ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Τμήμα Διδακτικής της Τεχνολογίας και Ψηφιακών Συστημάτων

Αλγόριθμος επεξεργασίας χωροχρονικών δεδομένων

Κελλάρης Κ. Γεώργιος

Σεπτέμβριος 2008

Περίληψη

Τα δεδομένα τροχιών κινούμενων οχημάτων μπορούν να χρησιμοποιηθούν σε μια σωρεία εφαρμογών. Πηγή των δεδομένων αυτών είναι συνήθως τα GPS συστήματα που είναι εγκατεστημένα στα οχήματα. Το μεγάλο μέγεθος των δεδομένων και τα σφάλματα των GPS συστημάτων καθιστούν την ανάγκη αφενός για συμπίεση και αφετέρου για ταίριασμα της τροχιάς με τον οδικό χάρτη. Έχουν προταθεί αρκετές λύσεις για κάθε ένα από τα δύο προβλήματα χωριστά, αλλά καμία δεν λύνει τη σύνθεση των δύο προβλημάτων, δηλαδή την παραγωγή συμπιεσμένης τροχιάς η οποία βρίσκεται πάνω στο δίκτυο. Σκοπός της έρευνας είναι η πρόταση και η αξιολόγηση τέτοιων μεθόδων.

Ευχαριστίες

Θερμές ευχαριστίες εκφράζω στον Επίκουρο Καθηγητή κ. Ιωάννη Θεοδωρίδη για την επίβλεψη και τη βοήθεια που μου παρείχε για την ολοκλήρωση της διπλωματικής μου εργασίας καθώς και στο μεταδιδακτορικό ερευνητή Νικόλαο Πελέκη.

Τέλος εκφράζω την ευγνωμοσύνη μου στους γονείς μου και την αδελφή μου για την υποστήριξη και βοήθειά τους σε όλη τη διάρκεια των μεταπτυχιακών σπουδών μου.

Περιεχόμενα

Κατάλογος Εικόνων

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

1.1 Εισαγωγή

Κάθε κινούμενο αντικείμενο μπορεί να αναπαρασταθεί από ένα μοναδικό *ID* και η κίνησή του από ένα σύνολο τριάδων <*x*, *y*, *t*> που δείχνουν τη θέση του (*x*, *y*) και τη χρονική στιγμή *t* που βρέθηκε σ' αυτή. Η λήψη των δεδομένων σε ένα τυπικό δέκτη GPS γίνεται ανά 30 δευτερόλεπτα. Η συμπίεση των δεδομένων αυτών είναι αναγκαία σε περίπτωση που χρειάζεται η μελέτη τροχιών πολλών οχημάτων ταυτόχρονα. Επίσης υπάρχει ένα σφάλμα από 2 έως 8 μέτρα στον εντοπισμό της θέσης [8], με αποτέλεσμα να χρειάζονται τεχνικές ταιριάσματος των δεδομένων με τον οδικό χάρτη για σωστά αποτελέσματα.

Οι αλγόριθμοι συμπίεσης και ταιριάσματος χάρτη μπορούν να χωριστούν σε δύο ομάδες, ανάλογα με το είδος των δεδομένων στα οποία εφαρμόζονται: στους offline, αν όλα τα δεδομένα μας είναι γνωστά από πριν, και στους online, αν τα δεδομένα λαμβάνονται σε πραγματικό χρόνο και επεξεργάζονται εκείνη την στιγμή. Offline αλγόριθμοι συμπίεσης είναι αυτός των Douglas-Peckeur [5] και μια παραλλαγή του ώστε να λαμβάνει υπόψη και το χρόνο [9] με τη χρήση της σύγχρονης ευκλείδειας απόστασης (Synchronous Euclidean Distance - SED). Online αλγόριθμοι συμπίεσης είναι οι OPW-TR [7], Thresholds και STTrace [9]. Για το πρόβλημα του ταιριάσματος χάρτη (map-matching) οι Brakatsoulas et al. [1] προτείνουν μία μέθοδο για offline δεδομένα.

Στην παρούσα μελέτη χρησιμοποιούνται οι αλγόριθμοι των Meratnia & de By [7] και Brakatsoulas et al. [1] τόσο για offline όσο και για online δεδομένα. Λύνουν ξεχωριστά το πρόβλημα της συμπίεσης ή του map-matching, αλλά δεν προσφέρουν μια συμπιεσμένη τροχιά πάνω στο δίκτυο [6]. Μία πρώτη λύση είναι η σειριακή χρήση των αλγορίθμων, όπως περιγράφεται στα επόμενα κεφάλαια. Η

έξοδος του αλγορίθμου των Brakatsoulas et al. [1] προσαρμόστηκε ώστε να μπορεί να χρησιμοποιηθεί ως είσοδος του αλγορίθμου των Meratnia & de By [7] και να λαμβάνει υπόψη και το χρόνο. Αυτό έγινε με την προσαρμογή του αλγορίθμου των Brakatsoulas et al. [1] ώστε να κρατάει πληροφορία και για το χρόνο και χρησιμοποιώντας ένα νέο αλγόριθμο στην έξοδό του για τον υπολογισμό του γραμμικού μέσου του χρόνου στις κορυφές.

Επίσης, προτείνεται μια εντελώς καινούργια μέθοδος η οποία προσφέρει συμπίεση υπό περιορισμούς δικτύου. Όλες οι προταθείσες μέθοδοι συγκρίνονται σε σχέση με την απλή εφαρμογή του αλγόριθμου του Map-Matching. Από τα αποτελέσματα προκύπτει ότι η απλή σειριακή εφαρμογή των αλγορίθμων της συμπίεσης και του ταιριάσματος χάρτη δεν προσφέρει ικανοποιητικά αποτελέσματα. Ο αλγόριθμος συμπίεσης, όταν προηγείται του αλγορίθμου του ταιριάσματος χάρτη, προσφέρει καλύτερους χρόνους, αλλά όχι πιο συμπιεσμένα αποτελέσματα. Η ανάγκη για καινούργια μέθοδο είναι υπαρκτή. Ο αλγόριθμος που προτείνουμε καταφέρνει να δώσει συμπιεσμένα αποτελέσματα, καθώς και δυνατότητα επιλογής βαθμού συμπίεσης. Είναι όμως αρκετά χρονοβόρος και δυσκολεύει την εφαρμογή του σε online δεδομένα.

Συνοπτικά, στην παρούσα μελέτη:

- Παρουσιάζεται το πρόβλημα συμπίεσης τροχιών υπό περιορισμούς δικτύου.
- Προτείνονται δύο μέθοδοι που χρησιμοποιούν ήδη υπάρχοντες αλγόριθμους συμπίεσης και ταιριάσματος χάρτη.
- Παρουσιάζονται προβλήματα που προκύπτουν στην εφαρμογή των αλγορίθμων ταιριάσματος χάρτη και αξιολόγησης ποιότητας και προτείνονται λύσεις.
- Προτείνεται μια εντελώς καινούργια μέθοδος συμπίεσης τροχιών πάνω σε δίκτυο.
- Αξιολογούνται και συγκρίνονται πειραματικά τα αποτελέσματα των διαφορετικών μεθόδων.
- Εξάγονται συμπεράσματα για την εφαρμογή των αλγορίθμων σε διαφορετικά δεδομένα

1.2 Δομή της παρούσας εργασίας

Η δομή της μελέτης έχει ως εξής: Στο Κεφάλαιο 2 παρουσιάζονται κάποιες από τις ήδη υπάρχουσες μεθόδους συμπίεσης τροχιών, ταιριάσματος τροχιάς στο δίκτυο και σύγκρισης τροχιών στο δίκτυο. Στο Κεφάλαιο 3 προτείνονται οι δύο μέθοδοι που συνδυάζουν τους υπάρχοντες αλγόριθμους και κάποια ζητήματα που προκύπτουν κατά τη σύνθεσή τους. Στο Κεφάλαιο 4 φαίνονται τα βήματα που ακολουθήθηκαν για την εφαρμογή της πειραματικής μελέτης, τα σχετικά ζητήματα που προέκυψαν και οι λύσεις τους. Στη συνέχεια παρουσιάζουμε τα πειραματικά αποτελέσματα που προέκυψαν και εξάγονται τα αποτελέσματα/συμπεράσματα της δουλειάς. Παρουσιάζονται γραφικές παραστάσεις τόσο για online όσο και για offline δεδομένα και συγκρίνονται και τα μεταξύ τους αποτελέσματα. Στο Κεφάλαιο 5 προτείνουμε μία καινούργια μέθοδο που θα προσφέρει συμπίεση υπό περιορισμούς δικτύου. Αναλύεται η λειτουργία της και δοκιμάζεται πειραματικά. Στο τέλος της ενότητας αξιολογούνται τα αποτελέσματα συγκριτικά με τα προηγούμενα. Τέλος, στο Κεφάλαιο 6 υπάρχουν κάποια γενικά συμπεράσματα και βήματα που θα μπορούσαν να ακολουθηθούν στο μέλλον.

ΚΕΦΑΛΑΙΟ 2

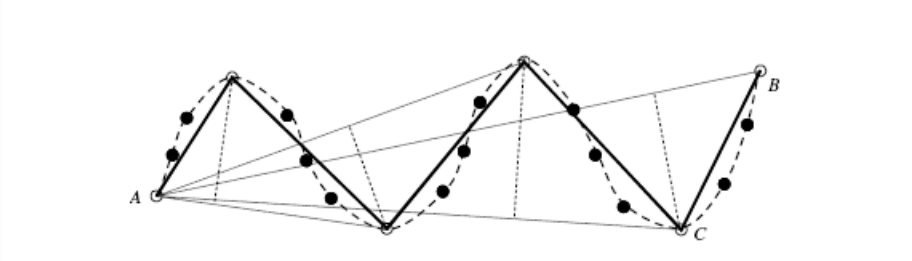
Σχετική έρευνα

2.1 Εισαγωγή

Για τη μελέτη μας χρειάζεται να παρουσιαστούν οι τρέχουσες δουλειές πάνω στα ζητήματα συμπίεσης, ταιριάσματος τροχιάς σε χάρτη και σύγκρισης τροχιών υπό περιορισμούς δικτύου. Στην ενότητα 2.2 παρουσιάζονται οι μέθοδοι συμπίεσης των Douglas-Peckeur [5] και Meratnia & de By [7] για offline δεδομένα και οι μέθοδοι Opening Window (OW) των Meratnia & de By [7], Thresholds και STTrace των Potamias et al. [9] για online δεδομένα. Στην ενότητα 2.3 παρουσιάζεται η μέθοδος ταιριάσματος τροχιάς σε χάρτη των Brakatsoulas et al. [1] για offline δεδομένα και στην ενότητα 2.4 η μέθοδος σύγκρισης τροχιών πάνω σε δίκτυο των Tiakas et al. [10].

2.2 Αλγόριθμοι συμπίεσης τροχιάς

Για τη συμπίεση των δεδομένων τροχιών αντικειμένων, ο πιο διαδεδομένος αλγόριθμος είναι των Douglas-Peckeur [5] που φαίνεται στην Εικόνα 1. Ο αλγόριθμος υπολογίζει την απόσταση κάθε εσωτερικού σημείου από τη γραμμή που ενώνει το πρώτο και το τελευταίο σημείο (AB στο σχήμα). Το σημείο με τη μεγαλύτερη απόσταση, έστω C, χρησιμοποιείται για τη δημιουργία 2 γραμμών, των AC και CB, και αναδρομικά ελέγχονται για την κάθε γραμμή τα υπόλοιπα σημεία. Όταν η απόσταση των εναπομεινάντων σημείων από τις γραμμές που έχουν δημιουργηθεί είναι μικρότερη από μία δεδομένη τιμή, ο αλγόριθμος σταματάει και επιστρέφει αυτές τις γραμμές ως αποτέλεσμα.



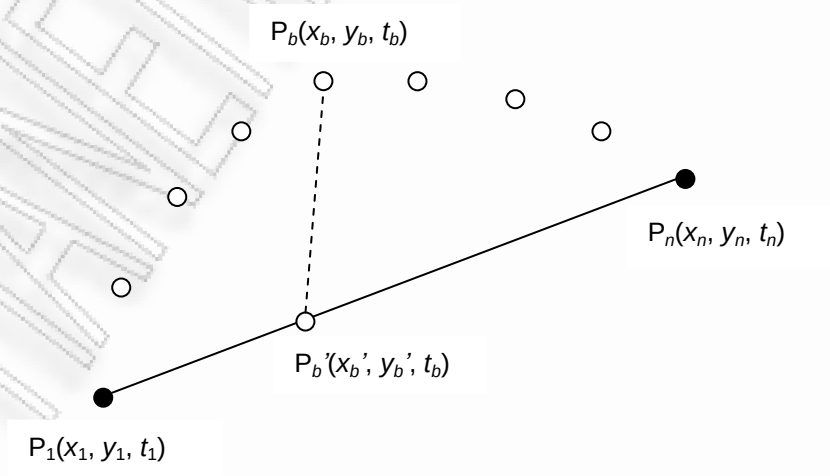
Εικόνα 1 - Douglas-Peckeur Αλγόριθμος [5]

Ο αλγόριθμος των Meratnia & de By [7] εισάγει και την παράμετρο του χρόνου στον παραπάνω αλγόριθμο. Αυτό επιτυγχάνεται χρησιμοποιώντας τη σύγχρονη ευκλείδεια απόσταση (SED) αντί για την απλή απόσταση σημείου από ευθεία όπως φαίνεται στην Εικόνα 2. Αν το σημείο P_b έχει τιμές $<x_b, y_b, t_b>$, και τα άκρα της γραμμής είναι τα σημεία $P_1(x_1, y_1, t_1)$ και $P_n(x_n, y_n, t_n)$ τότε η ζητούμενη απόσταση είναι η απόσταση του σημείου μας από το σημείο P_b με συντεταγμένες $<x_b', y_b', t_b>$ και υπολογίζεται όπως παρακάτω:

$$x_b' = x_1 + \frac{\Delta b}{\Delta n}(x_n - x_1)$$

$$y_b' = y_1 + \frac{\Delta b}{\Delta n}(y_n - y_1)$$

Όπου $\Delta n = t_n - t_1$ και $\Delta b = t_b - t_1$



Εικόνα 2 – Παράδειγμα SED

Αναλυτικά ο αλγόριθμος των Meratnia & de By [7] περιγράφεται στην Εικόνα 3.

Έστω τα σημεία $p_1 \ldots p_n$ και μια ανοχή *tol*.

1. Βρες το σημείο p_b που η SED απόστασή του d_b από τη γραμμή που ενώνει τα $p_1 \ldots p_n$ είναι η μεγαλύτερη. Αυτή η απόσταση είναι η απόσταση του p_b από ένα σημείο p_b' πάνω στη γραμμή που ενώνει τα p_1 και p_n, του οποίου η θέση υπολογίζεται ως εξής:

$$x_b' = x_1 + \frac{\Delta b}{\Delta n}(x_n - x_1)$$

$$y_b' = y_1 + \frac{\Delta b}{\Delta n}(y_n - y_1)$$

Όπου $\Delta n = t_n - t_1$, $\Delta b = t_b - t_1$ και t ο χρόνος κατά τον οποίο βρίσκεται στο αντίστοιχο σημείο.

2. Αν $d_b < tol$
   a. Τότε απόρριψε όλα τα σημεία $p_2 \ldots p_{n-1}$
3. Αλλιώς
   a. Εκτέλεσε από το βήμα 1 για τα σημεία $p_1 \ldots p_b$
   b. Εκτέλεσε από το βήμα 1 για τα σημεία $p_b \ldots p_n$

Εικόνα 3 - Αλγόριθμος Meratnia & de By [7]

Είναι φανερό ότι ο αλγόριθμος των Meratnia & de By [7] δεν μπορεί να χρησιμοποιηθεί για online δεδομένα, αφού για να εκτελεστεί χρειάζεται να διαθέτει όλα τα σημεία $P_1, \ldots, P_n$. Γι αυτό στην ίδια εργασία [7] προτείνεται ένας αλγόριθμος Opening Window (OW), ο οποίος επεξεργάζεται τα δεδομένα καθώς αυτά λαμβάνονται. Αρχικά ορίζεται ένα ευθύγραμμο τμήμα από το πρώτο σημείο της τροχιάς μέχρι το τρίτο. Όσο οι αποστάσεις των εσωτερικών σημείων από το τμήμα είναι μικρότερες από το δεδομένο όριο, προχωράμε το πέρας του ευθύγραμμου τμήματος κατά ένα σημείο. Όταν το όριο της απόστασης είναι να ξεπεραστεί, τότε, είτε το σημείο που βρίσκεται πιο μακριά, είτε το προηγούμενό του, χρησιμοποιούνται ως πέρας αυτού του τμήματος και αρχή του επομένου. Συνεχίζοντας την ίδια διαδικασία λαμβάνεται τελικά η συμπιεσμένη τροχιά.

