ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ 2018 **ΘΕΜΑ 2**

Βρούτση Νίκη 03115639

Κυριάκου Αθηνά 03117405

7ο Εξάμηνο

Γενικός Σχεδιασμός Συστήματος:

Σε αυτή την εργασία κληθήκαμε να βελτιώσουμε την υπηρεσία εξυπηρέτησης πελατών ταξί που κατασκευάσαμε στο πλαίσιο της 1ης εργασίας. Στην πρώτη εργασία υλοποιήσαμε μια παραλλαγή του αλγορίθμου αναζήτησης A·. Στην παρούσα εργασία βελτιώσαμε την υπηρεσία ώστε να λειτουργεί με τον εξής τρόπο:
Η ικανότητα ενός ταξί να εξυπηρετήσει τον πελάτη καθορίζεται απο την **διαθεσιμότητα** του ταξί, **τον αριθμό των ατόμων** που μπορεί να μεταφέρει σε σχέση με τον αριθμό που ζητεί ο πελάτης, **τη γλώσσα** που μιλά ο οδηγός του ταξί σε σχέση με τη γλώσσα που δηλώνει ότι μιλά ο πελάτης και το αν το ταξί μπορεί να εξυπηρετήσει **μεταφορές μεγάλων αποστάσεων.**
Η υπηρεσία παράγει **δύο** αποτελέσματα.

- **Το πρώτο** κατατάσσει αρχικά τα διαθέσιμα ταξί που μπορούν να εξυπηρετήσουν τον πελάτη με βάση την απόστασή τους από τον πελάτη, προτιμώντας διαδρομές σε οδούς ταχείας κυκλοφορίας, με πολλές λωρίδες, χαμηλή κίνηση και φωτισμένους δρόμους στην περίπτωση που είναι νύχτα. Τα 3 πρώτα ταξί αυτής της κατάταξης παρουσιάζονται στον χρήστη, μαζί με το μήκος της διαδρομής έως τον πελάτη που έχει υπολογιστεί.

- **Το δεύτερο** δημιουργεί μια δεύτερη κατάταξη των 3 πρώτων ταξί, με βάση την αξιολόγηση που διαθέτει κάθε ταξί από τους χρήστες. Η δεύτερη αυτή κατάταξη παρουσιάζεται στον πελάτη παράλληλα με την πρώτη, ώστε ο πελάτης να μπορεί να κάνει την τελική του επιλογή.

1. Βασικά στοιχεία του κώδικα σε Java:

Οι βασικότερες classes που δημιουργήθηκαν για την υλοποίηση της υπηρεσίας είναι:

1. **Astar**: κλάση με τη μέθοδο AstarImplementation που υλοποιεί τον αλγόριθμο Astar για την υπηρεσία. Η ανάλυση του αλγορίθμου περιγράφεται παρακάτω.

2. **PrologParser**: κλάση υπεύθυνη για την επικοινωνία μεταξύ Java και Prolog

3. **Extended Node**: για κάθε κόμβο, η κλάση περιλαμβάνει τις συντεταγμένες του (x,y), εάν είναι ο κόμβος στόχος, την τιμή της ευριστικής και του βάρους του, την απόστασή του από την αφετηρία καθώς και ένα Arraylist με τους γειτονικούς σε αυτόν κόμβους

4. **Client**: κλάση με τις συντεταγμένες και το ID του πελάτη

5. **Taxis**: κλάση με τις συντεταγμένες, το ID και τη βαθμολογία του ταξί

6. **Address**: κλάση με τη μέθοδο parseAddress για την ανάγνωση του lines.csv αρχείου, τη διάκριση των πεδίων που δίνονται για κάθε οδό και την εισαγωγή στη βάση του Prolog κατηγορήματος

7. **Traffic**: κλάση με τη μέθοδο parseTraffic για την ανάγνωση του traffic.csv αρχείου, τη διάκριση των πεδίων που δίνονται αναφορικά με την κίνηση κάθε δρόμου και την εισαγωγή στη βάση του Prolog κατηγορήματος

