1ο Θέμα στην Τεχνητή Νοημοσύνη

Μέλη Ομάδας :

1) ***Καναβάκης Ελευθέριος 03114180***
2) ***Αλεξάκης Κωνσταντίνος 03114086***

Συνοπτική παρουσίαση της υλοποίησης μας :

Για την υλοποίηση της συγκεκριμένης εφαρμογής χρησιμοποιήσαμε την αντικειμενοστραφή γλώσσα προγραμματισμού Java . Στην συνέχεια θα προχωρήσουμε σε μια συνοπτική παρουσίαση-επεξήγηση της εφαρμογής μας . Πριν όμως πραγματοποιήσουμε την περιγραφή αυτή θεωρούμε σκόπιμο να αναφερθούμε στα δεδομένα που μας δίνονται σαν είσοδο και τον τρόπο με τον οποίο τα διαχειριστήκαμε . Αρχικά , στο αρχείο nodes.csv μας δίνεται ο χάρτης της εφαρμογής . Φυσικά , ο χάρτης αυτός δεν δίνεται σε μορφή βολική για επεξεργασία καθώς αποτελείται από γραμμές excel . Ως εκ τούτου , δημιουργήσαμε με χρήση HashMap (περισσότερες λεπτομέρειες δίνονται παρακάτω) τον χάρτη μας . Παράλληλα , μας δίνονται τα αρχεία client.csv και taxis.csv τα οποία αφορούν τις συντεταγμένες του πελάτη και των ταξί αντίστοιχα . Αφού διαβάσαμε τα αρχεία αυτά , κάναμε match τους ήδη υπάρχοντες κόμβους του χάρτη με τα ταξί και τον πελάτη . Για την πραγματοποιήσει του matching αυτού αναζητήσαμε τους πλησιέστερους κόμβους στις συντεταγμένες εισόδου του κάθε ταξί ή πελάτη . Στην συνέχεια ,θα προχωρήσουμε στην περιγραφή της εφαρμογής μας . Για την περιγραφή αυτή επιλέξαμε να παραθέσουμε την λίστα των αρχείων .java μαζί με μια σύντομη περιγραφή και το τι υλοποιείται στο κάθε αρχείο .

Η λίστα των αρχείων της εφαρμογής μας καθώς και μία σύντομη περίληψη της εργασίας που υλοποιεί το κάθε αρχείο παρατίθενται παρακάτω :

- ***Pair.java*** , στην κλάση αυτή ορίζουμε μία custom τούπλα 2 στοιχείων . Η τούπλα αυτή δεν έχει συγκεκριμένο τύπο στοιχείων . Ως εκ τούτου , θα μπορούσε να πει κανείς ότι πρόκειται για ένα πολυεργαλείο καθώς στην τούπλα αυτή κρατάμε τόσο συντεταγμένες x,y όσο και άλλους πολυπλοκότερους συνδυασμούς όπως στοιχείο και τούπλα (πχ taxi id , tuple(taxi x,taxi y)) και άλλα.
- ***Quatret.java*** , στην κλάση αυτή ορίζουμε μία custom τούπλα 4 στοιχείων . Η τούπλα αυτή διαφέρει από την προηγούμενη καθώς είναι λιγότερο γενική . Πιο συγκεκριμένα , στην τούπλα αυτή κρατάμε την συντεταγμένη x , την συντεταγμένη y , την ευριστική τιμή απόστασης από τον πελάτη και τέλος έναν πίνακα (arraylist) στο οποίο κρατάμε τούπλες (2 στοιχειών →Pair.java) για το μονοπάτι μέχρι τον κόμβο που ορίζεται από τα πρώτα δύο στοιχεία της τούπλας αυτής . Από την παραπάνω περιγραφή εύκολα μπορεί να καταλάβει κανείς ότι η συγκεκριμένη δομή δεδομένων θα χρησιμοποιηθεί κατά κόρον κατά την υλοποίηση του αλγόριθμου Astar !
- ***Quatretcompare.java*** , στην κλάση αυτή κάνουμε implement τον comparator και κάνουμε override την συνάρτηση compare για την

κλάση Quatret. Φυσικά , ο λόγος είναι προφανής καθώς δεν μπορεί να συγκριθεί με τους τελεστές μικρότερου ή μεγαλύτερου ένα αντικείμενο της κλάσης Quatret . Ως , εκ τούτου είναι απαραίτητο να κάνουμε override την συνάρτηση compare . Στα πλαίσια της εφαρμογής μας , επιθυμούμε δύο αντικείμενα της κλάσης Quatret να συγκρίνονται με βάση το τρίτο τους όρισμα , δηλαδή με βάση την απόσταση τους από τον τελικό στόχο (πελάτης).

- ***Client.java*** , η κλάση αυτή έχει διπλό ρόλο καθώς όχι μόνο είναι υπεύθυνη για το store ενός αντικειμένου Client (τούπλα από συντεταγμένες X,Y) αλλά και καλείται για το διάβασμα του αρχείου client.csv.

- ***Taxis.java ,*** η κλάση αυτή όπως και η προηγούμενη έχει διπλό ρόλο καθώς όχι μόνο είναι για το store ενός αντικειμένου Taxi (τούπλα από συντεταγμένες και ακέραιος αριθμός για το identifier του κάθε taxi) αλλά και καλείται για το διάβασμα του αρχείου taxis.csv.

- ***NeighborNode.java ,*** η κλάση αυτή όπως μπορεί να καταλάβει κανείς και από το όνομα της είναι μια custom δομή δεδομένων η οποία θα μας βοηθήσει να κρατήσουμε γειτονικούς κόμβους . Πιο συγκεκριμένα , ένα αντικείμενο της κλάσης αυτής μπορεί να έχει τρία πεδία . Μια τούπλα 2 στοιχείων Double για συντεταγμένες , έναν double αριθμό για την ευριστική τιμή της απόστασης καθώς και έναν ακέραιο αριθμό για να κρατάμε το αναγνωριστικό (id) του κάθε δρόμου . Παράλληλα , στην κλάση ορίζεται και η συνάρτηση στην οποία υπολογίζεται η ευριστική τιμή που θα χρησιμοποιήσουμε για την εκτέλεση του αλγορίθμου Astar . Πιο συγκεκριμένα , στην συνάρτηση αυτή δεχόμαστε σαν ορίσματα δύο ζεύγη συντεταγμένων x,y . Στην συνέχεια , κάνουμε μετατροπή των συντεταγμένων αυτών με χρήση γωνιών (καθώς η γη είναι σφαιρική και όχι επίπεδη) για να φέρουμε τις συντεταγμένες αυτές σε κατάλληλη μορφή ώστε να υπολογίσουμε την ευκλείδεια απόσταση τους . Τέλος , υπολογίζουμε την απόσταση αυτή και την επιστρέφουμε .