Αναλυτικά ο αλγόριθμος OW φαίνεται στην Εικόνα 4.

Έστω *tol* η ανοχή και SED η σύγχρονη ευκλείδεια απόσταση όπως ορίστηκε στον αλγόριθμο της εικόνας. Αρχικά ο αλγόριθμος περιμένει να έρθουν τουλάχιστον 3 σημεία P_1, P_2, P_3.

1. Αποθήκευσε το P_1
2. Όρισε ως άκρα του ευθύγραμμου τμήματος AB τα P_1 και P_3.
3. Όσο έρχονται σημεία P_i (i>3)
    a. Υπολόγισε την απόσταση SED όλων των ενδιάμεσων σημείων του A και B από το AB.
    b. Αν η απόσταση αυτή για κάποιο ενδιάμεσο σημείο P_b ξεπερνάει το κατώφλι *tol*
        i. Πρόσθεσε το σημείο P_b
        ii. Όρισε ως άκρα του ευθύγραμμου τμήματος AB τα P_b και P_{b+2}
    c. Αλλιώς
        i. Όρισε ως πέρας του ευθύγραμμου τμήματος AB το P_i
4. Πρόσθεσε το P_n.

Εικόνα 4 - Αλγόριθμος OW

Εναλλακτικά, οι Potamias et al. [9], προτείνουν δύο online αλγορίθμους συμπίεσης, τους Thresholds και STTrace. Στον πρώτο χρησιμοποιείται η τρέχουσα θέση, ταχύτητα και κατεύθυνση του αντικειμένου ώστε να υπολογιστεί μια ασφαλής περιοχή που θα έπρεπε να βρίσκεται το νέο σημείο. Αν όντως κάτι τέτοιο συμβεί, τότε αγνοείται το εισερχόμενο σημείο, αφού αυτό μπορεί να προβλεφθεί από το προηγούμενό του. Για τον υπολογισμό της ασφαλούς περιοχής υπάρχουν δύο δυνατότητες:

1. Υπολογίζεται με βάση την θέση, ταχύτητα, κατεύθυνση του τρέχοντος σημείου, είτε αυτό επιλέχτηκε είτε απορρίφθηκε (Sample-Based)
2. Υπολογίζεται με βάση την θέση, ταχύτητα και κατεύθυνση του τελευταίου σημείου που επιλέχτηκε (Trajectory-Based)

Στην Εικόνα 5 βλέπουμε ένα παράδειγμα του Sample-Based Thresholds. Έστω B, C τα δύο πρόσφατα σημεία της τροχιάς. Ορίζεται u_s η ταχύτητα και φ_s η κατεύθυνση με βάση τα σημεία B και C. Ορίζεται επίσης du_s η ανοχή σε ποσοστό

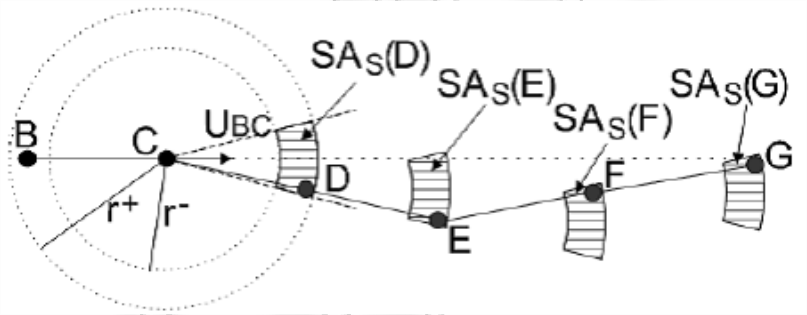
αλλαγής της ταχύτητας του αντικειμένου και $d\varphi_s$ η ανοχή αλλαγής της κατεύθυνσης σε μοίρες. Με βάση αυτές τις παραμέτρους, μπορεί να οριστεί η ασφαλής περιοχή $SA_S(D)$ όπου εκτιμάται να βρεθεί το αντικείμενο στη συνέχεια, χρησιμοποιώντας τη διαφορά dt_s του χρόνου του τελευταίου σημείου με το χρόνο του νέου που μόλις ελήφθη. Τα όρια r^+ και r^- υπολογίζονται ως εξής:

$$r_s^+ = dt_s \times u_s^+$$

$$r_s^- = dt_s \times u_s^-$$

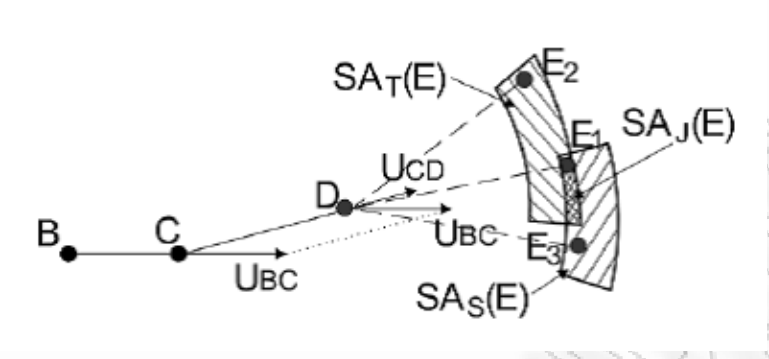
Όπου $u_s^+ = u_s \times (1 + du_s)$ και $u_s^- = u_s \times (1 - du_s)$.

Το άνω και κάτω όριο της περιοχής $SA_S(D)$ υπολογίζεται από τη γωνία φ_s και είναι αντίστοιχα $\varphi_s+d\varphi_s$ και $\varphi_s-d\varphi_s$.



Εικόνα 5 - Sample Based Thresholds [9]

Λόγω αδυναμιών και των δύο προσεγγίσεων, προτείνεται στη συνέχεια ένας συνδυασμός τους, όπου η ασφαλής περιοχή ορίζεται ως η τομή (Joint) των δύο περιοχών που προκύπτουν από τη Sample-Based και την Trajectory Based μέθοδο, όπως φαίνεται στην Εικόνα 6. Η $SA_T(E)$ είναι η ασφαλής περιοχή υπολογισμένη από την Sample-Based μέθοδο με βάση την ταχύτητα U_{CD} και $SA_S(E)$ από την Trajectory Based μέθοδο με βάση την ταχύτητα U_{BC}. Η $SA_J(E)$ είναι η τομή τους.



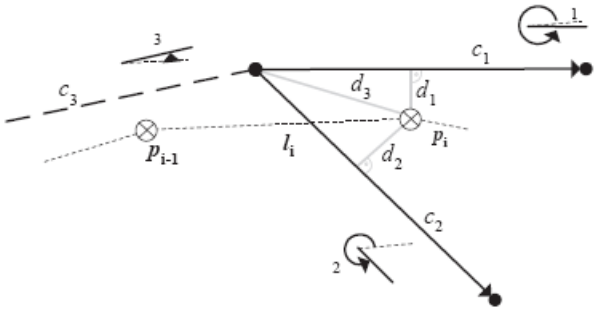
Εικόνα 6 - Joint Ασφαλής Περιοχή [9]

Σύμφωνα με τους Potamias et al. [9], ο αλγόριθμος Thresholds έχει μικρότερο υπολογιστικό κόστος από τον OW και θεωρητικά τα αποτελέσματά του είναι συγκρίσιμα.

Από την άλλη πλευρά, ο STTrace χρησιμοποιεί τα στοιχεία του τρέχοντος σημείου για την εισαγωγή σημείων στο τελικό αποτέλεσμα όπως ο Thresholds, αλλά συγχρόνως επιτρέπει τη διαγραφή προηγούμενων επιλεχθέντων σημείων. Αναλυτικά, θεωρείται σταθερός αριθμός σημείων που μπορούν να αποθηκευτούν ίσος με *M*. Για κάθε σημείο που επιλέγεται να εισαχθεί υπολογίζεται και η σύγχρονη ευκλείδεια απόστασή του (SED) από το ευθύγραμμο τμήμα με αρχή το προηγούμενό του σημείο και πέρας το επόμενό του. Αυτός ο υπολογισμός γίνεται για ένα σημείο όταν εισάγεται το επόμενό του. Όταν ο αριθμός των σημείων υπερβαίνει το *M*, τότε πρέπει να σβηστεί ένα από τα προηγούμενα ή να μην εισαχθεί το νέο σημείο. Η επιλογή γίνεται βρίσκοντας αρχικώς το σημείο με την μικρότερη SED. Στη συνέχεια υπολογίζεται η SED των δύο τελευταίων σημείων με το καινούργιο. Όποιο από τα δύο δίνει τη μικρότερη τιμή και κατ' επέκταση θεωρείται ότι φέρει τη μικρότερη πληροφορία, διαγράφεται, αν είναι σημείο που έχει ήδη επιλεγεί, ή δεν επιλέγεται, αν είναι το καινούργιο.

2.3 Αλγόριθμοι ταιριάσματος τροχιάς σε οδικό χάρτη

Στο πρόβλημα του ταιριάσματος της τροχιάς στον οδικό χάρτη, οι Brakatsoulas et al. [1] προτείνουν τον παρακάτω αλγόριθμο, δεδομένης μιας σειράς από σημεία που παριστάνουν την τροχιά ενός αντικειμένου. Για να ταιριάξει ένα σημείο p_i σε μια ακμή του δικτύου, δεδομένου ότι έχει ήδη ταιριάξει το σημείο p_{i-1} σε μια άλλη, ο αλγόριθμος αρχικά ανακαλύπτει τις γειτονικές ακμές της τρέχουσας θέσης και τις αξιολογεί. Στην Εικόνα 7 φαίνεται ένα παράδειγμα όπου οι c_1 και c_2 είναι οι υποψήφιες ακμές για το σημείο p_i.



Εικόνα 7 – Παράδειγμα εφαρμογής σε γειτονική ακμή [1]

Η αξιολόγηση γίνεται από τα εξής δύο μεγέθη:

$$s_d(p_i, c_j) = \mu_d - a \cdot d(p_i, c_j)^{nd}$$

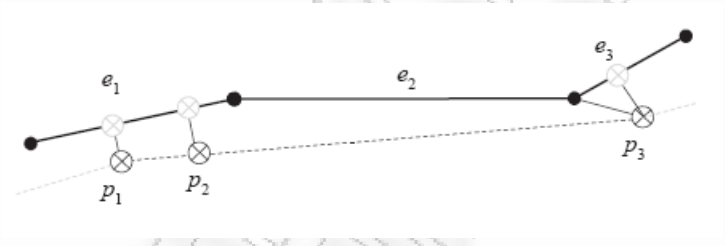
και

$$s_a(p_i, c_j) = \mu_a \cdot \cos\angle(a_i, a_j)^{\mathrm{na}}$$

Όπου το πρώτο μέγεθος αντιστοιχεί στην απόσταση και το δεύτερο στην κατεύθυνση. Οι μ_d, μ_a, n_a, n_d και a είναι σταθεροί όροι. Οι μ_d, μ_a υποδεικνύουν τη μέγιστη βαθμολογία. Επιλέγοντας μεγαλύτερη τιμή του μ_d σε σχέση με το μ_a, δίνεται μεγαλύτερο βάρος στην απόσταση από την κατεύθυνση. Οι παράμετροι n_a, n_d καθορίζουν το ρυθμό αύξησης του αντίστοιχου μεγέθους. Όσο μεγαλύτερο

είναι το άθροισμα s των μεγεθών s_d και s_a, τόσο καλύτερο είναι το ταίριασμα στη συγκεκριμένη ακμή.

Στην περίπτωση που η προβολή του σημείου δε βρίσκεται πάνω σε κάποια από τις γειτονικές ακμές, τότε ο αλγόριθμος δεν προχωράει στο επόμενο σημείο, αλλά θεωρεί την κοντινότερη ακμή μέρος της τροχιάς μας και συνεχίζει την αξιολόγηση με τις γειτονικές αυτής μέχρι να μην υπάρχει καλύτερο σκορ. Ένα παράδειγμα έχουμε στην Εικόνα 8. Βλέπουμε ότι για το σημείο p_3, η προβολή του δε βρίσκεται πάνω στη γειτονική ακμή e_2, οπότε η e_2 θεωρείται μέρος της τροχιάς μας και επανελέγχονται για το ίδιο σημείο οι γειτονικές της ακμές. Μοναδική γειτονική είναι η e_3, πάνω στην οποία βρίσκεται η προβολή του σημείου, οπότε δίνει και καλύτερη βαθμολογία s.

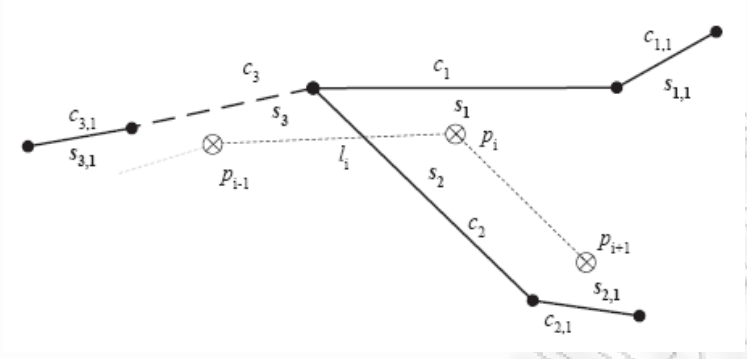


Εικόνα 8 – Παράδειγμα ταιριάσματος σε μη γειτονική ακμή [1]

Για την καλύτερη αξιολόγηση των ακμών προτείνεται η αξιολόγηση αναδρομικά μέχρι n ακμών μπροστά από το σημείο που βρισκόμαστε. Η βαθμολογία s για την κάθε υποψήφια ακμή είναι το άθροισμα όλων των βαθμολογιών για τη διαδρομή n ακμών μπροστά. Δηλαδή:

$$s(p_i, c_j) = \sum_{k,l=0}^{n} s(p_{i+l}, c_{j,k})$$

Στην Εικόνα 9 βλέπουμε ένα σχετικό παράδειγμα. Συγκεκριμένα, για το σημείο p_i θα επιλεγόταν η ακμή c_1 που δίνει καλύτερο σκορ από τη c_2, αν δεν υπήρχε προς τα εμπρός έλεγχος ακμών. Με τον προς τα εμπρός έλεγχο όμως, αξιολογείται και το σημείο p_{i+1}, το οποίο δίνει καλύτερη τιμή για την ακμή $c_{2,1}$ σε σχέση με την ακμή $c_{1,1}$,στην οποία θα βρισκόταν το αντικείμενο αν είχε επιλέξει τη c_1.



Εικόνα 9 – Παράδειγμα προς τα εμπρός ελέγχου ακμών [1]

Αναλυτικά ο αλγόριθμος Map-Matching των Brakatsoulas et al. [1] παρουσιάζεται στην Εικόνα 10.

Έστω ο πίνακας *T* με τα σημεία της τροχιάς, ο πίνακας *M* με τις ακμές και *look* το πλήθος των ακμών της προς τα εμπρός αξιολόγησης και ο πίνακας *Out* το αποτέλεσμα του αλγορίθμου.

1. Για το πρώτο στοιχείο του *T* βρες την κοντινότερη ακμή του *M* και τοποθέτησέ την στον πίνακα *Out*.
2. Όσο ο *T* έχει σημεία
    a. Αν η προβολή του τελευταίου σημείου ήταν πάνω στην ακμή που επιλέχτηκε ή οι γειτονικές ακμές δεν έδωσαν καλύτερο σκορ *s*
        i. Τότε διάβασε το επόμενο σημείο του *T*
        ii. Μηδένισε το σκορ *s*
    b. Για κάθε γειτονική ακμή A της τελευταίας επιλεγμένης
        i. Υπολόγισε το σκορ *s*
        ii. Εκτέλεσε την *LookAhead* με παραμέτρους την ακμή A και την τιμή της *look*.
        iii. Πρόσθεσε το αποτέλεσμα στο *s*
    c. Αν βρεθεί ακμή με καλύτερο σκορ *s*
        i. Τότε πρόσθεσέ την στον πίνακα *Out*

Διαδικασία LookAhead(ακμή A, look)

Έστω *total* το σκορ που υπολογίζεται.