8. **Map**: κλάση για την ανάγνωση του nodes.csv αρχείου, διάκριση των πεδίων και εισαγωγή στη βάση των Prolog κατηγορημάτων. Επιπλέον, δημιουργείται ο γράφος αναζήτησης στον οποίο εφαρμόζεται ο A* και αντιστοιχίζεται η θέση του πελάτη στον κοντινότερο χωροταξικά δυνατό κόμβο από αυτούς που δίνονται στο nodes.csv

9. **NextStates**: κλάση μέσω της οποίας ορίζεται το κλειστό σύνολο για να καθοριστούν οι κόμβοι τους οποίους έχει ήδη επισκεφθεί ο A*

2. Προεπεξεργασία των δεδομένων:

Κατά την ανάγνωση των .csv αρχείων, η προεπεξεργασία που γίνεται στα δεδομένα είναι:

- Αντιστοίχιση της θέσης του πελάτη στον κοντινότερο υπάρχοντα κόμβο που υπάρχει στον γράφο, ο οποίο δημιουργείται στην κλάση map.java βάσει του nodes.csv αρχείου (μέθοδος closestNode()).

- Κατά την ανάγνωση του αρχείου lines.csv στην κλάση Address.java χρησιμοποιείται η μέθοδος addCommas για να εισάγει κόμματα στα σημεία που λείπουν από το αρχείο και απαιτούνται για τη διάκριση των πεδίων του. Επιπλέον, αντικαθίστανται τα κενά πριν και μετά το κόμμα που ενδέχεται να υπάρχουν.

- Πριν την κλήση του αλγορίθμου A* για κάθε ταξί, αντιστοίχιση της θέσης του με τον κοντινότερο κόμβο στον γράφο.

3. Εφαρμογή λογικού προγραμματισμού στην παραλλαγή του A*

Από τα αρχεία εισόδου δημιουργούμε μία βάση με τα εξής γεγονότα σε Prolog:

1) Από το αρχείο lines.csv:

lineInfo(Id,Highway,Thename,Oneway,Lanes,Maxspeed,Railway,Boundary,Access,Natural,Barrier,Tunnel,Bridge,Incline, Waterway, Busway, Toll)

2) Από το αρχείο client.csv:
client(X,Y XDest,YDest,Time,Persons,Language,Luggage)

3) Από το αρχείο taxis.csv:
a) taxi(X,Y,Id,Available,Rating,LongDistance,Type)
b)speaksDriver(Id,Language)
c) minPassengers(Id,MinCapacity)
d) maxPassengers(Id,Maxapacity)

4) Από το αρχείο traffic.csv:

lineTraffic(Id ,StartTime,EndTime,TrafficLevel)

5) Από το αρχείο nodes.csv:

node(X,Y,LineId,Thename,LineCounter)

Η παραλλαγή του αλγορίθμου A* στηρίζεται σε 3 βασικούς κανόνες.

isQualifiedDriverForClient(DriverID): για το αν ο οδηγός ταξί είναι κατάλληλος

canMoveFromTo(NodeX, NodeY): δεδομενου του **isQualifiedDriverForClient** για το αν μπορούμε να μετακινηθούμε από έναν κόμβο x σε έναν γειτονικό του κόμβο y

weightFactor(Ax, Ay, Bx, By, Value): δεδομενου του **canMoveFromTo** για να υπολογίσουμε τον παράγοντα που θα πολλαπλασιάσουμε το κόστος για να μετακινηθούμε από έναν κόμβο σε έναν άλλο

Αρχικά, επιλέγουμε από τη βάση γεγονότων που έχουμε δημιουργήσει μόνο τους οδηγούς που πληρούν τα κριτήρια για να εξυπηρετήσουν τον πελάτη.
Τα κριτήρια αυτά είναι:
1) ο οδηγός να είναι διαθέσιμος
2) ο οδηγός να μιλάει την ίδια γλώσσα με τον πελάτη
3) ο αριθμός των επιβατών να χωράει στο ταξί
4) ο αριθμός των αποσκευών να χωράει στο ταξί

Στη συνέχεια, για κάθε οδηγό ταξί που είναι κατάλληλος για τον πελάτη, βρίσκουμε τις εναλλακτικές διαδρομές από τη θέση του ταξί στον πελάτη.