- ***Mapping.java ,*** η κλάση αυτή είναι υπεύθυνη για το διάβασμα του αρχείου nodes.csv και την δημιουργία του χάρτη με βάση τους κόμβους που βρίσκονται στο αρχείο αυτό . Πιο συγκεκριμένα για τον χάρτη μας θα χρησιμοποιήσουμε την δομή δεδομένων HashMap . Στο

HashMap αυτό κλειδί θα είναι κάθε φορά ο υπό εξέταση κόμβος και σαν values θα έχουμε μία τούπλα 2 στοιχείων η οποία θα αποτελείται από την ευριστική τιμής της απόστασης από τον κόμβο στόχο (πελάτης) και από έναν πίνακα (ArrayList) ο οποίος θα περιέχει όλους τους γειτονικούς κόμβους που είναι προσβάσιμη απ' ευθείας από τον κόμβο με συντεταγμένες που αντιστοιχούν στο εκάστοτε key . Ο πίνακας γειτονικών κόμβων που περιγράφηκε προηγουμένως ουσιαστικά θα είναι ένας πίνακας από αντικείμενα της κλάσης NeighborNode . Τέλος , αξίζει να σημειώσουμε πως για την αρχικοποίηση της ευριστικής τιμής του κάθε κόμβου κάνουμε generate μια τυχαία double τιμή καθώς δεν γνωρίζουμε ακόμα την ακριβή θέση του κόμβου στόχου (η τιμή αυτή θα ανανεωθεί αργότερα).

- ***Input.java ,*** η κλάση αυτή είναι υπεύθυνη για το διάβασμα των αρχείων client.csv (κλήση της μεθόδου data της κλάσης Client.java) και taxi.csv (κλήση της μεθόδου data της κλάσης Taxis.java).Αφού διαβάσει το κάθε αρχείο κρατάει τον πελάτη σαν έναν αντικείμενο της κλάσης Client(περιγράφεται παραπάνω) και τα ταξί σαν ένα πίνακα (ArrayList) από αντικείμενα της κλάσης Taxis (περιγράφηκε επίσης παραπάνω).

- ***Updatemap.java ,*** η κλάση αυτή είναι υπεύθυνη για την εύρεση του ακριβούς κόμβου στον οποίο ανήκει ο πελάτης και το κάθε ταξί στον χάρτη καθώς και της επικαιροποίησης της ευριστικής τιμής του κάθε κόμβου του χάρτη μετά την εύρεση του ακριβούς κόμβου στον χάρτη στον οποίο βρίσκεται ο πελάτης. Για την πραγματοποίηση των παραπάνω διατρέχεται ολόκληρο το HashMap που ορίσαμε στο αρχείο Mapping.java και βρίσκουμε την minimum απόσταση του πελάτη καθώς και του κάθε ταξί από τους υπάρχοντες κόμβου του χάρτη. Στην συνέχεια ενημερώνουμε τις συντεταγμένες τόσο του πελάτη όσο και των ταξί με τις συντεταγμένες του κόμβου του χάρτη μας που βρίσκονται όσο το δυνατόν πιο κοντά είτε στο εκάστοτε ταξί είτε στον πελάτη . Ουσιαστικά , βρίσκουμε τον κόμβο του χάρτη στον οποίο βρίσκεται ποιο κοντά ο πελάτης (το ίδιο κάνουμε και για κάθε ταξί) καθώς είναι αρκετά πιθανό πως οι συντεταγμένες που μας δίνονται σαν είσοδο δεν αποτελούν κόμβους του χάρτη μας (για παράδειγμα θα μπορούσε ο πελάτης να βρίσκεται ανάμεσα από δύο κόμβους ενός δρόμου, στην περίπτωση αυτή θα επιλεγεί ο πλησιέστερος από τους δύο σαν η θέση του πελάτη).Τέλος, διατρέχουμε και πάλι ολόκληρο

τον χάρτη και ανανεώνουμε την ευριστική τιμή την οποία προηγουμένως είχαμε αρχικοποιήσει τυχαία . Η ανανέωση της ευριστικής αυτής γίνεται και πάλι με χρήση της συνάρτησης που περιγράψαμε παραπάνω . Η τιμή της ευριστικής προσδιορίζεται κάθε φορά από τις συνταγμένες του κόμβου στον οποίο βρίσκεται ο πελάτης καθώς και της συντεταγμένες του προς εξέταση κόμβου (δηλαδή του κλειδιού του κάθε κλειδιού του hash map). Τέλος, αφού ολοκληρωθεί η συνάρτηση αυτή έχουμε ολοκληρώσει με επιτυχία τόσο το διάβασμα των δεδομένων εισόδου όσο και την επεξεργασία αυτών ώστε να βρίσκονται σε κατάλληλη μορφή.