1. Θεώρησε την A επιλεχθείσα ακμή
2. Μέχρι να βρεις *look* ακμές
    a. Αν η προβολή του τελευταίου σημείου ήταν πάνω στην ακμή που επιλέχτηκε ή οι γειτονικές ακμές δεν έδωσαν καλύτερο σκορ *s*
        ii. Τότε διάβασε το επόμενο σημείο του *T*
        iii. Πρόσθεσε το σκορ *s* στο *total*
        iv. Μηδένισε το σκορ *s*
    b. Για κάθε γειτονική ακμή A της τελευταίας επιλεγμένης
        v. Υπολόγισε το σκορ *s*
3. Επίστρεψε το *total*

Εικόνα 10 - Αλγόριθμος Map-Matching

Ο αλγόριθμος λειτουργεί για offline δεδομένα και δε διατηρεί πληροφορία για το χρόνο. Αν θεωρήσουμε *look*=0, οπότε δεν γίνεται κανένας προς τα εμπρός έλεγχος, ο αλγόριθμος θα μπορούσε να εφαρμοστεί και σε online δεδομένα.

2.4 Ομοιότητα τροχιών πάνω σε δίκτυο

Ένα επιπλέον ζήτημα που προκύπτει είναι η εύρεση μιας μεθόδου για τη σύγκριση δύο τροχιών πάνω σε δίκτυο. Οι περιορισμοί του δικτύου καθιστούν τις λύσεις που χρησιμοποιούν ευκλείδεια απόσταση στο χώρο, μη αποδεκτές. Αντίθετα, οι Tiakas et al. [10] προτείνουν έναν αλγόριθμο σύγκρισης τροχιών πάνω στο δίκτυο.

Συγκεκριμένα, ως $c(u_i,u_j)$ ορίζεται το κόστος μετάβασης από τον κόμβο u_i στον κόμβο u_j. Μπορεί να εκφράζει την απόσταση, το χρόνο ή οποιοδήποτε άλλο μέτρο. Με βάση αυτό το κόστος, ορίζεται η απόσταση $d(u_i,u_j)$ μεταξύ δύο κόμβων:

$$d(u_i,u_j) = \begin{cases} 0, c(u_i,u_j) = 0 \ \cap c(u_i,u_j) = 0 \\ \min\{c(u_i,u_j), c(u_j,u_i)\} \\ \max\{c(u_i,u_j), c(u_j,u_i)\}, διαφορετικά \end{cases}$$

Τελικά, η απόσταση $D_{net}(T_a,T_b)$ μεταξύ δύο τροχιών T_a και T_b μήκους m ορίζεται ως:

$$D_{net}(T_a,T_b) = \frac{1}{m} \cdot \sum_{i=1}^{m} (d(u_{ai},u_{bi}))$$

Όσον αφορά στο χρόνο, ως χρονική απόσταση $D_{time}(T_a,T_b)$ μεταξύ δύο τροχιών T_a και T_b ορίζεται:

$$D_{time}(T_a,T_b) = \frac{1}{m-1} \cdot \sum_{i=1}^{m-1} \frac{|(T_a[i+1].t - T_a[i].t) - (T_b[i+1].t - T_b[i].t)|}{\max\{(T_a[i+1].t - T_a[i].t), (T_b[i+1].t - T_b[i].t)\}}$$

Όπου Τ[i].t είναι ο χρόνος που το αντικείμενο βρέθηκε στο i-οστό σημείο της τροχιάς T.

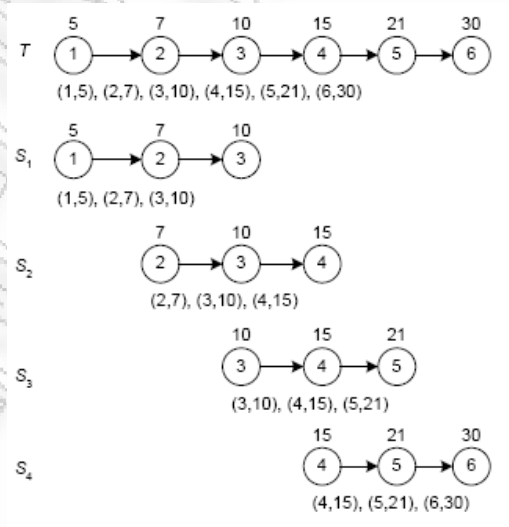
Εύκολα αποδεικνύεται ότι και τα δύο μέτρα απόστασης D_{net} για το χώρο και D_{time} για το χρόνο πληρούν τα κριτήρια του μετρικού χώρου. Η απόδειξη βρίσκεται στο [10].

Για να οριστεί η συνολική απόσταση συνδυάζονται τα δύο μέτρα και έχουμε:

$$D_{total} = W_{net} \cdot D_{net}(T_a, T_b) + W_{time} \cdot D_{time}(T_a, T_b)$$

Όπου W_{net} και W_{time} είναι προκαθορισμένες τιμές βαρύτητας για τις επιμέρους αποστάσεις.

Στην παραπάνω ανάλυση θεωρήθηκε ότι οι δύο εξεταζόμενες τροχιές είναι ίδιου μήκους m. Στη γενική περίπτωση κάτι τέτοιο δεν ισχύει και έστω ότι η τροχιά T_a είναι μήκους $m'<m$. Έστω μ ακέραιος $\mu<m'<m$. Η τροχιά T_b χωρίζεται σε m-μ+1 υπο-τροχιές μήκους μ και η T_a σε m'-μ+1 χρησιμοποιώντας ένα παράθυρο ίδιου μήκους που σταδιακά μετακινείται κατά έναν κόμβο από αριστερά προς τα δεξιά, όπως φαίνεται στην Εικόνα 11 (S_i είναι η κάθε υπο-τροχιά μήκους μ=3 και T η αρχική τροχιά μήκους m=6).



Εικόνα 11 - Χωρισμός σε υπο-τροχιές [10]

Στην περίπτωση που το μήκος της T_a ή και της T_b είναι μικρότερο από μ, τότε επαναλαμβάνεται το τελευταίο σημείο της τροχιάς, μέχρι το μήκος της να γίνει ίσο με μ. Για την συνολική απόσταση $D_{total}(T_a,T_b)$ υπολογίζεται το D_{total} των υποτροχιών.

ΚΕΦΑΛΑΙΟ 3

Προτεινόμενη Μεθοδολογία Συμπίεσης πάνω σε Δίκτυο

3.1 Εισαγωγή

Όπως αναφέρθηκε στο Κεφάλαιο 1, στη σχετική βιβλιογραφία δεν υπάρχει μέθοδος που να προσφέρει συμπίεση τροχιών υπό περιορισμούς δικτύου. Η εφαρμογή του αλγορίθμου ταιριάσματος τροχιάς σε χάρτη προσφέρει ένα βαθμό συμπίεσης, αλλά δεν είναι σχεδιασμένος για αυτό και δεν προσφέρει συμπίεση σε τροχιά που είναι ήδη ταιριασμένη πάνω στο δίκτυο. Στην ενότητα αυτή προτείνουμε δύο μεθόδους που πιθανώς θα πρόσφεραν τη δυνατότητα συμπίεσης τροχιάς υπό περιορισμούς δικτύου.

3.2 Προτεινόμενη μεθοδολογία

Συνδυάζοντας τους αλγόριθμους των Brakatsoulas et al. [1] και Meratnia & de By [7] για offline και online δεδομένα, προσπαθούμε να παράγουμε συμπιεσμένες τροχιές πάνω στο δίκτυο. Οι δύο εναλλακτικές μέθοδοι που προτείνουμε είναι οι εξής:

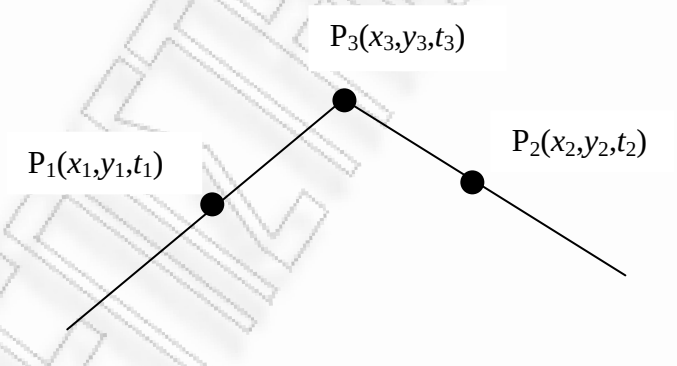
1) Η εφαρμογή του αλγόριθμου συμπίεσης (Comp) και στην έξοδό του να εφαρμοστεί ο αλγόριθμος ταιριάσματος χάρτη (MM) (Comp+MM)
2) Η εφαρμογή του αλγορίθμου ταιριάσματος χάρτη (MM), στην έξοδό του να εφαρμοστεί ο αλγόριθμος συμπίεσης (Comp) και στην συνέχεια εφαρμόζεται πάλι ο αλγόριθμος ταιριάσματος χάρτη. (MM+Comp+MM)

Για την εφαρμογή των αλγορίθμων σε online δεδομένα, χρησιμοποιούμε τον αλγόριθμο OW για τη συμπίεση και τον αλγόριθμο ταιριάσματος χάρτη με κανέναν προς τα εμπρός έλεγχο.

3.3 Ζητήματα που προκύπτουν κατά τη σύνθεση των δύο αλγορίθμων

Ο αλγόριθμος των Brakatsoulas et al. [1] δεν διατηρεί πληροφορία για τον χρόνο στις κορυφές. Γι αυτό, τροποποιείται ώστε, μαζί με την κάθε ακμή του πίνακα *Out* να υπάρχει και πληροφορία για την προβολή του πρώτου και του τελευταίου σημείου που βρίσκεται πάνω σε κάθε ακμή, αν υπάρχει, και ο χρόνος αυτού. Έχοντας πλέον αυτή την πληροφορία, ως τροχιά μπορούμε να θεωρήσουμε ένα σύνολο από κορυφές των ακμών του δικτύου, με χρονικές στιγμές που το αντικείμενό μας βρέθηκε σε αυτές. Έτσι, η έξοδος του αλγορίθμου MM μπορεί να χρησιμοποιηθεί ως είσοδος του αλγορίθμου Comp. Ένας τρόπος υπολογισμού του χρόνου στις κορυφές που προτείνεται είναι ο παρακάτω:

Έστω ότι έχουμε δύο σημεία P_1 και P_2 πάνω σε δύο ακμές και ψάχνουμε την τιμή t_3 για την κορυφή P_3 που συνδέει τις δύο ακμές όπως φαίνεται στην Εικόνα 12.



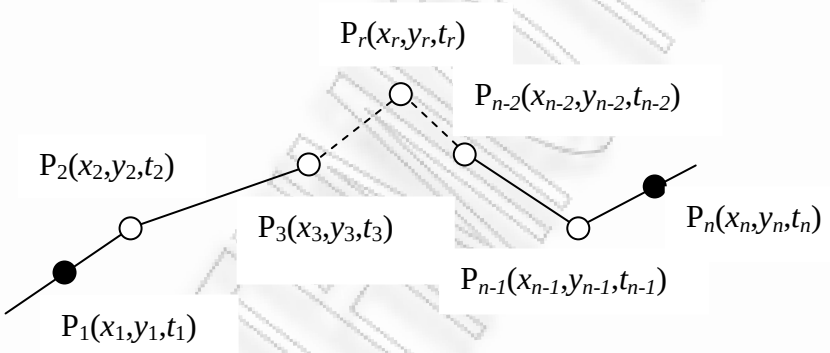
Εικόνα 12 – Παράδειγμα υπολογισμού χρόνου σε κορυφή

Η τιμή t_3 υπολογίζεται ως εξής:

$$t_3 = \frac{d1 \cdot t_2 + d2 \cdot t_1}{d1 + d2}$$

Όπου $d1 = \sqrt{(x_3 - x_1)^2 + (y_3 - y_1)^2}$ η ευκλείδεια απόσταση μεταξύ των σημείων P_1 και P_3 και $d2 = \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2}$ η ευκλείδεια απόσταση μεταξύ των σημείων P_2 και P_3.

Στη γενική περίπτωση που τα σημεία P_1 και P_2 δεν είναι σε γειτονικές ακμές, τότε ως απόσταση $d1$ θεωρούμε την απόσταση του σημείου P_1 από τη γειτονικότερη κορυφή, συν τα μήκη των ακμών μέχρι να φτάσουμε στη ζητούμενη κορυφή, έστω P_r, και $d2$ την απόσταση του σημείου P_n από τη γειτονικότερη κορυφή, συν τα μήκη των ακμών μέχρι να φτάσουμε στην P_r, όπως φαίνεται στην Εικόνα 13. Έστω ότι ψάχνουμε την τιμή του χρόνου στο σημείο P_r γνωρίζοντας τα t_1 και t_n.



Εικόνα 13 - Γενική περίπτωση υπολογισμού χρόνου σε κορυφή

Η τιμή t_r υπολογίζεται ως εξής:

$$tr = \frac{d1 \cdot t_n + d2 \cdot t_1}{d1 + d2}$$

Όπου $d1 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} + \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2} + \cdots + \sqrt{(x_r - x_{r-1})^2 + (y_r - y_{r-1})^2}$ και $d2 = \sqrt{(x_{r+1} - x_r)^2 + (y_{r+1} - y_r)^2} + \cdots + \sqrt{(x_{n-2} - x_{n-1})^2 + (y_{n-2} - y_{n-1})^2} + \sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2}$

Αναλυτικά, ο αλγόριθμος NodePasses που υπολογίζει το χρόνο στις κορυφές φαίνεται στην Εικόνα 14.

Έστω ο πίνακας *Out* με τις ακμές και τις προβολές των πρώτων και των τελευταίων σημείων πάνω στην κάθε ακμή και τους χρόνους αυτών, *T΄* η ζητούμενη έξοδος και *dist* η απόσταση ενός σημείου από ένα άλλο ακολουθώντας την διαδρομή του δικτύου.

1. Πάρε την προβολή του πρώτου σημείου του πίνακα *Out* και τοποθέτησέ το στον πίνακα *T΄*
2. Για κάθε ακμή A_i (i>1) του *Out*
    a. Υπολόγισε την απόσταση dist της προβολής του τελευταίου σημείου της ακμής A_{i-1} από την αρχική κορυφή της A_i ή της πρώτης από τις A_j (j<i-1) που υπάρχει προβολή σημείου σε περίπτωση που A_{i-1} δεν έχει, από την αρχική κορυφή της A_i
    b. Υπολόγισε την απόσταση dist της προβολής του αρχικού σημείου της A_i από την αρχική κορυφή της A_i ή της πρώτης από τις A_j *(j>i)* για τις οποίες υπάρχει προβολή σημείου σε περίπτωση που η τρέχουσα δεν έχει, από την αρχική κορυφή της A_i
    c. Βρες το χρόνο στην αρχική κορυφή της A_i με βάση τους χρόνους κάθε προβολής και της απόστασης και πρόσθεσε στον πίνακα Τ' τις συντεταγμένες της A_i και του υπολογισθέντα χρόνου.
    d. Υπολόγισε την απόσταση dist της προβολής του τελευταίου σημείου της A_i από την τελική κορυφή της A_i ή της πρώτης από τις A_j (j<i) που υπάρχει προβολή σημείου σε περίπτωση που η τρέχουσα δεν έχει, από την τελική κορυφή της A_i
    e. Υπολόγισε την απόσταση dist της προβολής του αρχικού σημείου της A_{i+1} ή της πρώτης από τις A_j *(j>i+1)* που υπάρχει προβολή σημείου σε περίπτωση που η τρέχουσα δεν έχει, από την τελική κορυφή της A_i
    f. Βρες το χρόνο στην τελική κορυφή A_i με βάση τους χρόνους κάθε προβολής και της απόστασης και πρόσθεσε στον πίνακα *T΄* τις συντεταγμένες της A_i και του υπολογισθέντα χρόνου.
3. Πάρε την προβολή του τελευταίου σημείου του πίνακα *Out* και τοποθέτησέ το στον πίνακα *T΄*

Εικόνα 14 - Αλγόριθμος NodePasses

ΚΕΦΑΛΑΙΟ 4

Αξιολόγηση

4.1 Εισαγωγή

Για την παραγωγή και αξιολόγηση των αποτελεσμάτων μας χρειαζόμαστε αρχικές τροχιές αντικειμένων στις οποίες θα εφαρμόσουμε τις προτεινόμενες μεθόδους και θα συγκρίνουμε τα αποτελέσματα. Χρειαζόμαστε αλγόριθμο για την παραγωγή τροχιάς αντικειμένου, αλγόριθμο εισαγωγής θορύβου σε αυτήν την τροχιά ώστε να είναι κοντά σε πραγματικά δεδομένα και αλγορίθμους αξιολόγησης των μεγεθών που ελέγχουμε, όπως η ομοιότητα και ο βαθμός συμπίεσης.