Για να βρουμε τη διαδρομή από το ταξί προς τον πελάτη:

- Ξεκινώντας από τον κόμβο που αντιστοιχεί στη θέση του ταξί, ελέγχουμε τους γείτονες του κομβου
- Για κάθε γείτονα του κόμβου, ελέγχουμε με τον κανόνα canMoveFromTo(NodeX, NodeY), αν είναι δυνατή η μετακίνηση προς τον γειτονικό κόμβο που εξετάζουμε.
- **Αν όχι,** επαναλαμβάνουμε τη διαδικασία για τον επόμενο γείτονα.
- **Αν ναι**, υπολογίζουμε το κόστος με τη συνάρτηση calculateFactor(NodeX, NodeY). Η συνάρτηση αυτή υπολογίζει το Value επί το οποίο θα πολλαπλασιάσουμε την τιμή της ευριστικής μεταξύ των δύο κόμβων που εξετάζουμε.Για να υπολογίσουμε το Value χρησιμοποιούμε τον κανόνα weightFactor(Ax, Ay, Bx, By, Value). Το συνολικό κόστος υπολογίζεται από τη σχέση totalCost = RouteCost() + weightFactor * neighbor.getWeight() και συνεπώς όσο μικρότερη η τιμή weightFactor, δηλαδή το Value που υπολογίζουμε, τόσο μικρότερο και άρα καλύτερο το κόστος διαδρομής.

Η τιμή **Value** επί την οποία πολλαπλασιάζεται το ευριστικό κόστος της ακμής, προκύπτει από το γινόμενο του επιμέρους κόστους των εξής παραμέτρων:

- **Βάρος** ανάλογα με το **είδος του δρόμου** (ταχείας κυκλοφορίας,κατοικήσιμη περιοχή κ.ο.κ): αναλογικά με το πόσο γρήγορα έως πόσο αργά μπορεί να κινηθεί το ταξί στο κάθε είδος δρόμου, έχουμε δημιουργήσει γεγονότα **penalty (RoadType, Cost)** όπου το εύρος τιμών του Cost κυμαίνεται από 0.3 (για αυτοκινητόδρομο) εως 0.9 (για κατοικήσιμη περιοχή).

Έχουμε λοιπόν τον **κανόνα**: roadTypeRank(LineID, Value) :- lineInfo(LineID, Type, _, _, _, _, _, _, _, _, _, _, _, _, _, _, _, _,_), penalty(Type, Score), Value = Score.

- **Βάρος** ανάλογα με την **κίνηση**: για τις 3 πιθανές καταστάσεις κίνησης high, medium, low έχουμε δημιουργήσει **3 γεγονότα** :

trafficRank(high, 1).
trafficRank(medium, 0.8).
trafficRank(low, 0.6).

και τον **κανόνα**: trafficRank(LineID, TRank) :- client(_, _, _, _, Time, _, _, _), lineTraffic(LineID, StartTime, EndTime, Value), StartTime =< Time, EndTime >= Time, trafficRank(Value, TRank).

όπου ανάλογα με την ώρα που ο πελάτης αναζητά ταξί και υπολογίζει την κίνηση στη συγκεκριμένη οδό.

- **Βάρος** ανάλογα με τον **αριθμό λωρίδων**: όσο περισσότερες λωρίδες, τόσο χαμηλότερο το κόστος. Το κόστος λωρίδων είναι 1- αριθμός_λωρίδων * 0.1 .

Το παραπάνω περιγράφεται από τον **κανόνα**:laneRank(LineID, LRank) :- lineInfo(LineID, _, _, _, _, Number, _, _, _, _, _, _, _, _, _, _, _, _, _), LRank is 1 - Number * 0.1.

- **Βάρος** ανάλογα με το αν είναι **φωτισμένος ο δρόμος**: αρχικά δημιουργούμε έναν κανόνα που καθορίζει τις βραδινές ώρες.

Δεδομένου ότι είναι νύχτα και ο δρόμος έχει φώτα (βάρος 0,75) δημιουργούμε τον **κανόνα**: lightRank(LineID, LitRank) :-client(_, _, _, _, Time, _, _, _),isNight(Time), lineInfo(LineID, _, _, _,yes, _, _, _, _, _, _, _, _, _, _, _, _, _, _, _), LitRank = 0.7.