- ***Astar.java ,*** στην κλάση αυτή υλοποιείται ο αλγόριθμος αναζήτησης A* ο οποίος επιστρέφει ένα set από ελάχιστα μονοπάτια από ένα ταξί προς τον κόμβο στόχο (πελάτης). Για την υλοποιήσει του αλγορίθμου αυτού θα χρησιμοποιήσουμε μία λίστα ελάχιστης προτεραιότητας (minimum priority queue) η οποία αποτελεί την πλέον κατάλληλη δομή δεδομένων για την υλοποίηση του μετώπου αναζήτησης καθώς στην κορυφή της λίστας αυτής θα κρατάμε πάντα τον κόμβο με την ελάχιστη απόσταση από τον στόχο . Παράλληλα , θα χρησιμοποιήσουμε ένα HashSet για να υλοποιήσουμε το κλειστό σύνολο , δηλαδή θα κρατάμε τους κόμβου που έχουμε ήδη εξετάσει έτσι ώστε να μην εξετάσουμε ξανά έναν κόμβο που έχει ήδη εξεταστεί . Επιπρόσθετα , θα χρησιμοποιήσουμε και ένα HashMap στο οποίο για κάθε κόμβο που εισάγουμε στο μέτωπο αναζήτησης θα κρατάμε την καλύτερη απόσταση του από τον στόχο . Με την χρήση του HashMap επιτυγχάνουμε να εισάγουμε στο μέτωπο αναζήτησης κόμβους που είτε δεν υπάρχουν σε αυτό και δεν ανήκουν ούτε στο κλειστό σύνολο είτε κόμβους που υπάρχουν στο μέτωπο αναζήτησης αλλά έχουν μεγαλύτερη απόσταση από ότι έχει ο υπό εξέταση κόμβος .Φυσικά , αποδεχόμαστε και κόμβους που έχουν distance ίσο με το έως τώρα βέλτιστο καθώς αναζητάμε όλα τα ελάχιστα μονοπάτια . Αφού περιγράψαμε τις δομές δεδομένων η υλοποίηση του αλγορίθμου είναι σχετικά απλή . Προσθέτουμε στην ουρά προτεραιότητας κόμβους που δεν ανήκουν στο κλειστό σύνολο και έχουν απόσταση από τον στόχο μικρότερη ή ίση από την έως τώρα καταγεγραμμένη ως βέλτιστη . Αυτό , το υλοποιούμε μέχρι να αδειάσει η ουρά προτεραιότητας . Κάθε φορά που βρίσκουμε τον κόμβο στόχο κρατάμε το ελάχιστο μονοπάτι έως αυτόν σε ένα πίνακα (ArrayList) από πίνακες (ArrayLists) από ελάχιστα μονοπάτια . Τέλος , θα θέλαμε να τονίσουμε ότι στην παραπάνω περιγραφή δεν εξηγήσαμε πως βρίσκουμε παραπάνω από ένα ελάχιστα μονοπάτια αλλά μόνο πως βρίσκουμε το πρώτο

minimum path . Λεπτομέρειες για την υλοποίηση αυτή θα δωθούν αναλυτικά παρακάτω .

- ***KML.java ,*** η συγκεκριμένη κλάση είναι όπως καταλαβαίνουμε και από το όνομά της υπεύθυνη για την δημιουργία του τελικού .kml αρχείου το οποίο θα περιλαμβάνει όλες τις ελάχιστες διαδρομές για κάθε ταξί . Όπως , ζητείται και από την εκφώνηση της άσκησης το ταξί με την ελάχιστη διαδρομή αναπαρίσταται με πράσινο χρώμα ενώ τα υπόλοιπα αναπαρίστανται με όσο το δυνατόν διαφορετικά χρώματα . Δυστυχώς , στην πράξη αν και για το κάθε χρώμα κάναμε Random generate RGB colors αρκετά χρώματα βγήκαν παρόμοια . Η μέθοδος make της συνάρτησης αυτής δέχεται σαν όρισμα μια αρκετά πολύπλοκη δομή δεδομένων (πίνακας από τούπλες όπου η κάθε τούπλα αποτελείται από την απόσταση της ελάχιστης διαδρομής προς τον πελάτη καθώς και έναν πίνακα από πίνακες με τα ελάχιστα μονοπάτια) και με βάση αυτή τυπώνει όλα τα ελάχιστα μονοπάτια . Αν μάλιστα για ένα ταξί υπάρχουν περισσότερα από ένα ελάχιστα μονοπάτια τότε τα μονοπάτια αυτά τυπώνονται σε μορφή γράφου από το σημείο το οποίο έχουμε διακλάδωση και κάτω.
- ***TaxiBeat.java*** , πρόκειται ουσιαστικά για την main συνάρτηση της εφαρμογής μας η οποία θα μπορούσε να πει κανείς ότι κινεί τα νήματα καθώς καλεί τα υπόλοιπα αρχεία (.java) με σκοπό να υλοποιήσουν αυτό για το οποίο σχεδιάστηκαν. Με αφορμή την περιγραφή της συνάρτησης αυτής θα κάνουμε μία **σύνοψη** των όσων περιγράψαμε παραπάνω . Αρχικά , με χρήση της κλάσης Mapping.java δημιουργούμε ένα HashMap με τον χάρτη μας βασισμένο στο αρχείο nodes.csv . Στην συνέχεια , με την χρήση των κλάσεων Input.java,UpdateMap.java,Taxis.java και Client.java διαβάζουμε της θέσης των ταξί και του πελάτη πάνω στον χάρτη μας και επικαιροποιούμε την ευριστική του κάθε κόμβου στον χάρτη . Αφού , τελειώσαμε με τα δεδομένα εισόδου εκτελούμε τον Astar για κάθε ταξί . Τέλος , βρίσκουμε το ταξί με την διαδρομή ελαχίστου μήκους και καλώντας την make της KML.java τυπώνουμε την διαδρομή αυτή με πράσινο και όλες τις υπόλοιπες διαδρομές για τα υπόλοιπα ταξί με διαφορετικό χρώμα.

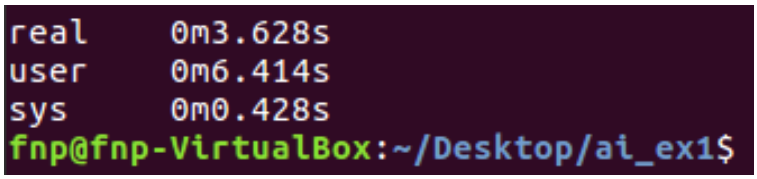
Μεταγλώττιση και εκτέλεση : στο συμπιεσμένο αρχείο το οποίο θα παραδώσουμε δεν συμπεριλαμβάνουμε τα εκτελέσιμα .class αρχεία καθώς για την εκτέλεση τους απαιτείται η ίδια έκδοση java με την οποία μεταγλωττίστηκαν . Φυσικά , για να μεταγλωττίσει κάποιος τα αρχεία .java σε . class αρκεί να τρέξει την παρακάτω εντολή σε ένα τερματικό :

```
fnp@fnp-VirtualBox:~/Desktop/ai_ex1$ javac Astar.java Client.java Input.java KML.java Mapping.java Nei
ghborNode.java Pair.java Quatret.java Quatretcompare.java TaxiBeat.java Taxis.java Updatemap.java
```