Ακολουθούν αναλυτικά τα βήματα που ακολουθήσαμε, η υλοποίηση των βημάτων αυτών, τα πειραματικά αποτελέσματα που προέκυψαν και η αξιολόγηση των αποτελεσμάτων.

4.2 Μεθοδολογία αξιολόγησης

Αναλυτικά, για την παραγωγή και την αξιολόγηση των αποτελεσμάτων προτείνονται τα ακόλουθα βήματα:

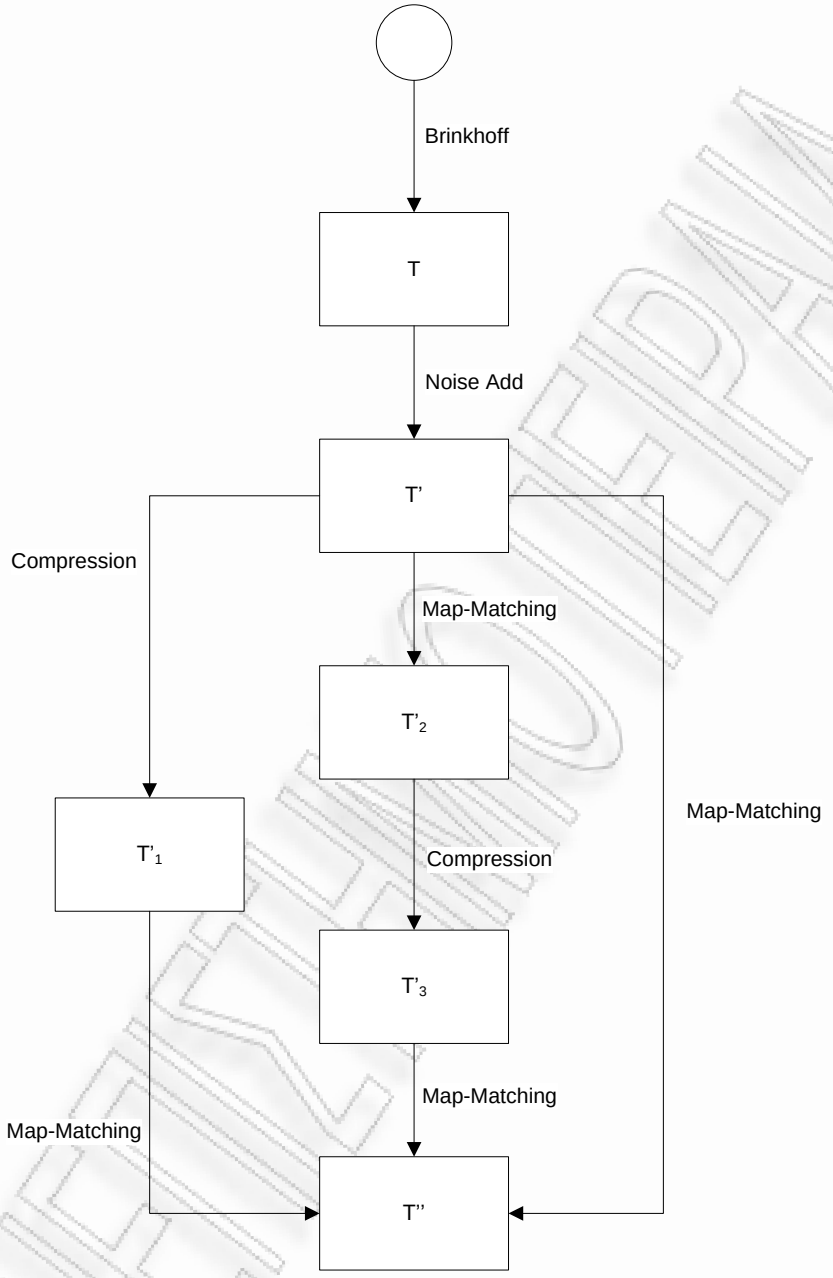
1. Δημιουργείται μία τροχιά *T* πάνω στο δίκτυο
2. Προστίθεται τεχνητός θόρυβος πάνω σε αυτήν την τροχιά κι έτσι παίρνουμε την *T'*
3. Παράγουμε τη ζητούμενη συμπιεσμένη τροχιά πάνω στο δίκτυο *T''* με τις εξής εναλλακτικές μεθόδους:
    a. Συμπιέζουμε την τροχιά *T'* και στη συνέχεια εκτελούμε τον αλγόριθμο map-matching (ταιριάσματος χάρτη) (Comp-MM)

b. Εκτελούμε τον αλγόριθμο map-matching στην τροχιά *T'*, συμπιέζουμε το αποτέλεσμα και στη συνέχεια εκτελούμε ξανά το map-matching. (MM-Comp-MM)
c. Εκτελούμε τον αλγόριθμο map-matching στην *T'* για σύγκριση των αποτελεσμάτων. (MM)

4. Αξιολογούμε την τροχιά *T''* σε σύγκριση με την *T*. Τα κριτήρια είναι ο βαθμός της συμπίεσης (κ1), η ποιότητα/απόσταση από την ταιριασμένη στον χάρτη αρχική τροχιά *T* (κ2) και ο χρόνος εκτέλεσης (κ3) του κάθε βήματος.

Η αξιολόγηση γίνεται σε σχέση με τα αποτελέσματα της απλής εφαρμογής του αλγορίθμου ταιριάσματος χάρτη στα δεδομένα μας. Για την σύγκριση των αποτελεσμάτων όσων αφορά στην ποιότητα χρησιμοποιούμε τον αλγόριθμο των Tiakas et al. [10] κατάλληλα προσαρμοσμένο, ώστε να λειτουργεί με τα δεδομένα μας.

Επίσης, χρησιμοποιούμε τον αλγόριθμο του map-matching με τιμή *look*=0 (κανένας προς τα εμπρός έλεγχος) και τον αλγόριθμο συμπίεσης OW των Meratnia & de By στην ίδια τροχιά Τ'. Με αυτό τον τρόπο μπορούμε να εξάγουμε ασφαλή συμπεράσματα και για την εφαρμογή των παραπάνω μεθόδων σε online δεδομένα. Τα προαναφερθέντα βήματα φαίνονται σχηματικά στην Εικόνα 15.



Εικόνα 15 – Βήματα εφαρμογής

Για το 1ο βήμα χρησιμοποιήθηκε η γεννήτρια του Brinkhoff [2] [3]. Στην παραγόμενη τροχιά προστίθεται θόρυβος και στις δύο συντεταγμένες, ο οποίος ακολουθεί την κανονική κατανομή με μέση τιμή 0 και τυπική απόκλιση 4, όπως αναφέρουν οι Pfoser & Jensen [8]. Αυτό επιτυγχάνεται με την χρήση αλγορίθμου παραγωγής Gaussian θορύβου. Έτσι ολοκληρώνεται και το 2ο βήμα.

Για το 3ο βήμα, εκτός από τους αλγόριθμους για offline δεδομένα εκτελούμε τα επιμέρους βήματα χρησιμοποιώντας τον αλγόριθμο συμπίεσης Opening Window και του map-matching με *look*=0 (κανένας προς τα εμπρός έλεγχος) στην τροχιά μας, ώστε να ελέγξουμε τα αποτελέσματα για online δεδομένα. Αυτό επιτυγχάνεται θεωρώντας τα σημεία της παραγόμενης τροχιάς μη γνωστά εκ των προτέρων.

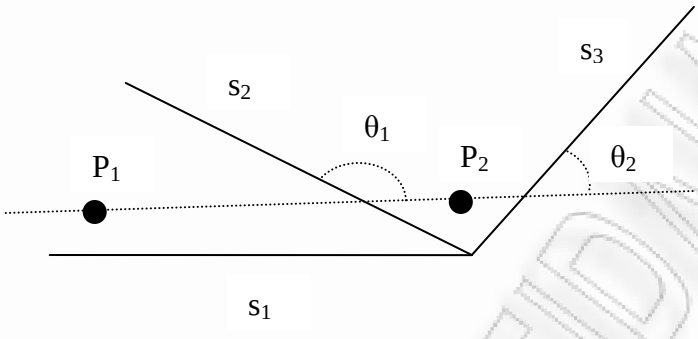
Για το 4ο βήμα, έχουμε ως ανεξάρτητες μεταβλητές τις τιμές της ανοχής του αλγορίθμου της συμπίεσης και τον αριθμό των προς τα εμπρός ακμών που ελέγχουμε στον αλγόριθμο του ταιριάσματος χάρτη. Εξαρτημένες μεταβλητές είναι τα τρία κριτήρια που ορίσαμε. Για το κριτήριο του χρόνου μετράμε την εκτέλεση του κάθε αλγόριθμου ξεχωριστά σε κάθε εναλλακτικό τρόπο του 3ου βήματος. Για την συμπίεση, συγκρίνουμε την παραγόμενη ταιριασμένη στον χάρτη τροχιά με την αρχική, όπως προέκυψε από την γεννήτρια Brinkhoff. Για την αξιολόγηση της ποιότητας, εφαρμόζουμε μία παραλλαγή του αλγορίθμου των Tiakas et al. [10], όπως περιγράφεται στην ενότητα 4.4, στην τελική τροχιά και στο αποτέλεσμα του αλγόριθμου του map-matching στην αρχική τροχιά.

4.3 Ζητήματα του αλγορίθμου ταιριάσματος χάρτη

Οι Brakatsoulas et al. [1] για τον υπολογισμό του δείκτη κατεύθυνσης *sa* χρησιμοποιούν το *cos*(θ) της γωνίας της τροχιάς μας και των υποψήφιων ακμών. Συγκεκριμένα, ο τύπος είναι όπως προαναφέρθηκε:

$$s_a(p_i, c_j) = \mu_a \cdot \cos(a_i, a_j)^{\text{na}}$$

Όπως αναφέρουν, για την παράμετρο *na* χρησιμοποιούν την τιμή 4 για τα δεδομένα τους. Υψώνοντας όμως το *cos*(θ) σε άρτια δύναμη, παίρνουμε θετικό αποτέλεσμα άσχετα από το πρόσημο του συνημίτονου. Αυτό θα έδινε λάθος αποτελέσματα σε περίπτωση όπως φαίνεται στην Εικόνα 16.



Εικόνα 16 – Παράδειγμα λανθασμένης επιλογής ακμής

Συγκεκριμένα, έστω ότι για το σημείο P_1 ο αλγόριθμος ορίζει ως ακμή την s_1. Για την αξιολόγηση του σημείου P_2 θα χρησιμοποιηθεί το συνημίτονο των γωνιών θ_1 και θ_2 που σχηματίζουν τα P_1, P_2 με τις υποψήφιες ακμές s_2 και s_3 αντίστοιχα, μιας και τις αποστάσεις του P_2 από τις δύο ακμές τις θεωρούμε ίσες. Η ακμή s_3 θα πρέπει να δώσει καλύτερο αποτέλεσμα επειδή $\theta_2<\theta_1$. Αλλά, σε απόλυτες τιμές, η s_2 βαθμολογείται ως καλύτερη, αφού η γωνία θ_1 είναι μεγαλύτερη από 90 μοίρες και δίνει αρνητική τιμή συνημίτονου μεγαλύτερη κατ' απόλυτη τιμή από αυτή που δίνει η θ_2. Αν ο αλγόριθμος λειτουργήσει με κανέναν προς τα εμπρός έλεγχο, δηλαδή τιμή *look*=0, θα επιλέξει λανθασμένη ακμή, την s_2 αντί της s_3. Για τα πειράματά μας, θα ορίσουμε την τιμή της παραμέτρου *na* ίση με 3, ώστε να διατηρείται το πρόσημο του συνημίτονου.

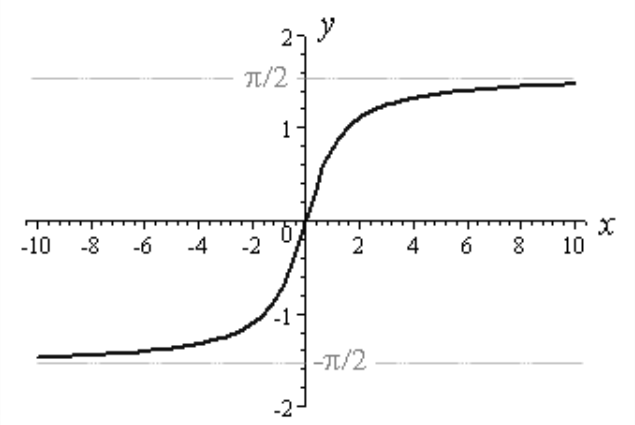
4.4 Ζητήματα στο θέμα της ομοιότητας τροχιών πάνω σε δίκτυο

Ο αλγόριθμος των Tiakas et al. [10] δεν μπορεί να χρησιμοποιηθεί αυτούσιος γιατί στην περίπτωση που ο γράφος δεν είναι κατευθυνόμενος, τότε $c(u_i,u_j)$= $c(u_j,u_i)$) και η απόσταση $d(u_i,u_j)$ που ορίζεται θα δίνει πάντα τιμή 0 αν ο u_i ταυτίζεται με τον κόμβο u_j και 1 σε κάθε άλλη περίπτωση. Επομένως δεν έχουμε ένα ακριβές μέτρο σύγκρισης για τροχιές που μπορεί, για παράδειγμα, να είναι

απλώς μετατοπισμένες κατά ένα κόμβο. Γι αυτό προτείνουμε μια εναλλακτική λύση.

Συγκεκριμένα, ορίζουμε ως $c(u_i,u_j)$ το κόστος να μεταβούμε από τον κόμβο u_i στον κόμβο u_j χρησιμοποιώντας τον αλγόριθμο shortest-path του Dijkstra [4]. Έτσι υπολογίζουμε την απόσταση δύο κόμβων υπό περιορισμούς δικτύου.

Με βάση αυτό το κόστος, πρέπει να ορίσουμε την απόσταση $d(u_i,u_j)$ μεταξύ δύο κόμβων. Χρειαζόμαστε $0\leq d(u_i,u_j)<1$ για πεδίο τιμών $0\leq c(u_i,u_j)<\infty$. Για να εξασφαλίσουμε κάτι τέτοιο, μπορούμε να χρησιμοποιήσουμε την τριγωνομετρική συνάρτηση του τόξου εφαπτομένης ($arctan(x)$), της οποίας η γραφική παράσταση είναι όπως φαίνεται στην Εικόνα 17.



Εικόνα 17 – Γραφική παράσταση του τόξου εφαπτομένης

Παρατηρούμε ότι η συνάρτηση του τόξου εφαπτομένης μας προσφέρει τη δυνατότητα, για τιμή που αγγίζει το άπειρο, να επιστρέφει τιμή κοντά στο π/2 και για τιμή 0, να επιστρέφει 0. Καταφέρνουμε έτσι να εξασφαλίσουμε την απαίτηση μας. Επίσης, βλέπουμε ότι η καμπύλη της συνάρτησης αυξάνεται γρήγορα για τιμές του x μικρότερες του 2, και στη συνέχεια έχουμε μικρό ρυθμό αύξησης. Αυτή η ιδιότητα είναι χρήσιμη για τα δεδομένα μας, μιας και χρειαζόμαστε λεπτομέρεια όταν συγκρίνουμε τροχιές με μικρές διαφορές, η οποία δεν είναι αναγκαία σε τροχιές με μεγάλη απόσταση.

Μας ενδιαφέρουν οι τιμές για θετικά *x*. Επίσης χρειαζόμαστε ασύμπτωτη όχι στην τιμή π/2 αλλά στην τιμή 1. Επομένως μία αποδεκτή λύση για την $d(u_i,u_j)$ θα ήταν:

$$d(u_i, u_j) = \frac{2}{\pi}\arctan(c(u_i, u_j))$$

Και τελικά η απόσταση μεταξύ δύο τροχιών T_a και T_b μήκους m ορίζεται ως:

$$D_{net}(T_a, T_b) = \frac{1}{m} \cdot \sum_{i=1}^{m} (d(u_{ai}, u_{bi}))$$

Όσον αφορά στο χρόνο, ως χρονική απόσταση μεταξύ δύο τροχιών T_a και T_b ορίζουμε τη χρονική διαφορά των δύο αντικειμένων για να μεταβούν από τον έναν κόμβο στον επόμενο:

$$D_{time}(T_a, T_b) = \frac{1}{m-1} \cdot \frac{2}{\pi} \cdot \sum_{i=1}^{m-1} \arctan(|(T_a[i+1].t - T_a[i].t) - (T_b[i+1].t - T_b[i].t)|)$$

Όπου $T[i].t$ είναι ο χρόνος που το αντικείμενο βρέθηκε στο i-στό σημείο της τροχιάς μας.