Αν δεν έχει φώτα, το βάρος είναι 1.

Τέλος, **το συνολικό βάρος**, δηλαδή η τιμή Value υπολογίζεται από τη **σχέση**:
Value = HRank * TRank * LRank * LitRank

και περιγράφεται με τον **κανόνα**:

```
weightFactor(Ax, Ay, Bx, By, Value) :-
   node(Ax, Ay, LineID, _, _),
   node(Bx, By, LineID, _, _),
   roadTypeRank(LineID, HRank),
   trafficRank(LineID, TRank),
   laneRank(LineID, LRank),
   lightRank(LineID, LitRank)
   Value is HRank * TRank * LRank * LitRank.
```

Τέλος, για όλα τα κριτήρια έχει οριστεί ένα default βάρος για την περίπτωση που η πληροφορία που εξετάζουμε δεν υπάρχει για το συγκεκριμένο γεγονός.

4. Υλοποίηση του αλγορίθμου:

Σε αυτή την υλοποίηση του αλγορίθμου, σε αντίθεση με την προηγούμενη άσκηση, καθορίζεται από το input το μέγεθος του μεγίστου μετώπου αναζήτησης.

Ο αλγόριθμος καλείται από τη Main.java για κάθε ταξί, με ορίσματα για κόμβο έναρξης τη θέση του ταξί (την κοντινότερη αυτής στον γράφο αναζήτησης), το μέγιστο μέτωπο αναζήτησης και τον χώρο αναζήτησης βάσει της θέσης του πελάτη. Επιστρέφει τη βέλτιστη διαδρομή που θα πρέπει να ακολουθήσει το ταξί για να φτάσει στον πελάτη.

Πιο συγκεκριμένα, για την εύρεση της βέλτιστης διαδρομής για κάθε ταξί προς τον πελάτη, συγκρίνονται οι κόμβοι μεταξύ τους ως προς το f, δηλαδή το άθροισμα της πραγματικής μέχρι τώρα υπολογιζόμενη απόστασής τους από τον κόμβο έναρξης (θέση του ταξί) g με την ευριστική απόσταση h μέχρι τη θέση του πελάτη. Ο υπολογισμός της ευριστικής σε αυτή την υλοποίηση προκύπτει από την απόσταση haversine (haversine.java) του προς εξέταση κόμβου από τη θέση του πελάτη πολλαπλασιαζόμενη με έναν παράγοντα factor που θέτουμε στην Prolog και δίνει προτεραιότητα σε κάποια ταξί ανάλογα με το αν ικανοποιούν τις απαιτήσεις του πελάτη.

Στη συνέχεια, από το σύνολο των ταξί επιλέγονται μόνο αυτά που ικανοποιούν τις προϋποθέσεις που αναφερθήκαμε παραπάνω.

Για κάθε ένα από αυτά τα ταξί ,με τη χρήση των κανόνων της prolog και του ειδικά υπολογισμένου βάρους σε κάθε ακμή βρίσκουμε την συντομότερη διαδρομή απο το ταξί προς τον πελάτη.

Τέλος, ταξινομούμε τα αποτελέσματα που βρήκαμε:

- ως προς τη συντομότερη απόσταση (k καλύτερες διαδρομές)

- ως προς την αξιολόγηση των χρηστών.

5. Απεικόνιση αποτελεσμάτων:

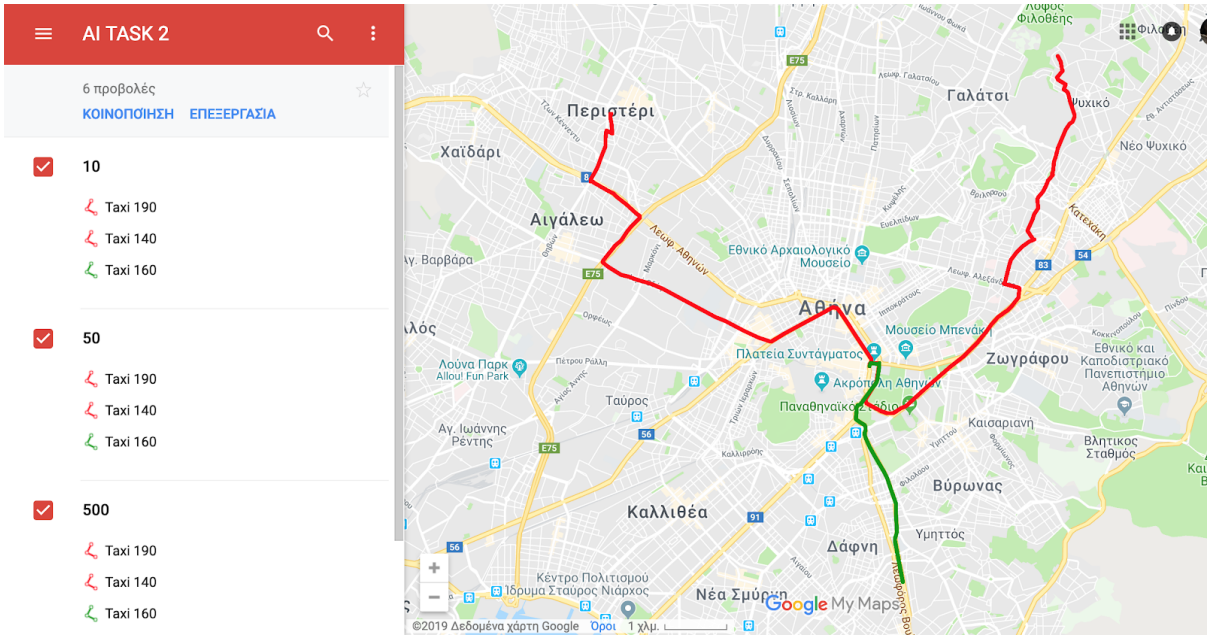
Το πρόγραμμα μας τυπώνει στην οθόνη το παρακάτω αποτέλεσμα:

```
Available taxis sorted by distance:
160 - 3.8634134724647358km
190 - 8.170491885986051km
140 - 8.474103404823683km
Available taxis sorted by rating:
190 - Rating: 7.4
140 - Rating: 7.1
160 - Rating: 6.1
```

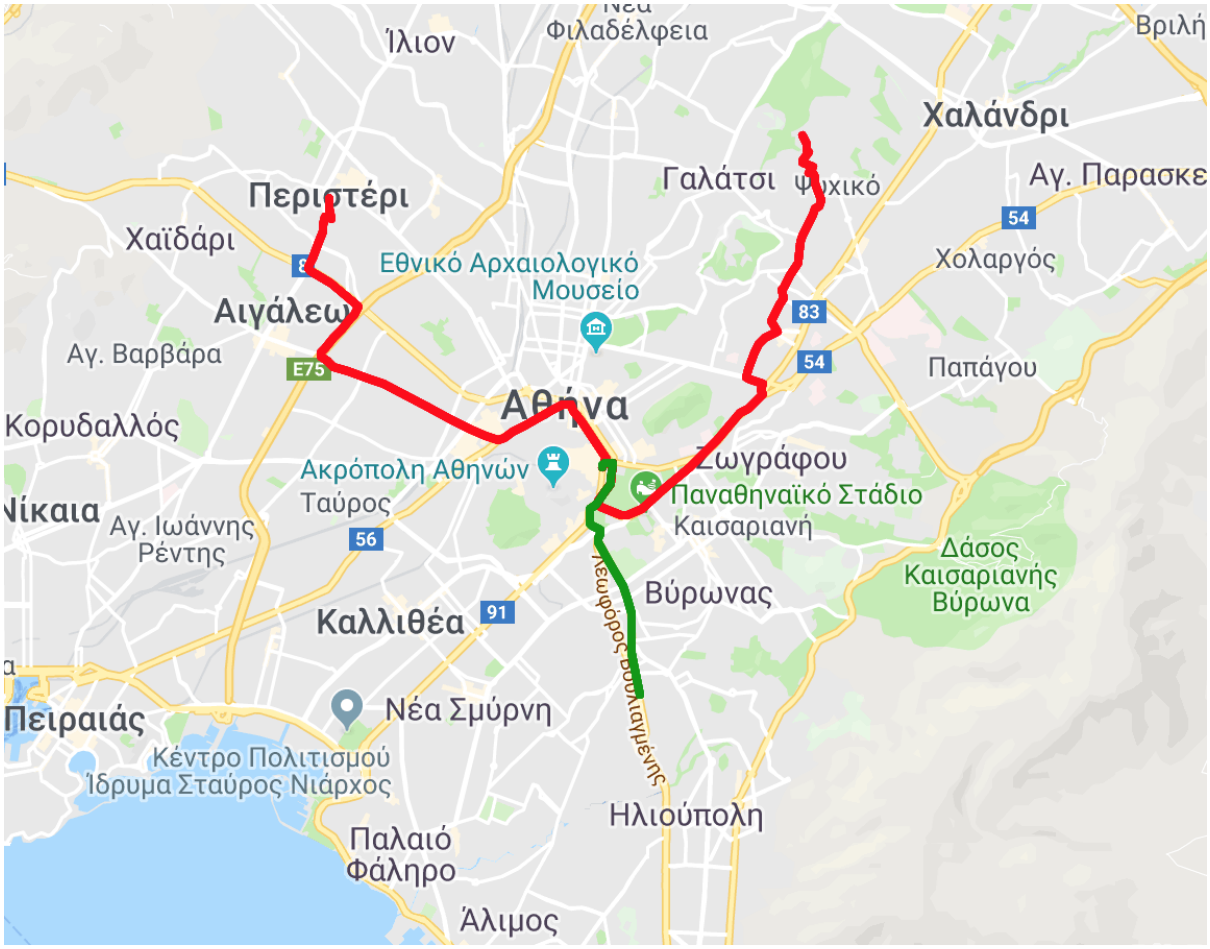
Το κοντινότερο ταξί που επιλέγεται για να εξυπηρετήσει τον πελάτη και εκτελεί τις διαδρομές που παρουσιάζονται στον πίνακα, είναι το ταξί με **id 160** και συντεταγμένες: **23.740078,37.945631.**

