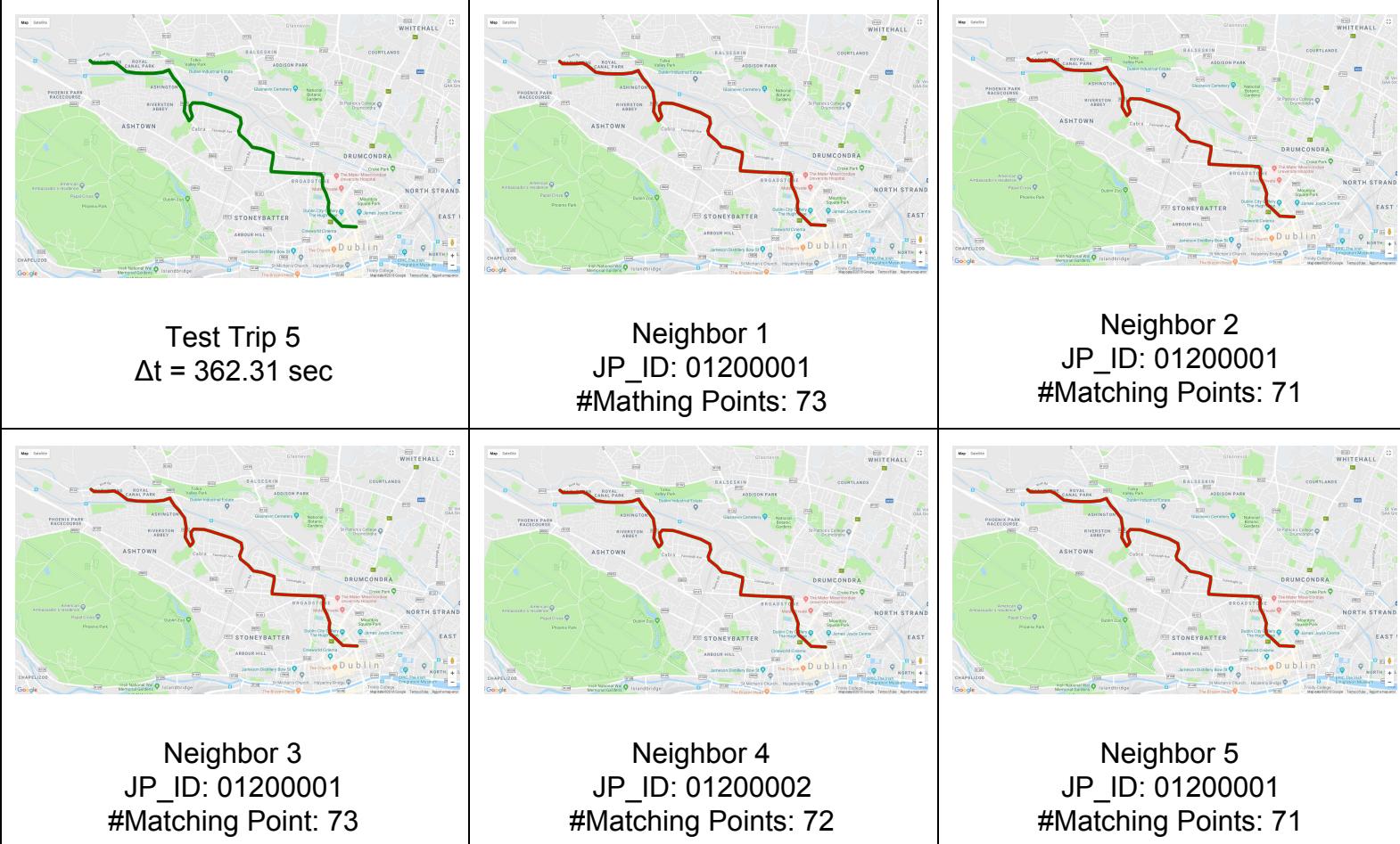


Neighbor 4
JP_ID: 00790001
#Matching Points: 59



Neighbor 5
JP_ID: 00790001
#Matching Points: 59

Test Trip 5



Ερώτημα 3

Ο αλγόριθμος Knn που χρησιμοποιούμε σε αυτό το ερώτημα είναι η υλοποίηση που είχαμε κάνει στην προηγούμενη εργασία μαζί με κάποιες παραλλαγές. Συγκεκριμένα οι παρακάτω συναρτήσεις δεν έχουν αλλαχτεί από την προηγούμενη εργασία:

__init__(), fit(), predict(). Η συνάρτηση find_Knn πλέον λειτουργεί ως εξής:

- Δέχεται ως είσοδο ένα στοιχείο του test_set (item)
- Για κάθε στοιχείο του train_set υπολογίζουμε την απόσταση του από το item με την χρήση της fastdtw (έχοντας ως συνάρτηση απόστασης την haversine)
 - Αν η απόσταση είναι μέσα στις k μικρότερες , τότε ανανεωνούμε τον πίνακα distances και τον πίνακα index (ο οποίος κρατάει τον αριθμό index για καθένα από τα k στοιχεία που βρίσκονται στον πίνακα distances)
- Αφού εξετάσουμε όλα τα στοιχεία, βρίσκουμε μέσω του πίνακα index τα JourneyPatternId των κοντινότερων διαδρομών στο item και κάνουμε majority voting επιστρέφοντας την τιμή του πιο κοινού Id.
- Ο αλγόριθμος επιστρέφει μια λίστα με ένα JourneyPatternId για κάθε στοιχείο του test_set .

10-fold cross validation

Επείδη στην προσπάθεια μας να κάνουμε cross validation με ολόκληρο το dataset παρατηρήσαμε μεγάλες καθυστερήσεις , αποφασίσαμε να το πραγματοποιήσουμε με το 1/10 του dataset. Τα αποτελέσματα του cv είναι τα παρακάτω:

Average Accuracy: 0.905
Elapsed time : 1715.05 sec.

Prediction

Τέλος κάνουμε fit στον αλγόριθμο ολόκληρο το train_set (Trainnr , Ids) και στην συνέχεια κάνουμε predict το Testnr εξάγοντας τα αποτελέσματα στο testSet_JourneyPatternIDs.csv .