Για να οριστεί η συνολική απόσταση, συνδυάζουμε τα δύο μέτρα όπως στην εργασία [10] και έχουμε:

$$D_{total} = W_{net} \cdot D_{net}(T_a, T_b) + W_{time} \cdot D_{time}(T_a, T_b)$$

Όπου W_{net} και W_{time} είναι προκαθορισμένες τιμές βαρύτητας. Η γεννήτρια θορύβου δεν επηρεάζει τις τιμές των χρόνων κάθε σημείου, οπότε θεωρούμε ότι τα λάθη που θα προκύψουν όσον αφορά στο χρόνο είναι ανάλογα με αυτά που θα προκύψουν στο χώρο. Επομένως θεωρούμε και τα δύο ισοβαρή και ορίζουμε $W_{net}=W_{time}=0{,}5$.

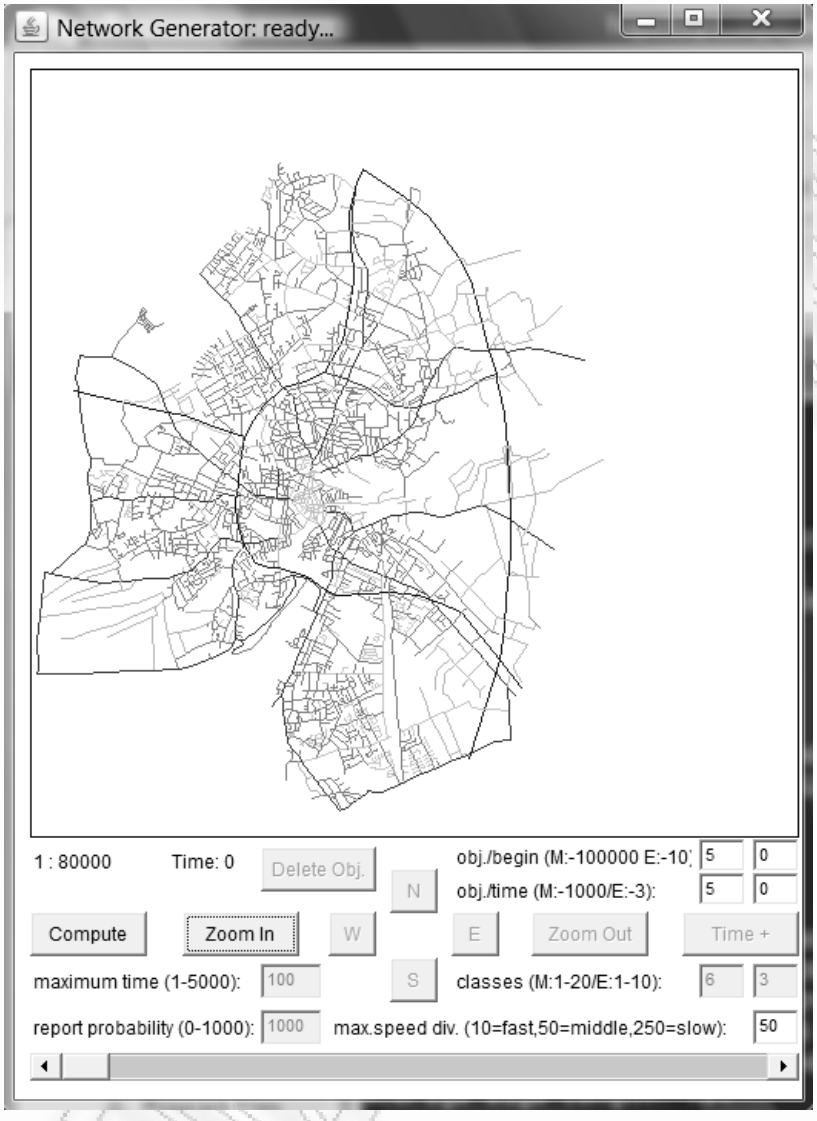
Στην περίπτωση που οι δύο τροχιές δεν είναι ίδιου μήκους, ακολουθούμε ανάλογη λογική της εργασίας [10]. Έστω ότι η τροχιά T_a είναι μήκους $\mu<m$. Τότε χωρίζουμε την τροχιά T_b σε m-μ+1 υπο-τροχιές μήκους μ, χρησιμοποιώντας ένα

παράθυρο μήκους μ που σταδιακά προχωράμε κατά έναν κόμβο από αριστερά προς τα δεξιά.

Υπολογίζουμε το D_{total} της τροχιάς T_a με κάθε υπο-τροχιά της T_b και ως τελική τιμή D_{total} ορίζουμε το μέσο όρο των επί μέρους αξιολογήσεων.

4.5 Ρυθμίσεις και θέματα υλοποίησης

Η υλοποίηση των αλγορίθμων έγινε σε περιβάλλον Java με την χρήση του NetBeans. Για την δημιουργία τροχιάς, χρησιμοποιείται ο Brinkhoff Generator [3] με τα δοκιμαστικά αρχεία *oldenburgGen* για το δίκτυο και *propOL.txt* για τις ρυθμίσεις (εκτέλεση του *runDataGeneratorOL.bat*). Στην επιλογή *maximum time* βάζουμε τιμή 300 και παράγουμε 3 διαφορετικές τροχιές για τιμές *max. speed div.* 75, 150 και 250 (μικρότερη τιμή της παραμέτρου *max. speed div.* ισοδυναμεί με μεγαλύτερη ταχύτητα του αντικειμένου, δηλαδή λιγότερα σημεία σε κάθε ακμή). Η παραγόμενη τροχιά αποθηκεύεται σε ένα αρχείο κειμένου που έχει οριστεί στο αρχείο *runDataGeneratorOL.bat*. Στην Εικόνα 18 παρουσιάζεται η οθόνη της εφαρμογής Brinkhoff Generator.



Εικόνα 18 - Brinkhoff Generator

Στην εφαρμογή που υλοποιήσαμε για να αξιολογήσουμε τις προτεινόμενες μεθόδους, τα δεδομένα διαβάζονται από τα 3 αρχεία, τα 2 του δικτύου και αυτό της παραγόμενης τροχιάς. Αυτό της τροχιάς που είναι της μορφής {*type of event, object ID, class, time stamp, x, y*}, των κορυφών που είναι της μορφής {*node name length, node name, id, x, y*} και των ακμών που είναι της μορφής {*node1-id, node2-id, edge name length, edge name, edge id, edge class*} [7]. Όλα τα δεδομένα κρατούνται σε δομές τύπου Vector.

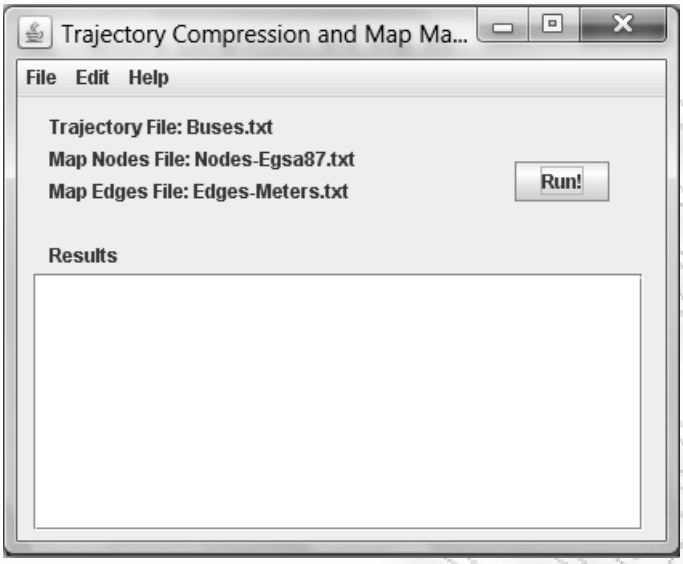
Αναλυτικότερα, για τη δομή της τροχιάς χρησιμοποιούνται τριάδες του τύπου {*x, y, t*}. Η έξοδος του αλγόριθμου του Map-Matching αποτελείται από δεδομένα της

μορφής {*x1, y1, x2, y2, xfirst, yfirst, tfirst, xlast, ylast, tlast*}, όπου είναι οι συντεταγμένες της ακμής {*x*1, *y*1, *x*2, *y*2} και οι συντεταγμένες της προβολής του πρώτου και του τελευταίου σημείου πάνω στην ακμή αυτή μαζί με τους αντίστοιχους χρόνους. Αυτή η έξοδος εισάγεται στον αλγόριθμο υπολογισμού του χρόνου στις κορυφές ο οποίος τελικά παράγει αποτελέσματα της μορφής της τροχιάς.

Για το χάρτη του δικτύου, χρησιμοποιήθηκε δομή που αποτελείται από δεδομένα της μορφής {*id, x, y, πίνακας γειτόνων*}. Συγκεκριμένα, κάθε κορυφή με *id* έστω *a* τοποθετείται στην θέση *a* του Vector με στοιχεία τις συντεταγμένες της. Για κάθε ακμή που έχει κορυφή την *a* και δεύτερη κορυφή έστω *b*, τοποθετείται το *id* της *b* στον πίνακα των γειτόνων της *a* και το *id* της *a* στον πίνακα γειτόνων της *b*. Με αυτόν τον τρόπο επιτυγχάνεται γρήγορη αναζήτηση των ακμών του δικτύου και των γειτονικών τους.

Οι αλγόριθμοι των Meratnia & de By [7] και υπολογισμού του χρόνου στις κορυφές χρησιμοποιούν αναδρομή. Ο αλγόριθμος των Brakatsoulas et al. [1] υλοποιείται χωρίς χρήση αναδρομής για την αξιολόγηση προς τα εμπρός των σημείων. Στην αρχικοποίηση του γίνεται σειριακή αναζήτηση όλων των ακμών για την εύρεση της κοντινότερης ακμής στο πρώτο σημείο της τροχιάς.

Για τη σύγκριση των τροχιών χρειάζεται η εκτέλεση του αλγόριθμου shortest-path του Dijkstra [4]. Για καλύτερα αποτελέσματα εκτελούμε από πριν τον αλγόριθμο για όλα τα σημεία της ταιριασμένης στον χάρτη εκδοχής της αρχικής τροχιάς που έχει παραχθεί από την γεννήτρια του Brinkhoff και κρατάμε τις αποστάσεις αυτών των κόμβων από τους υπόλοιπους, σε ένα δισδιάστατο πίνακα τύπου Vector. Έτσι ο αλγόριθμος σύγκρισης τροχιών δεν εκτελεί τον αλγόριθμο shortest-path, απλώς ανατρέχει στον αντίστοιχο πίνακα για να λάβει τις τιμές που χρειάζεται. Αποφεύγουμε έτσι τυχών επικαλυπτόμενους υπολογισμούς.



Εικόνα 19 – Οθόνη εφαρμογής

Το γραφικό περιβάλλον του προγράμματος είναι αυτό που φαίνεται στην Εικόνα 19. Ο χρήστης από το μενού File επιλέγει τα 3 αρχεία της εφαρμογής, της τροχιάς, των κόμβων και των ακμών του χάρτη. Πατώντας το κουμπί Run εκτελούνται οι αλγόριθμοι και αξιολογούνται για τιμές της μεταβλητής *look* 2, 4 και 6 (μεταβλητή για το πόσες ακμές προς τα εμπρός ελέγχει ο αλγόριθμος ταιριάσματος χάρτη) και *tol* 10, 50 και 90 (ανοχή του αλγόριθμου της συμπίεσης) με βάση τον χρόνο εκτέλεσης, τον βαθμό της συμπίεσης και την ομοιότητα με την ταιριασμένη στον χάρτη αρχική τροχιά (για το map-matching της αρχικής τροχιάς χρησιμοποιείται τιμή *look*=4). Επίσης εκτελούνται οι αλγόριθμοι OW και map-matching με *look*=0 για αξιολόγηση αποτελεσμάτων των αλγορίθμων με online δεδομένα. Μεταβλητή σε αυτή την περίπτωση είναι η ανοχή *tol* που παίρνει τιμές 10, 50 και 90. Τα αποτελέσματα εμφανίζονται στο Results.

Ο χρήστης, τέλος, έχει και τη δυνατότητα να αλλάξει τις προεπιλεγμένες τιμές παραμέτρων από το μενού Edit. Συγκεκριμένα για την τροχιά μπορεί να επιλέξει το *object-id* και το *object class* και για τον αλγόριθμο του ταιριάσματος χάρτη τις τιμές των μ_d, n_d, μ_a, n_a και a. Προεπιλεγμένες τιμές σε αυτές τις παραμέτρους ορίζουμε τις χρησιμοποιηθέντες στην εργασία [1] μ_d=10, n_d=1,4, μ_a=10, n_a=3

(αντί για 4 που χρησιμοποιείται ώστε να διατηρηθεί το πρόσημο του συνημίτονου) και *a*=0,17.

4.6 Πειραματικά Αποτελέσματα

Για τα αποτελέσματα χρησιμοποιούμε τρεις παραγόμενες τροχιές από τη γεννήτρια Brinkhoff, μία με τιμή *max. speed div.*=75, δηλαδή με αρκετά απομακρυσμένα σημεία, μία με τιμή *max. speed div.*=150), δηλαδή με σημεία με μικρότερη απόσταση μεταξύ τους και μία με τιμή 250 (προεπιλεγμένη τιμή).

Παράμετροι που μπορούμε να ρυθμίσουμε είναι η μεταβλητή *look* του map-matching αλγόριθμου, στην οποία ορίζουμε τιμές 2, 4, 6 με προεπιλεγμένη τιμή 4, και η ανοχή *tol* του αλγόριθμου συμπίεσης για τιμές 10, 50 και 90 με προεπιλεγμένη τιμή 10.

Συνοπτικά, οι παράμετροι της πειραματικής μελέτης εμφανίζονται στον Πίνακας 1. Υπογραμμισμένες είναι οι προεπιλεγμένες τιμές.

Πίνακας 1 - Παράμετροι πειραματικής μελέτης

Παράμετρος	Τιμή		
Max. speed div.	75	150	<u>250</u>
look	2	<u>4</u>	6
tol	<u>10</u>	50	90

Οι παράμετροι που θέλουμε να αξιολογήσουμε είναι ο χρόνος εκτέλεσης, ο βαθμός συμπίεσης του αποτελέσματος σε σχέση με τα αρχικά σημεία που προέκυψαν από τη γεννήτρια Brinkhoff και η απόσταση της παραγόμενης τροχιάς από την ταιριασμένη στον χάρτη έκδοση της τροχιάς που προέκυψε από τη γεννήτρια.

Οι αλγόριθμοι που αξιολογούνται είναι οι δύο που προτάθηκαν, MM-Comp-MM και ο Comp-MM, σε αντιπαραβολή με το map-matching αλγόριθμο των Brakatsoulas et al. [1].

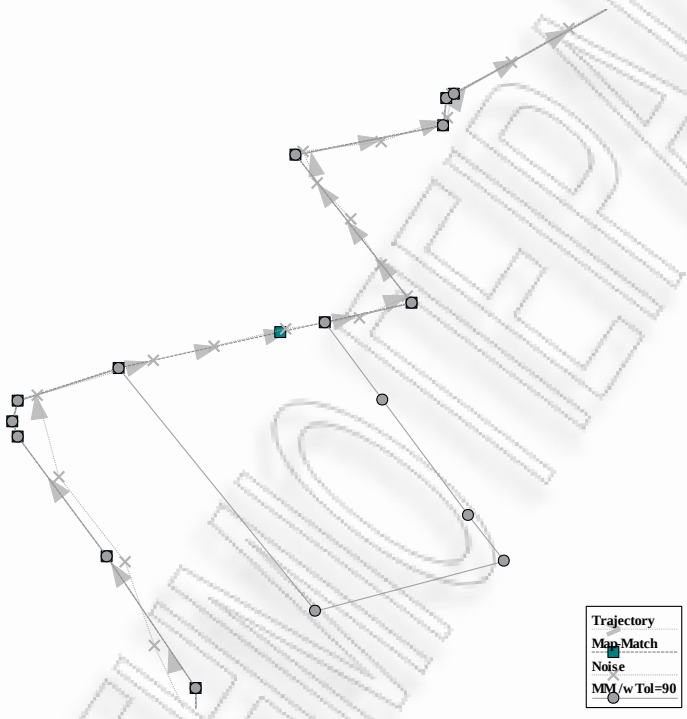
Οι αλγόριθμοι εφαρμόζονται τόσο σε offline, όσο και σε online δεδομένα. Για τα online δεδομένα θεωρούμε ότι τα σημεία δεν είναι όλα γνωστά από πριν και εκτελούμε για MM το map-matching αλγόριθμο των Brakatsoulas et al. [1] με κανέναν προς τα εμπρός έλεγχο (τιμή παραμέτρου *look*=0) και για Comp τον αλγόριθμο OW των Meratnia & de By [7].