Οπτική απεικόνιση των διαδρομών από το MyMaps:

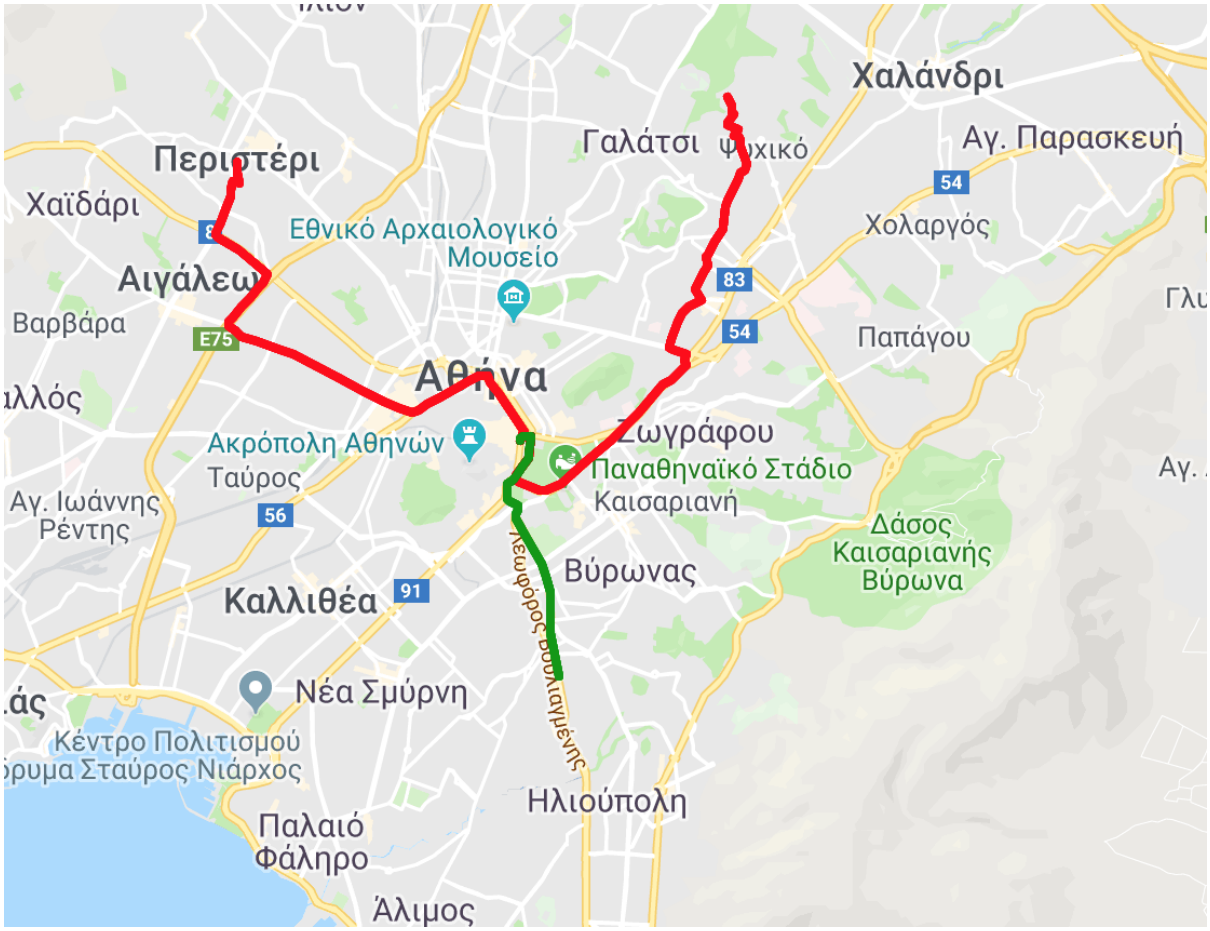
https://drive.google.com/open?id=1ynP6kWtp_P73YiEBZZX1azxHc7jF2Y01&usp=sharing