Παρακάτω παραθέτουμε και ένα μέρος του αποτελέσματος εξόδου της εφαρμογής μας όταν την εκτελούμε με είσοδο τα default αρχεία που μας δίνονται από την εκφώνηση της άσκησης . Όπως φαίνεται και παρακάτω για την εκτέλεση της εφαρμογής μας αρκεί η εντολή : java TaxiBeat nodes.csv client.csv taxis.csv example (το τελευταίο όρισμα αποτελεί το όνομα του αρχείου kml, στην περίπτωση αυτή example2.kml).

```
fnp@fnp-VirtualBox:~/Desktop/ai_ex1$ time java TaxiBeat nodes.csv client.csv taxis.csv example2

Taxi's id : 100
Client was found !
Number of min paths found : 9
Distance to goal in kilometers : 1.3696755979043926

-----------------------------------------------------

Taxi's id : 110
Client was found !
Number of min paths found : 7
Distance to goal in kilometers : 4.281238880665585

-----------------------------------------------------

Taxi's id : 120
Client was found !
Number of min paths found : 3
Distance to goal in kilometers : 3.0127584592724883

-----------------------------------------------------

Taxi's id : 130
Client was found !
Number of min paths found : 19
Distance to goal in kilometers : 8.886769418710813

-----------------------------------------------------

Taxi's id : 140
Client was found !
Number of min paths found : 13
Distance to goal in kilometers : 6.637343356417281

-----------------------------------------------------

Taxi's id : 150
Client was found !
Number of min paths found : 9
Distance to goal in kilometers : 1.8313939262122114

-----------------------------------------------------

Taxi's id : 160
Client was found !
Number of min paths found : 4
Distance to goal in kilometers : 3.6087763913050344

-----------------------------------------------------
```

Τέλος , παραθέτουμε και screenshot του χρόνου εκτέλεσης της εφαρμογής μας για τα default αρχεία εισόδου που δίνονται στην εκφώνηση της άσκησης .



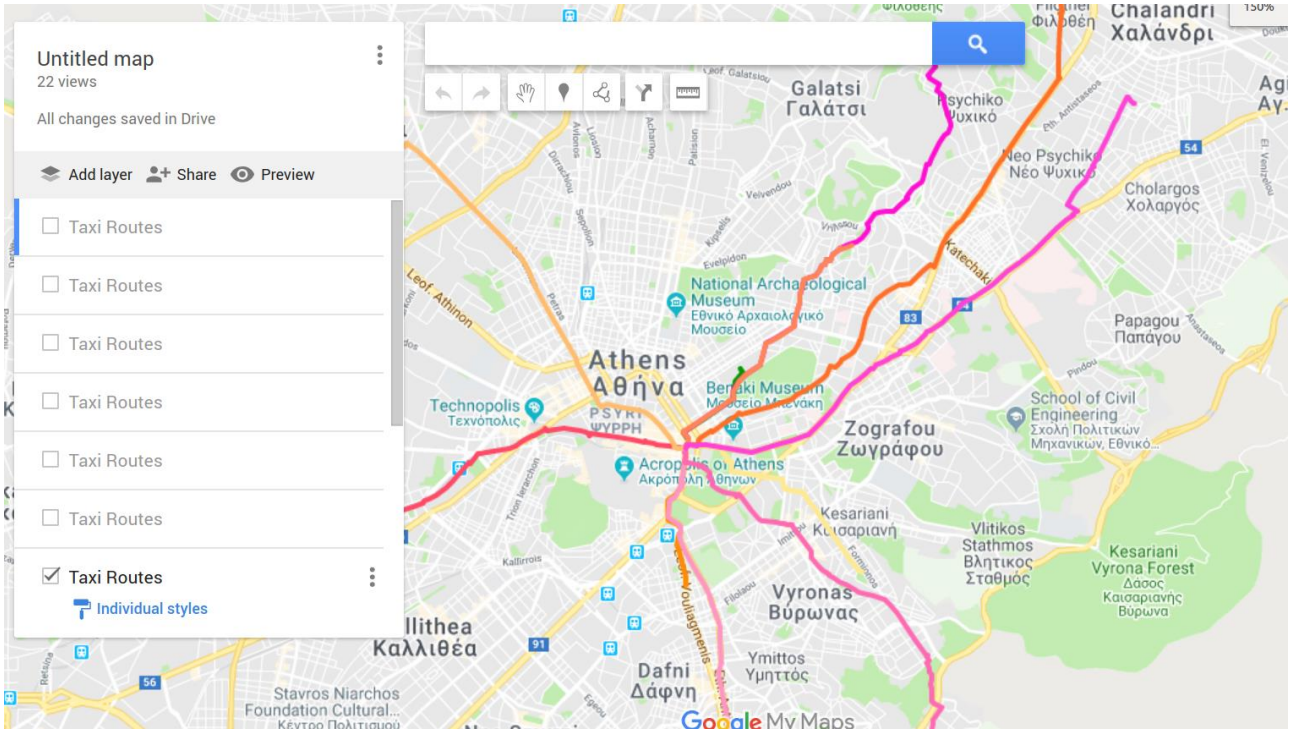
Εύρεση set ελαχίστων μονοπατιών : Όπως αναφέραμε και παραπάνω, στην άσκηση αυτή μας ζητείται να βρούμε εναλλακτικές ελάχιστες διαδρομές . Στην προσπάθεια μας να το πετύχουμε αυτό συναντήσαμε αρκετά εμπόδια τα οποία και θα σχολιάσουμε παρακάτω . Αρχικά , σαν τύπο δεδομένων χρησιμοποιήσαμε doubles για τις αποστάσεις μας . Το γεγονός αυτό προσθέτει σημαντική ακρίβεια στην υλοποίηση μας . Πιο συγκεκριμένα , οι αποστάσεις μας αποτελούνται από παραπάνω από 13 δεκαδικά ψηφία . Ως εκ τούτου, η εφαρμογή μας επιτυγχάνει ακρίβεια nm στους υπολογισμούς . Το γεγονός αυτό καθιστά ξεκάθαρο ότι η εύρεση δύο μονοπατιών με το ίδιο ακριβός είναι πρακτικά ανέφικτη . Το γεγονός αυτό μας οδήγησε στο να εισάγουμε μια μεταβλητή ceiling επί την οποία πολλαπλασιάζουμε την απόσταση για να δεχόμαστε και αποστάσεις με μικρές αποκλίσεις . Πιο συγκεκριμένα , για την μεταβλητή αυτή επιλέξαμε μία τιμή γύρω στο 1% . Δηλαδή , για μία απόσταση 1 χιλιομέτρου δεχόμαστε απόκλιση 10 μέτρων , για μία απόσταση 10 χιλιομέτρων δεχόμαστε απόκλιση 100 μέτρων κτλ. Η τιμή της μεταβλητής αυτής δεν επιλέχθηκε τυχαία . Προσπαθήσαμε να διατηρήσουμε τις αποκλίσεις σε λογικά πλαίσια . Πιο συγκεκριμένα , θεωρήσαμε ότι μια μέση απόσταση από ένα ταξί προς ένα στόχο είναι περίπου τα 5 χιλιόμετρα . Στην απόσταση αυτή προσθέτουμε απόκλιση 50 μέτρα . Παρατηρούμε , ότι η απόκλιση αυτή είναι απολύτως αποδεκτή καθώς στην πράξη 50 μέτρα περισσότερα συνήθως δεν επιφέρουν καμία απολύτως αλλαγή όταν μιλάμε για αυτοκίνητα (αν μιλάμε για αποστάσεις με τα πόδια φυσικά τα πράγματα θα άλλαζαν). Στην συνέχεια , επανεξετάσαμε την ευριστική μας την οποία και εν τέλει δεν αλλάξαμε . Παρατηρήσαμε ότι η ευριστική αυτή (μετατροπή με γωνίες σε συντεταγμένες σφαίρας και στην συνέχεια ευκλείδεια απόσταση) είχε καλύτερη ακρίβεια τόσο από την αντίστοιχη manhattan distance όσο και από τις naïve manhattan και Euclidean distances . Φυσικά , θα μπορούσαμε να πραγματοποιήσουμε την στρογγυλοποίηση που περιγράψαμε παραπάνω με την χρήση μίας λιγότερης ακρίβειας ευριστικής αλλά έτσι θα μειώναμε και την απόδοση του A* . Ως εκ τούτου , διατηρήσαμε την ευριστική μας συνάρτηση . Τέλος , πριν