Ακολουθούν δύο παραδείγματα εκτέλεσης των αλγορίθμων για διαφορετικές τιμές της *max. speed div.* και διαφορετικές μεθόδους.

Εικόνα 20 - Παράδειγμα αποτελέσματος για *max speed div.*=150

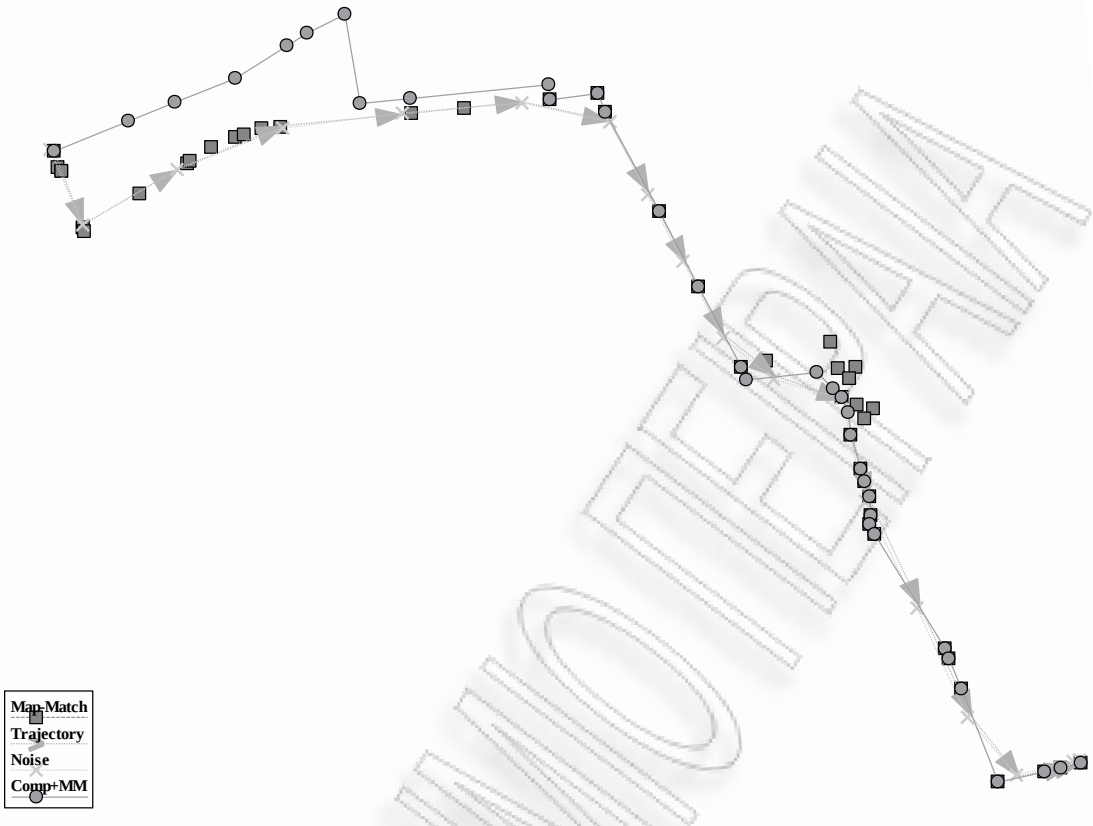
Στην Εικόνα 20, Trajectory είναι η αρχική τροχιά που παράχθηκε από τη γεννήτρια Brinkhoff, noise η ίδια τροχιά μετά την προσθήκη τεχνητού θορύβου,

map-match το αποτέλεσμα του αλγορίθμου ταιριάσματος χάρτη στην αρχική τροχιά πριν τον θόρυβο και MM w/ Tol=90 η εφαρμογή της μεθόδου Comp+MM για ανοχή *tol*=90.



Εικόνα 21 – Λεπτομέρεια Εικόνα 20

Στην Εικόνα 21 με τη λεπτομέρεια της προηγούμενης τροχιάς, βλέπουμε ότι η μέθοδος Comp+MM δίνει λάθος διαδρομή λόγω του μεγάλου βαθμού συμπίεσης των αρχικών σημείων με ανοχή *tol*=90.



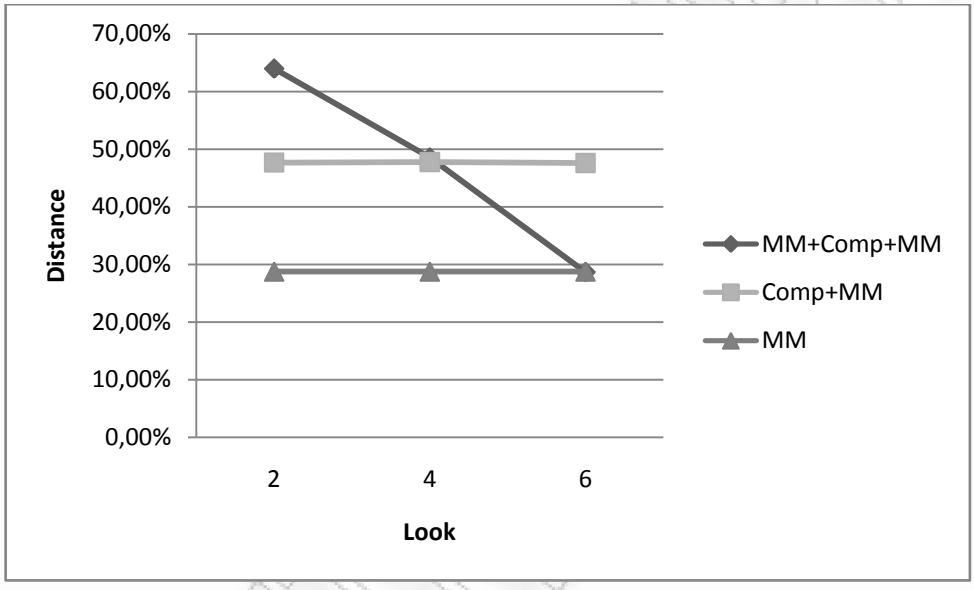
Εικόνα 22 - Παράδειγμα εκτέλεσης για *max speed div.*=75

Ένα δεύτερο παράδειγμα φαίνεται στην Εικόνα 22. Παρατηρούμε σε δύο σημεία τις διαδρομής να υπάρχουν αρκετά λάθη, δεδομένου ότι τα σημεία της δειγματοληψίας ήταν λίγα. Ο αλγόριθμος Comp+MM έδειξε διαφορετική διαδρομή.

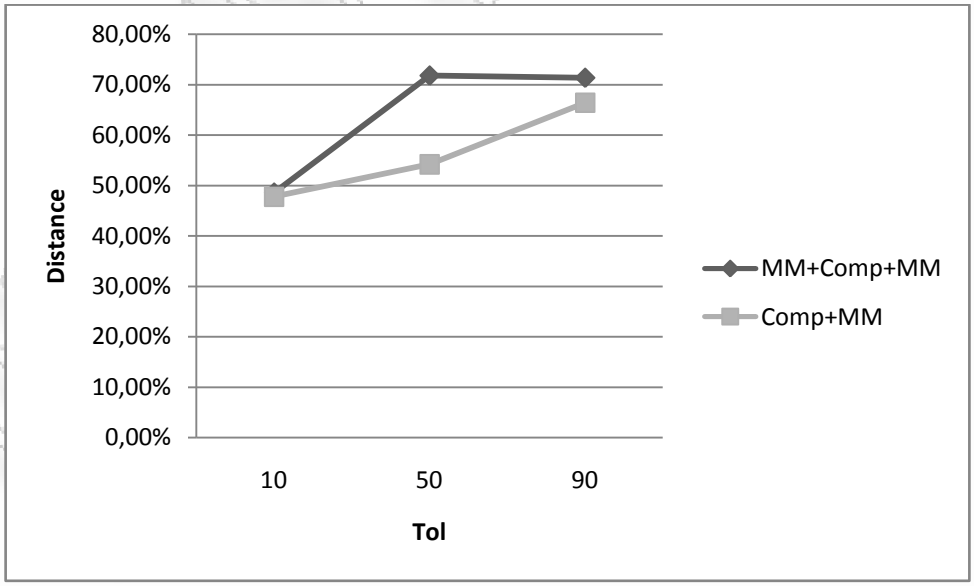
Ακολουθούν γραφικές παραστάσεις με τα αναλυτικά αποτελέσματα τόσο για offline όσο και για online δεδομένα. Στο Παράρτημα Α βρίσκονται όλα τα πειραματικά αποτελέσματα σε πίνακες.

4.6.1 Αποτελέσματα για offline δεδομένα

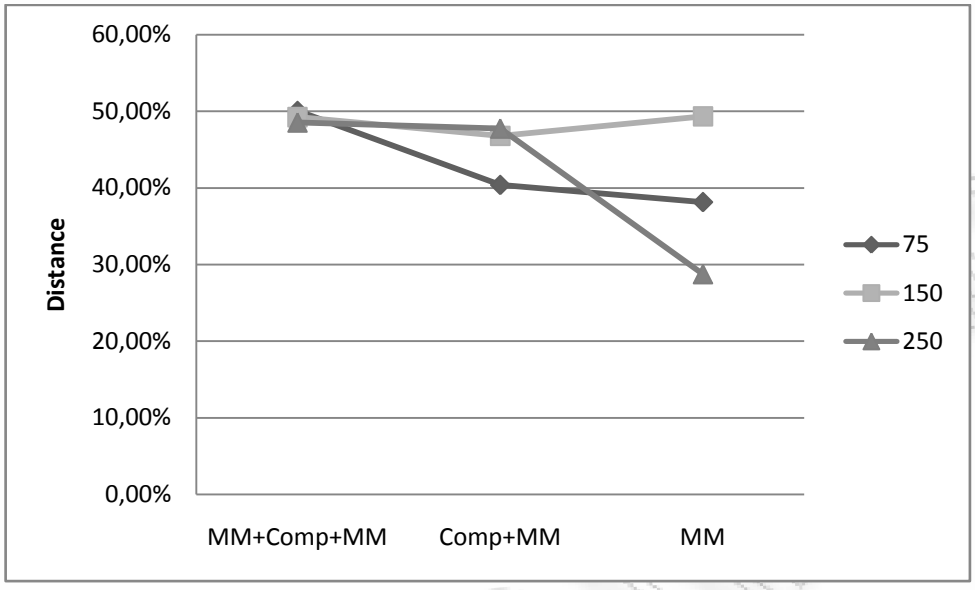
Ακολουθούν οι γραφικές απεικονίσεις των αποτελεσμάτων για την εφαρμογή των μεθόδων σε offline δεδομένα.



(α)



(β)



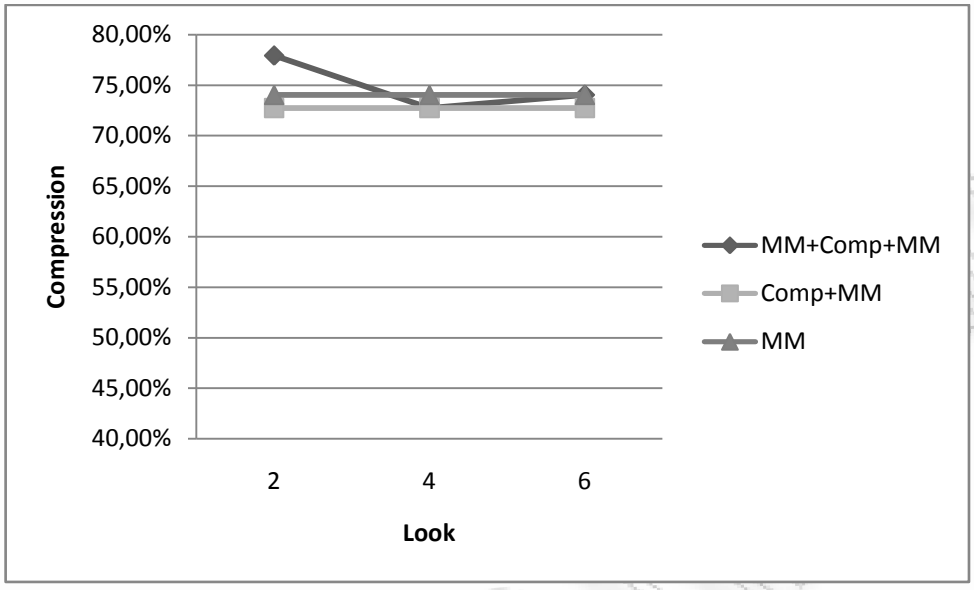
(γ)

Εικόνα 23 - Απόσταση για διαφορετικές τιμές (α) της παραμέτρου *look* (β) της παραμέτρου *tol* και (γ) της παραμέτρου *max. speed div.*

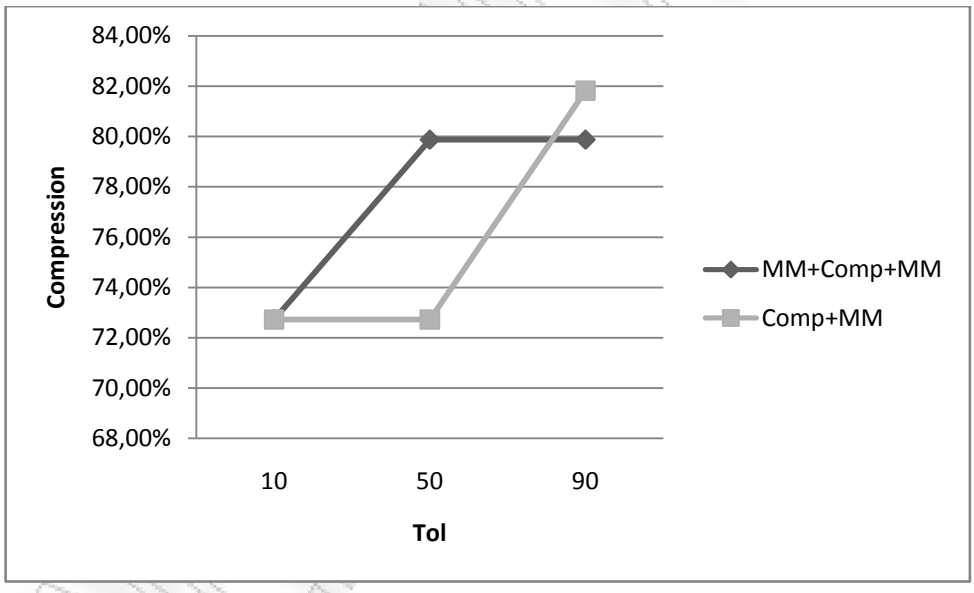
Στην Εικόνα 23(α) παρατηρούμε ότι για έλεγχο περισσοτέρων ακμών προς τα εμπρός έχουμε ελαφριά βελτίωση της ποιότητας, όπως φαίνεται από τη διπλή εφαρμογή του αλγόριθμου ταιριάσματος χάρτη στην μέθοδο MM+Comp+MM.

Στην Εικόνα 23(β) φαίνεται η τιμή της παραμέτρου ανοχής για τον αλγόριθμο συμπίεσης να επηρεάζει, όπως ήταν αναμενόμενο, την ποιότητα του τελικού αποτελέσματος. Μεγαλύτερη τιμή εξάγει αποτελέσματα χαμηλότερης ποιότητας.