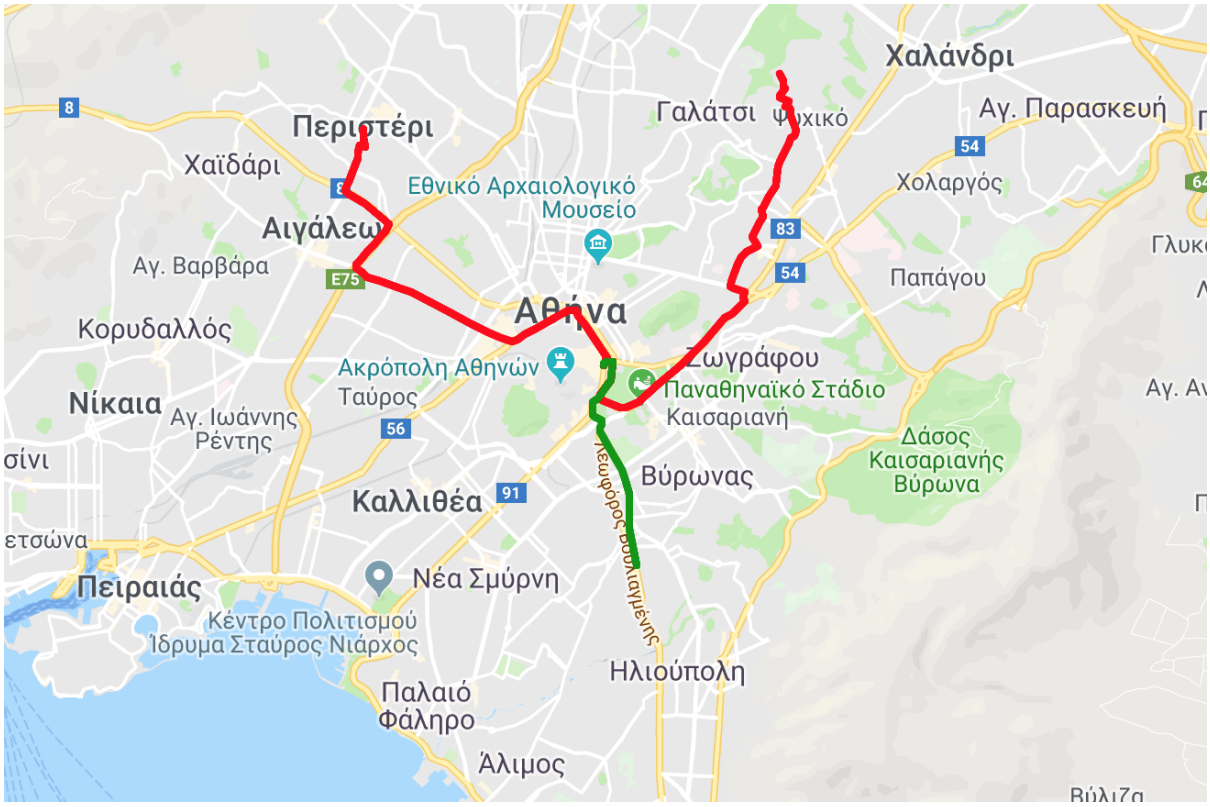
Στη συνέχεια απεικονίζονται οι διαδρομές για μέγιστα μέτωπα αναζήτησης 10, 500, 5000 και 10000 αντίστοιχα.