παραθέσουμε τα αποτελέσματα μας σε χάρτες θα θέλαμε να σχολιάσουμε και κάτι ακόμα . Η ικανότητα του αλγορίθμου μας να προτείνει εναλλακτικές διαδρομές εξαρτάται αρκετά και από το γεγονός ότι ο χώρος στον οποίο δουλεύουμε είναι περίπου επίπεδος (φυσικά η γη δεν είναι επίπεδη αλλά για μικρές αποστάσεις η καμπυλότητα της γης δεν είναι ιδιαίτερα εμφανείς) . Το γεγονός αυτό αποτελεί πρόβλημα καθώς είναι γνωστό από την γραφοθεωρία πως σε μία επίπεδη επιφάνεια η συντομότερη απόσταση μεταξύ δύο σημείων είναι μία ευθεία γραμμή . Ως εκ τούτου, τα ταξί μας προσπαθούν για να επιτύχουν την μικρότερη απόσταση να πραγματοποιήσουν όσο το δυνατόν λιγότερες στροφές . Οι εναλλακτικές διαδρομές που μπορούν να προκύψουν λοιπόν δεν είναι τίποτα παραπάνω από εναλλακτικές στροφές . Το γεγονός αυτό σε συνδυασμό με το ότι θεωρούμε όλους τους δρόμους προσπελάσιμους καθιστά την εφαρμογή μας κάπως χαζή καθώς προτείνει και εναλλακτικές διαδρομές οι οποίες στην πραγματικότητα είναι απλά απέναντι πλευρές σε ένα τετράγωνο .

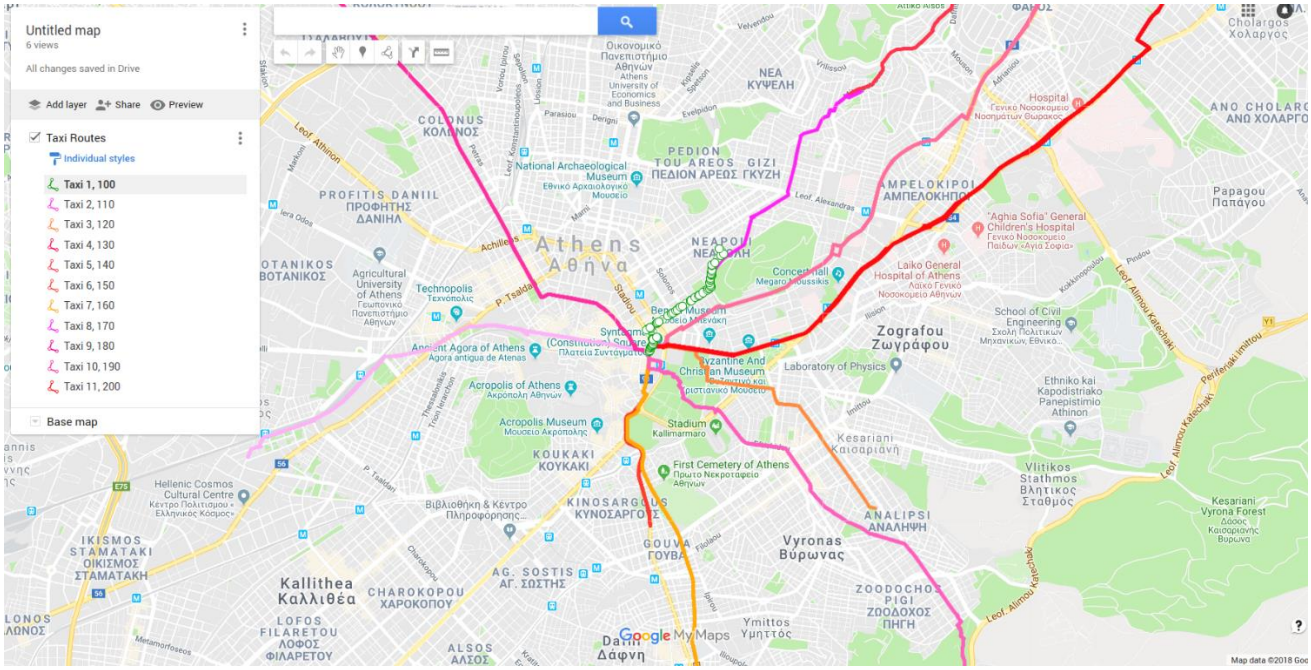
Στην συνέχεια θα παραθέσουμε τα αποτελέσματα μας με την μορφή χάρτη :