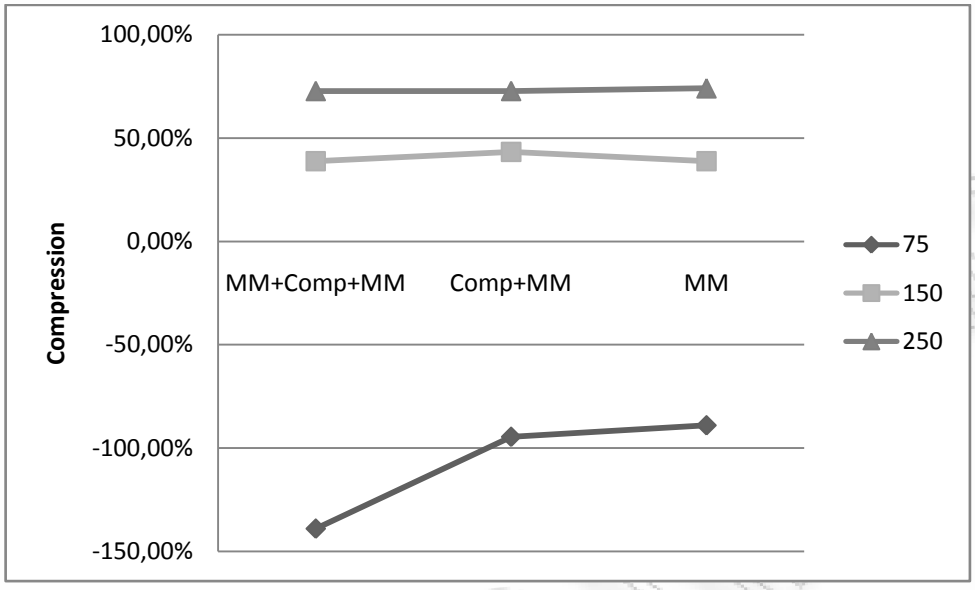
Στην Εικόνα 23(γ) παρατηρούμε ότι η επίδοση της κάθε μεθόδου όσον αφορά την ποιότητα δεν εξαρτάται γενικά από τον αριθμό των σημείων. Πιο συγκεκριμένα, όσον αφορά στην μέθοδο του ταιριάσματος χάρτη, παρατηρούμε ότι τροχιές με σημεία αρκετά αραιά, είτε τροχιές με πολύ κοντινά σημεία, δε δίνουν καλύτερα αποτελέσματα από τροχιές με ενδιάμεσο αριθμό σημείων.


80,00%
75,00%
70,00%
65,00%
60,00%
55,00%
50,00%
45,00%
40,00%
Compression
2
4
6
Look
MM+Comp+MM
Comp+MM
MM

(α)


84,00%
82,00%
80,00%
78,00%
76,00%
74,00%
72,00%
70,00%
68,00%
Compression
10
50
90
Tol
MM+Comp+MM
Comp+MM

(β)



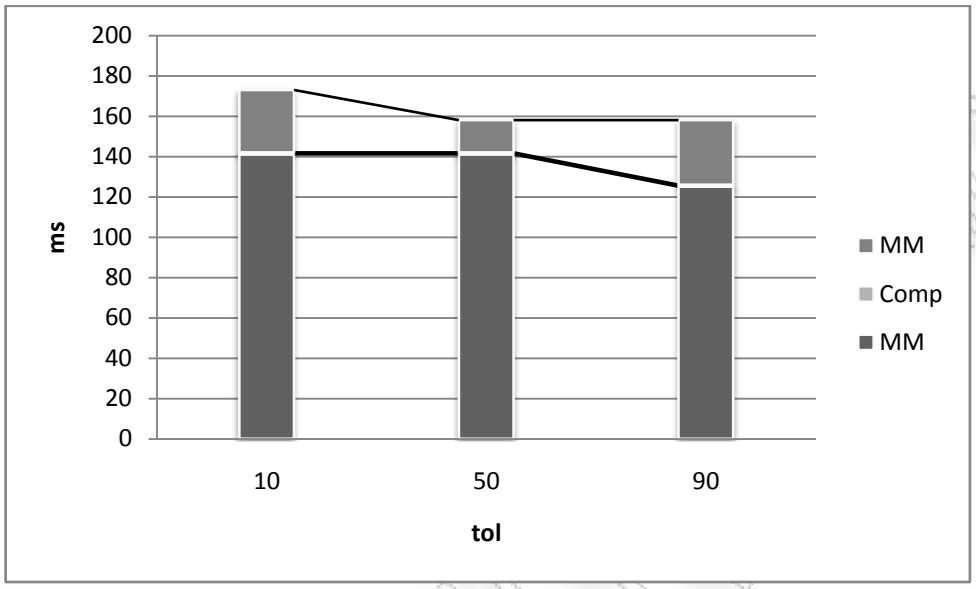
(γ)

Εικόνα 24 - Συμπίεση για διαφορετικές τιμές (α) της παραμέτρου *look* (β) της παραμέτρου *tol* (γ) της παραμέτρου *max. speed div.*

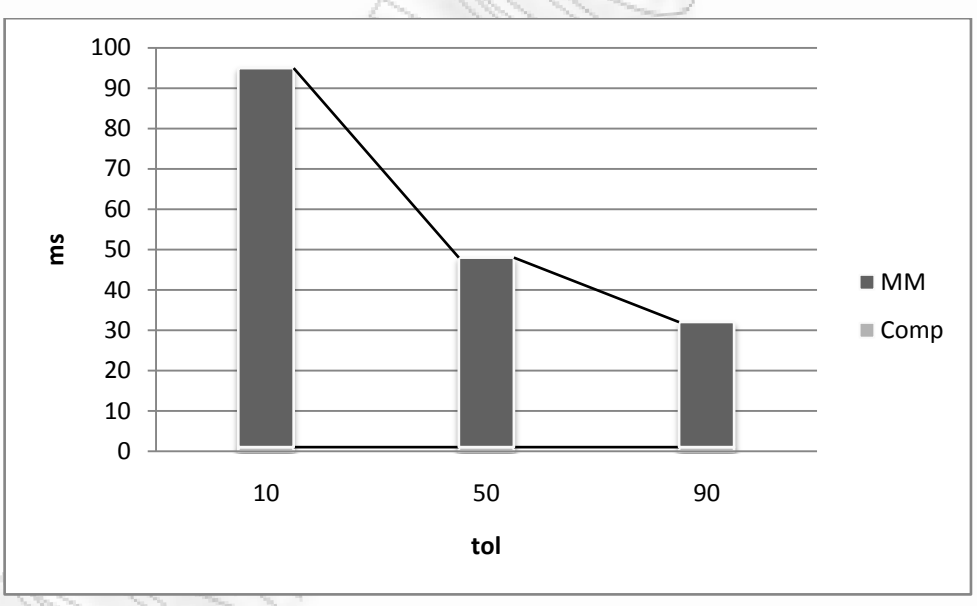
Στη Εικόνα 24(α) παρατηρούμε ότι η συμπίεση εξαρτάται από την τιμή των προς τα εμπρός ακμών προς έλεγχο, μόνο αν ακολουθείται από αλγόριθμο συμπίεσης των δεδομένων.

Στην Εικόνα 24(β) φαίνεται ότι μεγαλύτερη τιμή της παραμέτρου ανοχής δίνει αποτελέσματα με μεγαλύτερο βαθμό συμπίεσης.

Στην Εικόνα 24(γ), όπως ήταν αναμενόμενο, όσο περισσότερα σημεία έχει η αρχική τροχιά μας τόσο καλύτερη συμπίεση επιτυγχάνεται εν τέλει, αφού συγκρίνουμε αποτέλεσμα τροχιάς πάνω στο δίκτυο με τροχιά σημείων στο χώρο. Στην περίπτωση της τιμής 75 για *max. speed div.* έχουμε αρνητική συμπίεση, δηλαδή παράγουμε περισσότερα σημεία για να περιγράψουμε τη διαδρομή πάνω στο δίκτυο, πράγμα που δείχνει ότι τα σημεία της αρχικής τροχιάς ήταν πολύ αραιά.



(α)



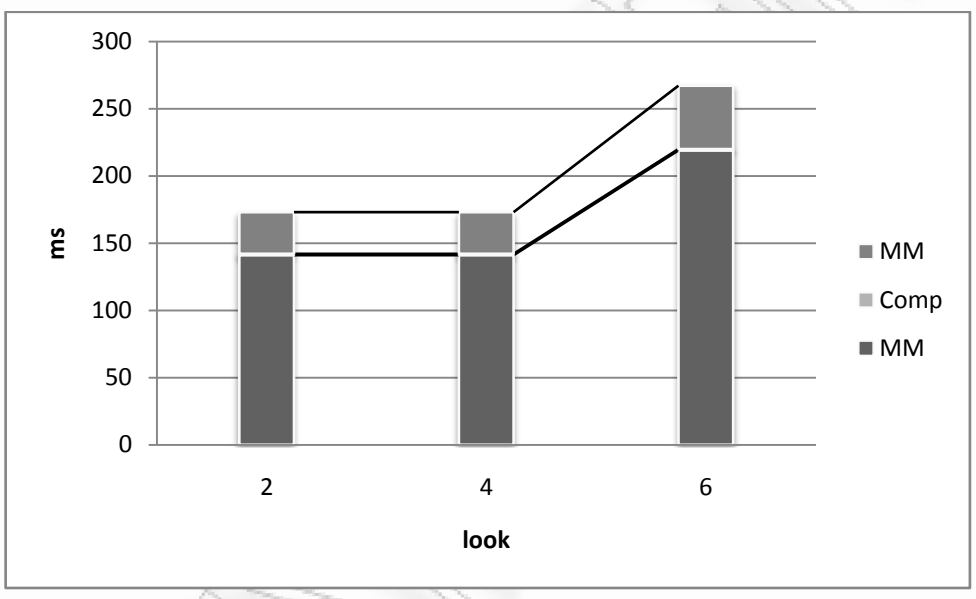
(β)

Εικόνα 25 - Χρόνος εκτέλεσης των μεθόδων (α) MM+Comp+MM και (β) Comp+MM σε σχέση με τις διαφορετικές τιμές της μεταβλητής *tol*

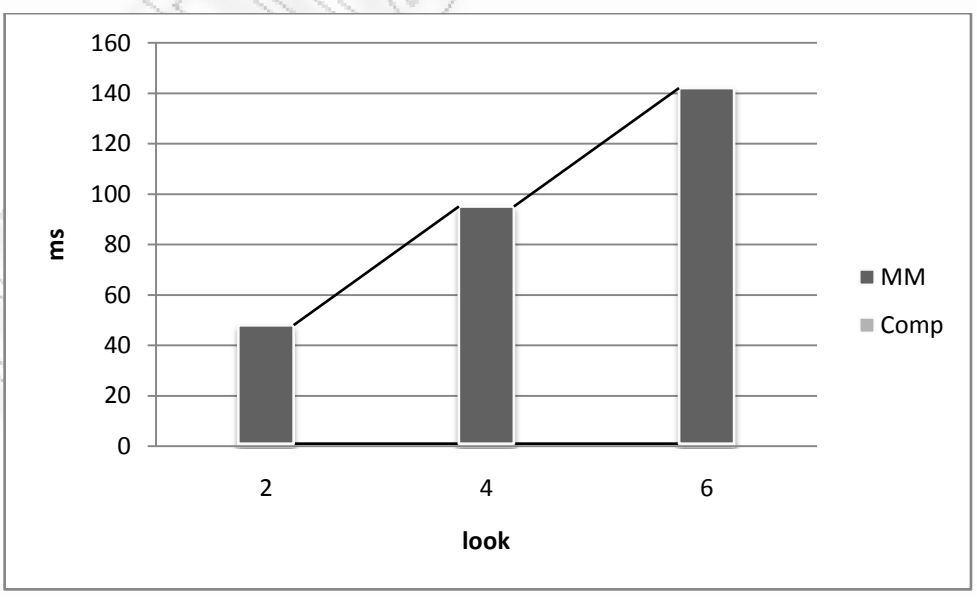
Στην Εικόνα 25(α) παρατηρούμε μικρή μείωση του χρόνου εκτέλεσης της μεθόδου MM+Comp+MM για μεγαλύτερες τιμές της μεταβλητής *tol*. Αυτό

συμβαίνει γιατί η μέθοδος της συμπίεσης δίνει λιγότερα σημεία στη μέθοδο του ταιριάσματος χάρτη τη δεύτερη φορά.

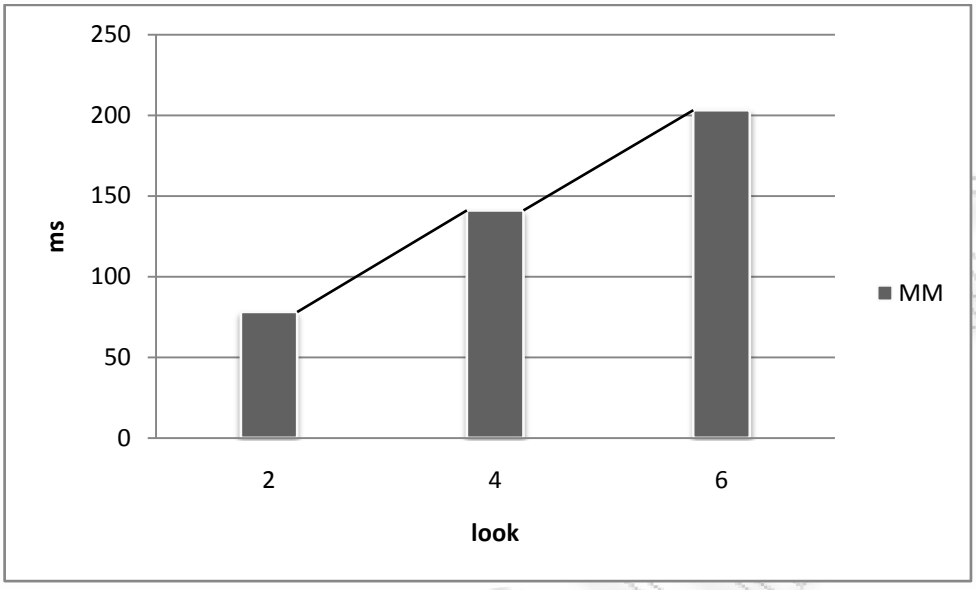
Στην Εικόνα 25(β) η βελτίωση του χρόνου εκτέλεσης για μεγαλύτερες τιμές της μεταβλητής *tol* φαίνεται καθαρότερα για τη μέθοδο Comp+MM, όπου δεν προηγείται ο αλγόριθμος ταιριάσματος χάρτη του αλγορίθμου συμπίεσης. Μεγαλύτερη τιμή *tol* δίνει λιγότερα σημεία, οπότε και μικρότερο χρόνο εκτέλεσης στη μέθοδο MM.



(α)



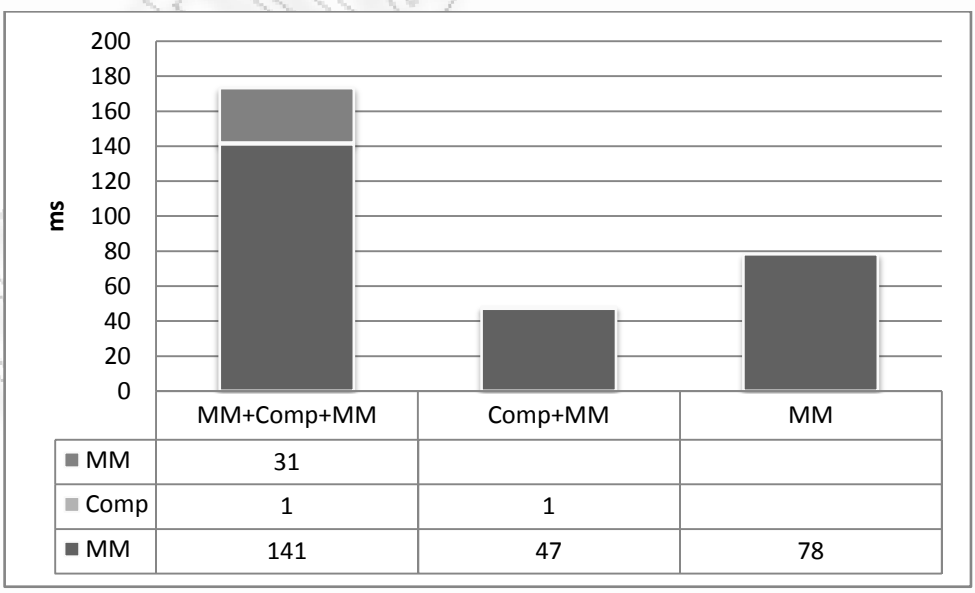
(β)