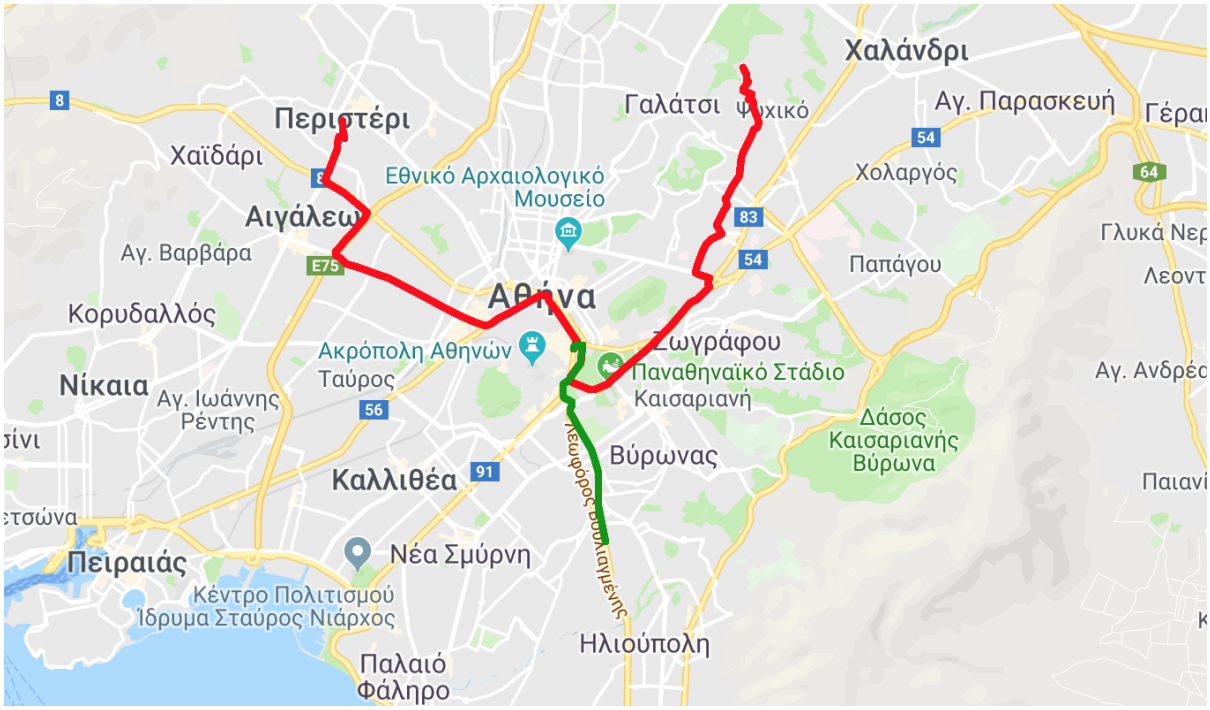
Απεικόνιση για μέγιστο μέτωπο αναζήτησης 10



Απεικόνιση για μέγιστο μέτωπο αναζήτησης 500



Απεικόνιση για μέγιστο μέτωπο αναζήτησης 5000



Απεικόνιση για μέγιστο μέτωπο αναζήτησης 10000