Default client και taxis :

Ceiling = 1.0 (απόκλιση 0. per node) αρχείο default1.kml :

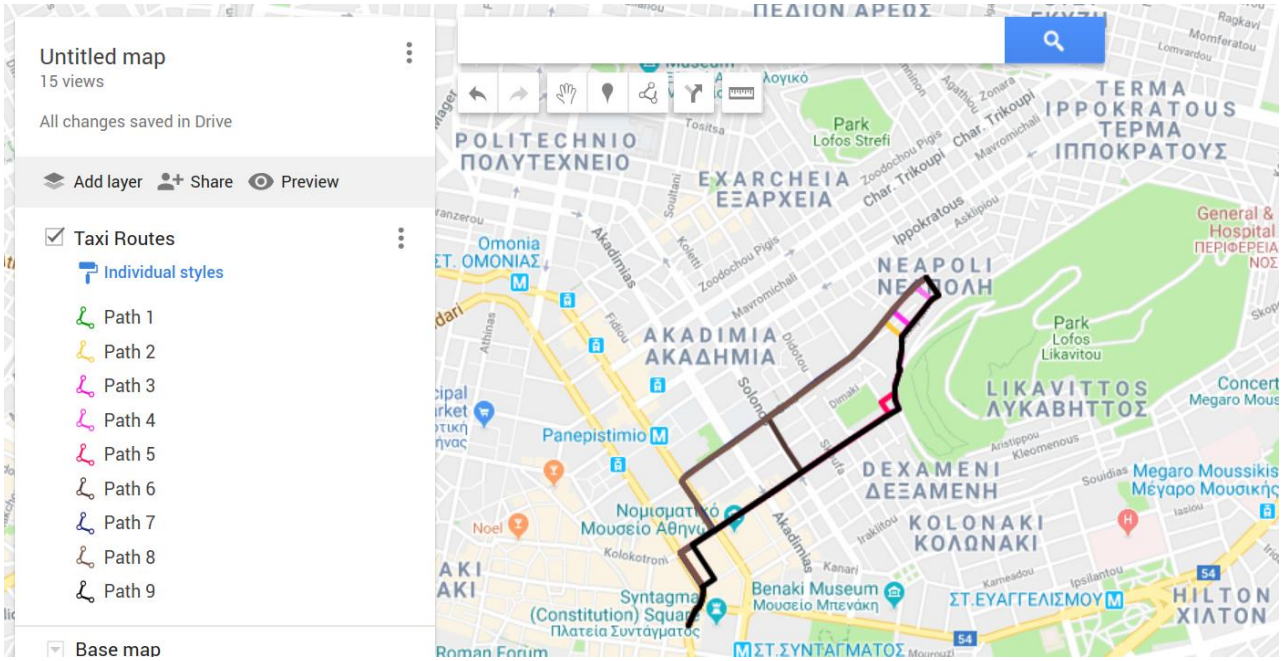


Ceiling = 1.001 (απόκλιση 0.1% per node) αρχείο default3.kml :



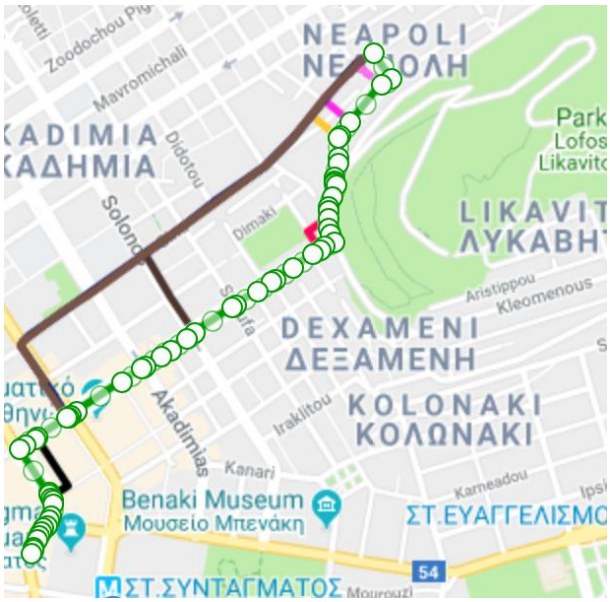
Στο παραπάνω αρχείο έχουμε παραθέσει όλες τις διαδρομές που βρήκαμε για όλα ταξί προς τον πελάτη. Στην συνέχεια , για να δείξουμε καλύτερα την δυνατότητα του αλγορίθμου μας να τυπώνει εναλλακτικές διαδρομές αυξήσαμε την απόκλιση και τυπώσαμε σε ξεχωριστό αρχείο kml όλες τις διαδρομές μόνο για το ταξί με την μικρότερη απόσταση . Δηλαδή το ταξί με identifier 100 . Ο χάρτης του αρχείου αυτού φαίνεται παρακάτω :

Ceiling = 1.02 (απόκλιση 2% per node) αρχείο default2.kml :

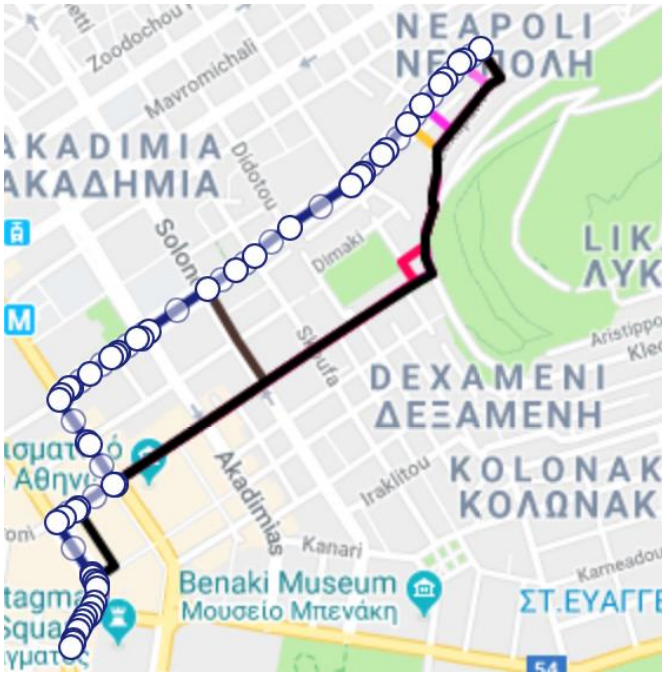


Στην συνέχεια πατήσαμε σε κάποιες από αυτές τις διαδρομές και παραθέτουμε screenshots όταν και αυτές είναι selected .

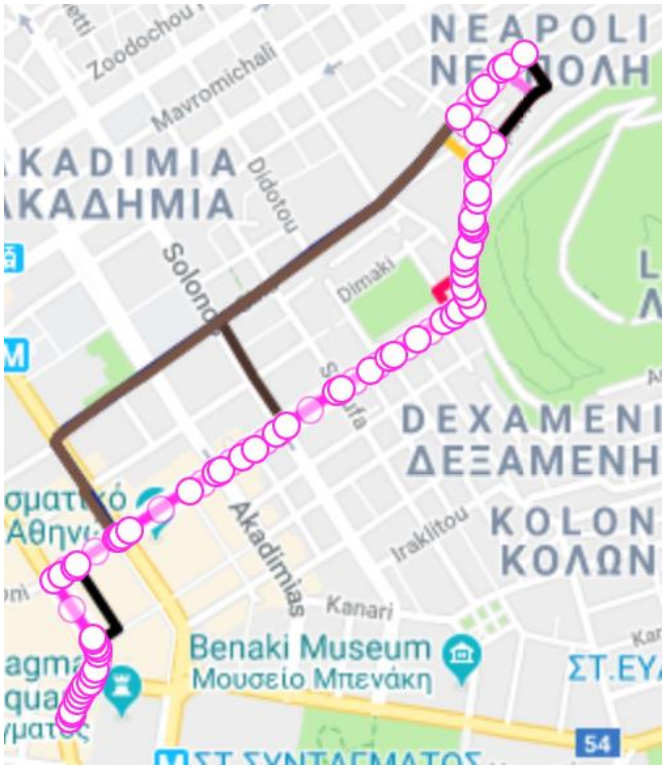
Path 1 :



Path 2 :



Path 3:



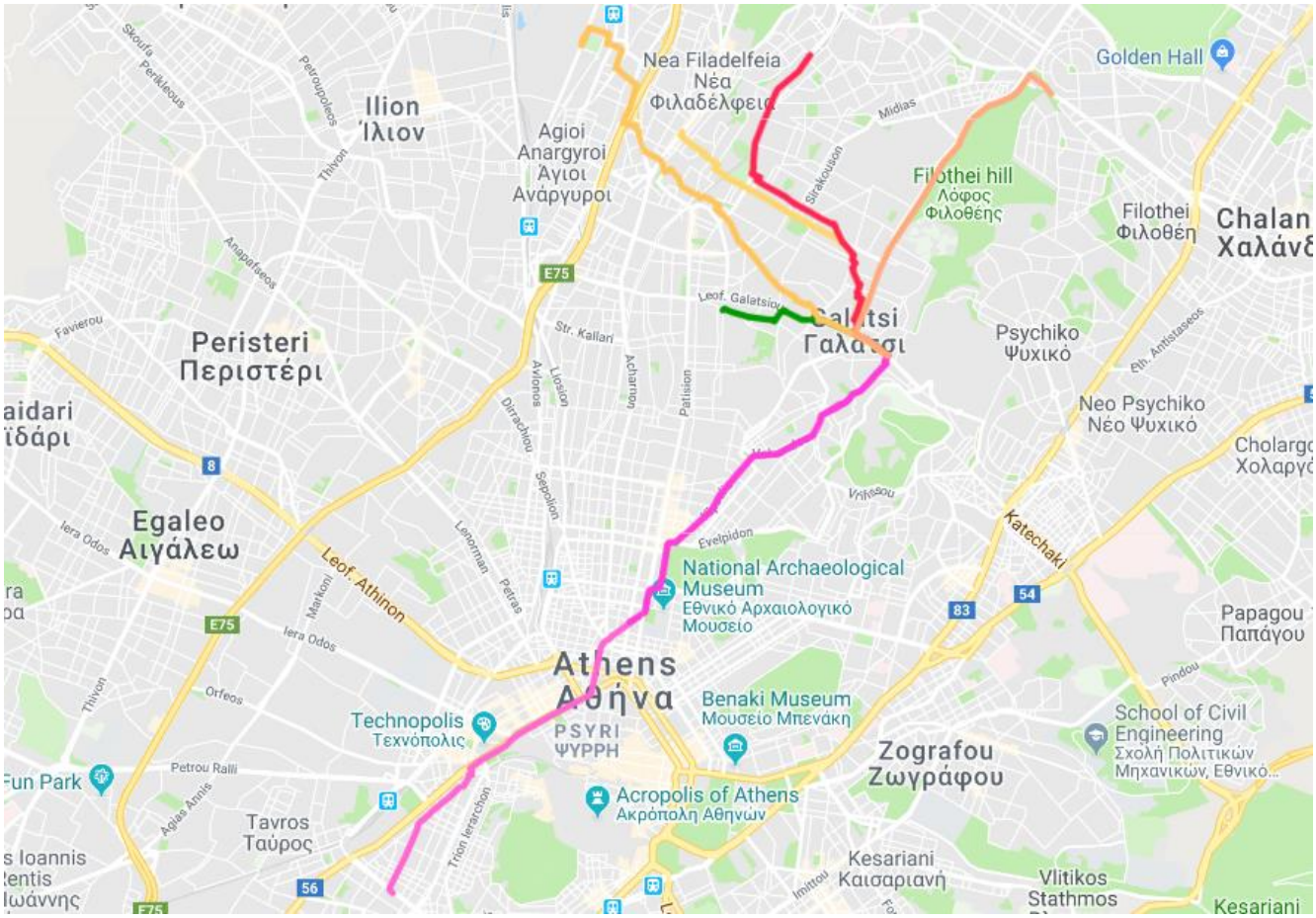
Παρατηρούμε ότι για ceiling = 1 η εφαρμογής μας αδυνατεί να βρει εναλλακτικές διαδρομές . Στην συνέχεια , όσο αυξάνουμε την τιμή του ceiling παρατηρούμε ότι όλο και περισσότερες διαδρομές προκύπτουν από την εφαρμογή μας . Είναι χαρακτηριστικό πως για ceiling=1.02 βρήκαμε 9 εναλλακτικές διαδρομές για το κοντινότερο ταξί . Μάλιστα, η διαδρομές αυτές κυμαίνονται από 1.37 χιλιόμετρα έως 1.39 χιλιόμετρα το οποίο συνάδει απόλυτα με τις υποθέσεις που κάναμε σχετικά με την απόκλιση . Από το γεγονός αυτό θα χαρακτηρίζαμε ότι η εφαρμογή μας είναι σίγουρα ικανή να προτείνει εναλλακτικές διαδρομές οι οποίες μάλιστα είναι «ελάχιστες» . Φυσικά , οι εναλλακτικές διαδρομές αυτές μπορούν να προταθούν με βάση τις παραδοχές που κάναμε σύμφωνα με την εκφώνηση της άσκησης (διπλοί δρόμοι κτλ)!

Τέλος , θα δημιουργήσουμε και τα δικά μας αρχεία .csv με θέσεις πελάτη και ταξί . Τα ονόματα των αρχείων είναι myclient.csv και mytaxis.csv αντίστοιχα .

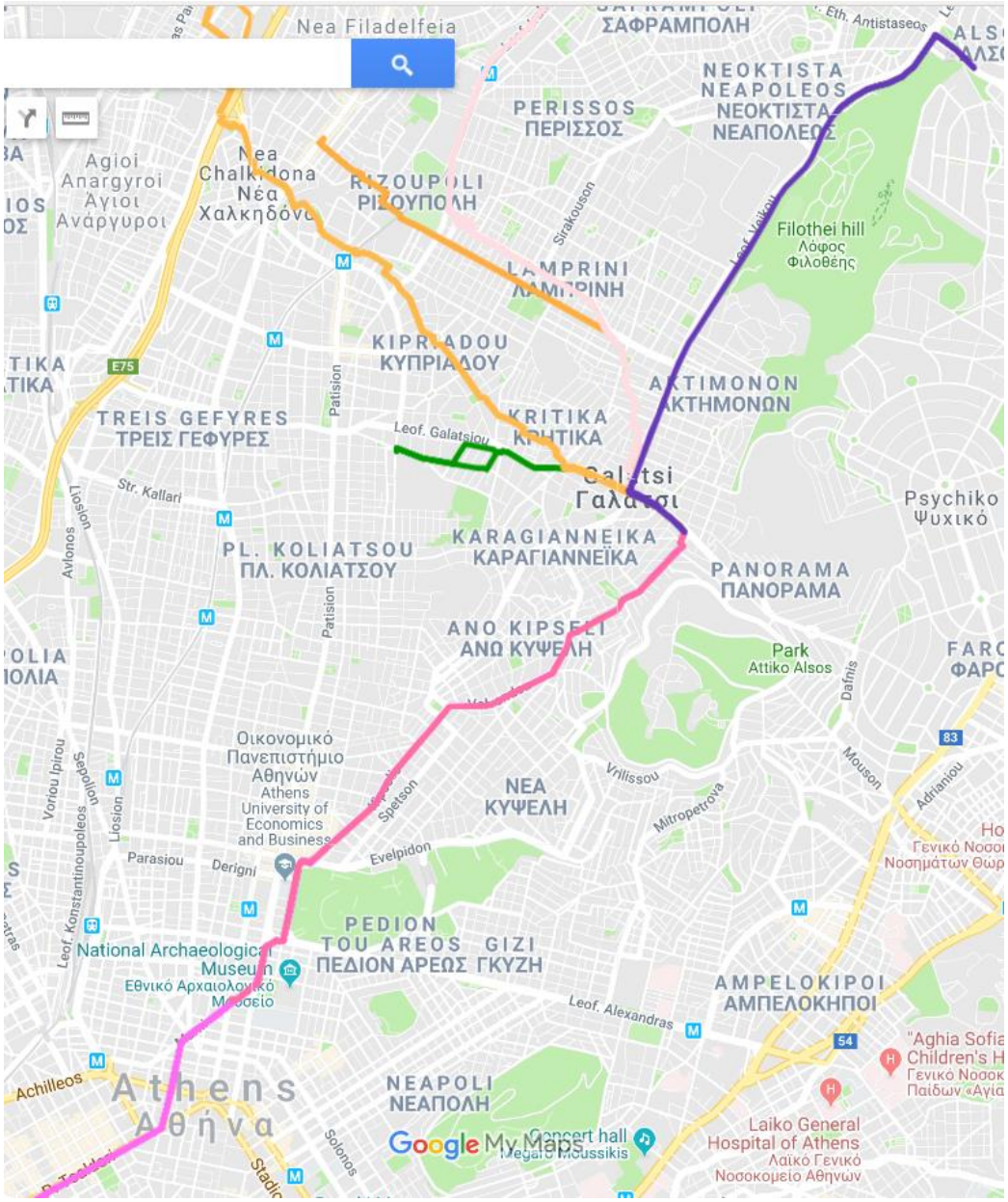
Η έξοδος της εφαρμογής μας με είσοδο τα παραπάνω αρχείο φαίνεται παρακάτω :

Custom client και taxis :

Ceiling = 1.0 (απόκλιση 0. per node) αρχείο custom1.kml :



Ceiling = 1.001 (απόκλιση 0.1% per node) αρχείο custom2.kml :



Παρατηρούμε ότι και πάλι με αν θέσουμε μία μικρή απόκλιση στο μήκος των διαδρομών η εφαρμογή μας μπορεί να προτείνει με επιτυχία εναλλακτικές διαδρομές ! Βέβαια , θεωρούμε σημαντικό να τονίσουμε ότι όσο πιο μεγάλη είναι η απόκλιση που θέτουμε ναι μεν αυξάνονται οι εναλλακτικές διαδρομές αλλά αυξάνονται και οι εναλλακτικές διαδρομές που στην πραγματικότητα δεν είναι εναλλακτικές (πχ στρίβω σε διαφορετικές πλευρές του τετραγώνου).Ως εκ τούτου είναι σημαντικό ,η απόκλιση που δεχόμαστε να είναι αρκετά μικρή ώστε να αποφεύγουμε τέτοιες συμπεριφορές .