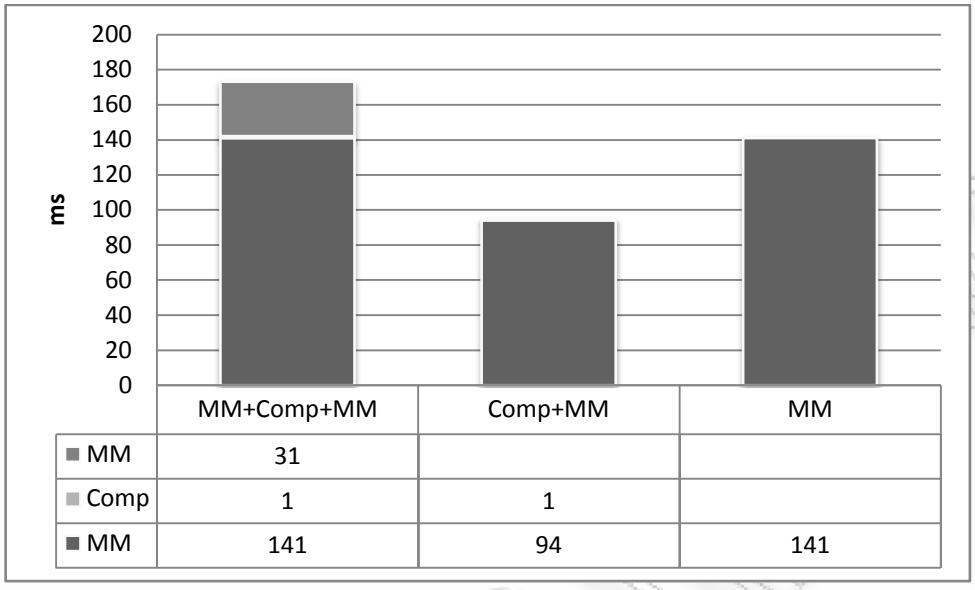
(γ)

Εικόνα 26 – Χρόνος εκτέλεσης των μεθόδων (α) MM+Comp+MM, (β) Comp+MM και (γ) MM για διάφορες τιμές της *look*

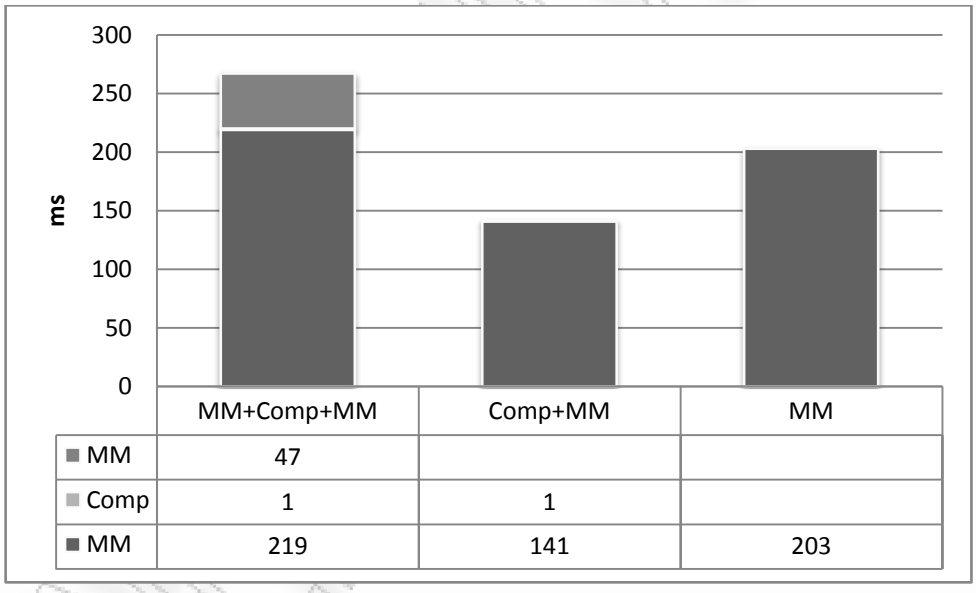
Από τα διαγράμματα στην Εικόνα 26 βλέπουμε ότι μεγαλύτερος αριθμός προς τα εμπρός ελέγχου ακμών, μεγαλώνει το χρόνο εκτέλεσης. Τα δύο αυτά μεγέθη φαίνονται να είναι ανάλογα μεταξύ τους.



(α)



(β)

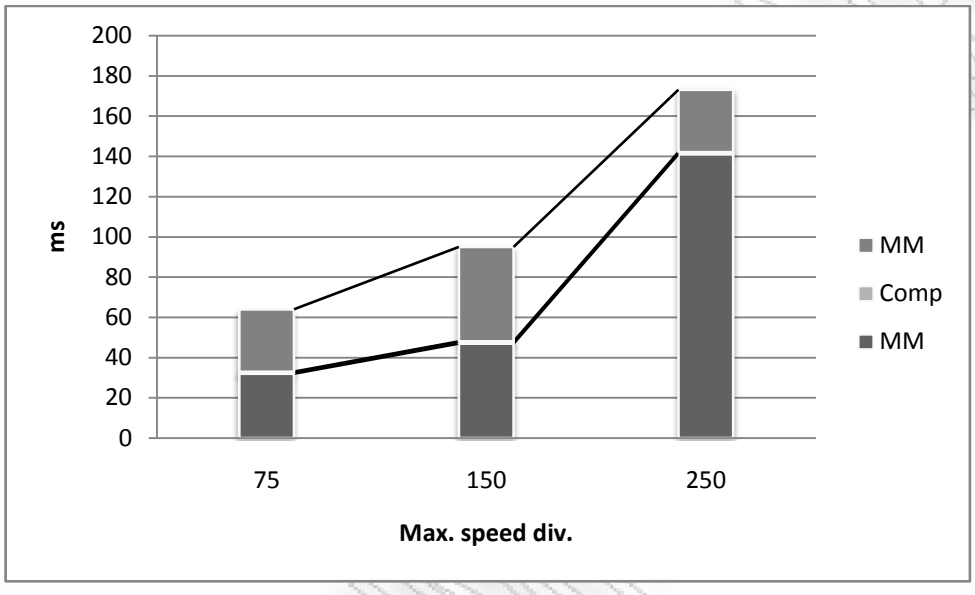


(γ)

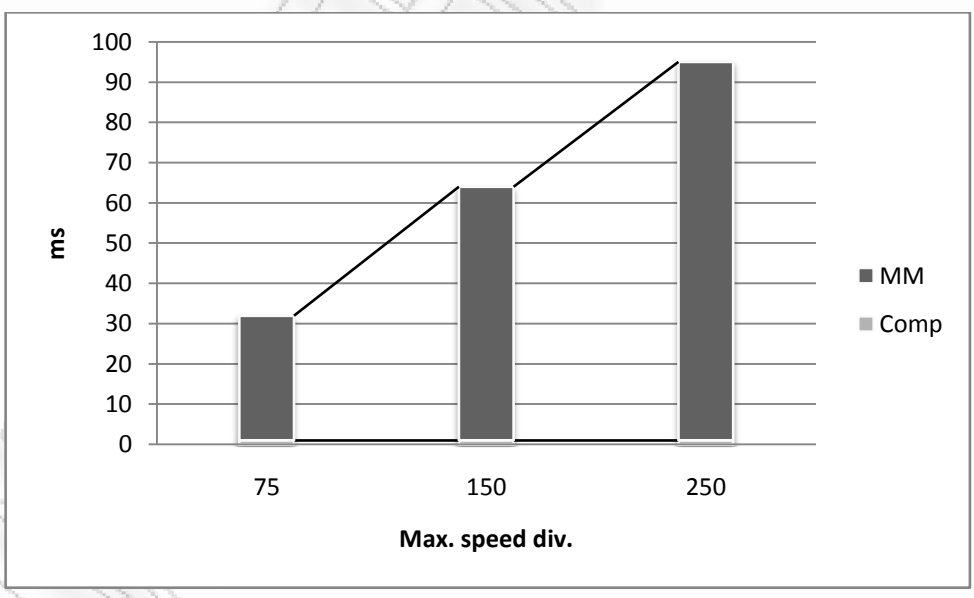
Εικόνα 27 - Χρόνος εκτέλεσης των αλγορίθμων για τιμή (α) *look*=2, (β) *look*=4 και (γ) *look*=6

Από τα τρία διαγράμματα στην Εικόνα 27 παρατηρούμε ότι η μέθοδος Comp+MM δίνει τα καλύτερα αποτελέσματα. Αυτό εξηγείται από το γεγονός ότι ο αλγόριθμος του ταιριάσματος χάρτη εφαρμόζεται μετά από τη συμπίεση, με αποτέλεσμα να εκτελείται για λιγότερα σημεία. Αυτό εξηγεί και το μικρότερο

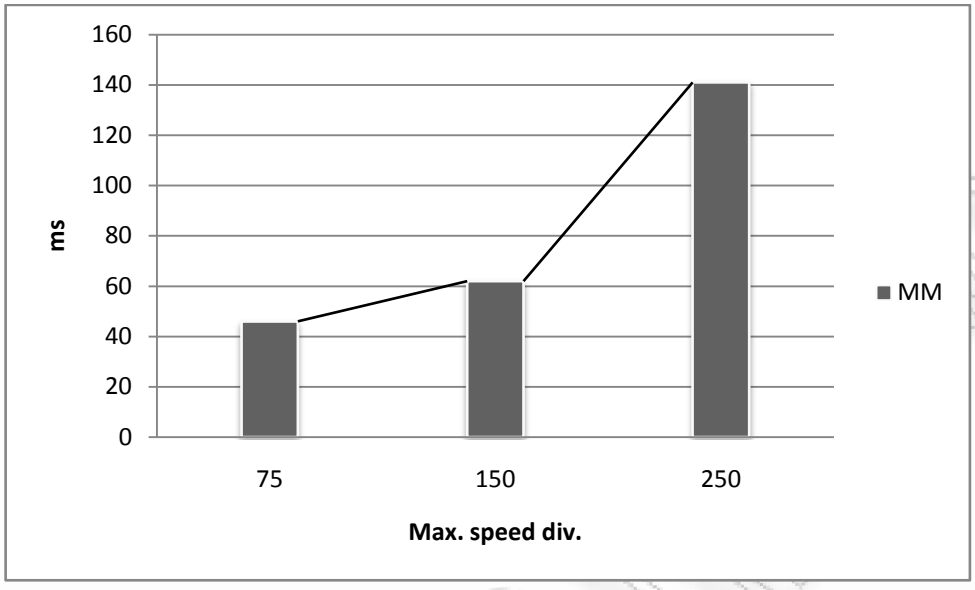
χρόνο που χρειάστηκε για να εκτελεστεί για δεύτερη φορά το ταίριασμα χάρτη στη μέθοδο MM+Comp+MM.



(α)



(β)



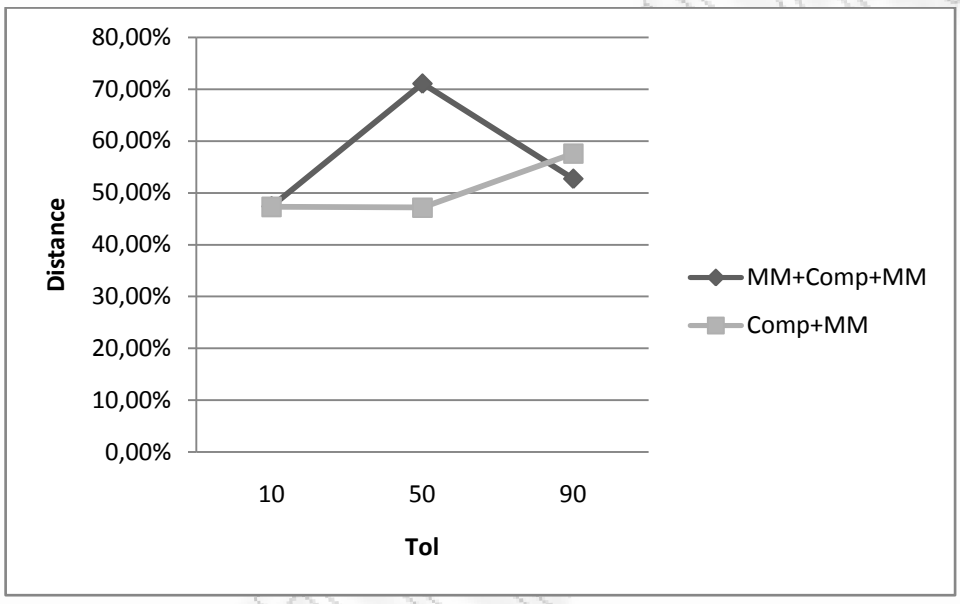
(γ)

Εικόνα 28 - Χρόνος εκτέλεσης των μεθόδων (α) MM+Comp+MM, (β) Comp+MM και (γ) MM για διαφορετικές τιμές της *Max. speed div.*

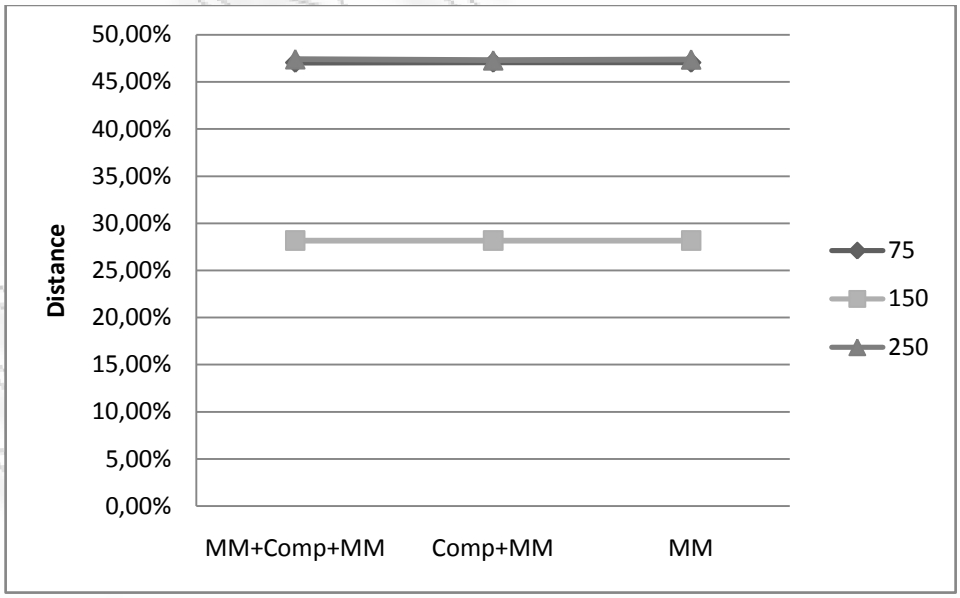
Στα διαγράμματα στην Εικόνα 28 παρατηρούμε ότι με μικρές τιμές της μεταβλητής *max. speed div.* κερδίζουμε σε χρόνο εκτέλεσης σε όλες τις μεθόδους. Αυτό σημαίνει ότι όσο λιγότερα σημεία έχουμε στην τροχιά μας, τόσο καλύτερους χρόνους πετυχαίνουμε, ένα αποτέλεσμα απολύτως λογικό.

4.6.2 Αποτελέσματα για online δεδομένα

Για τα online δεδομένα χρησιμοποιούμε τον αλγόριθμο του ταιριάσματος χάρτη για *look*=0 και τον αλγόριθμο OW. Ακολουθούν οι γραφικές απεικονίσεις των αποτελεσμάτων.



(α)

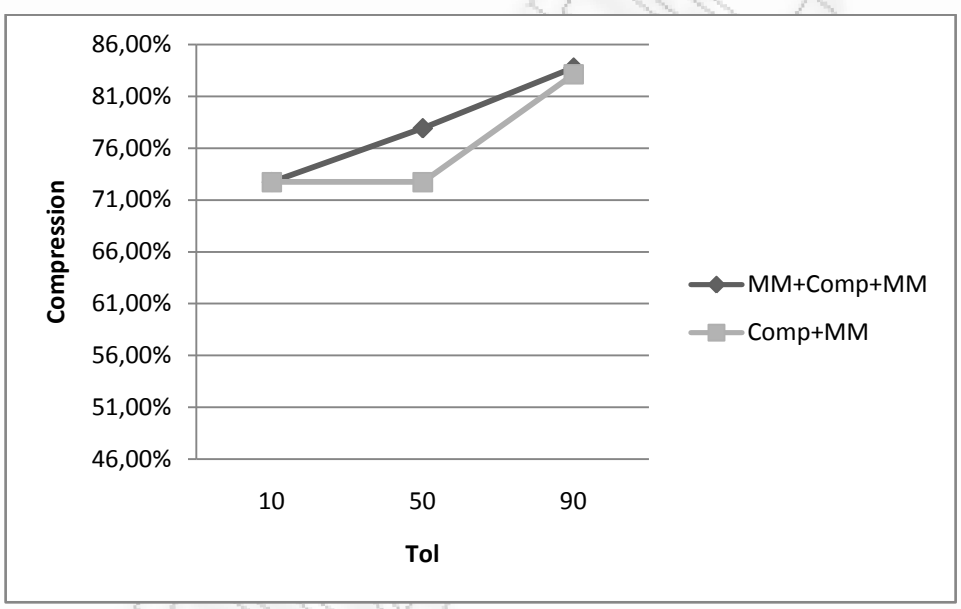


(β)

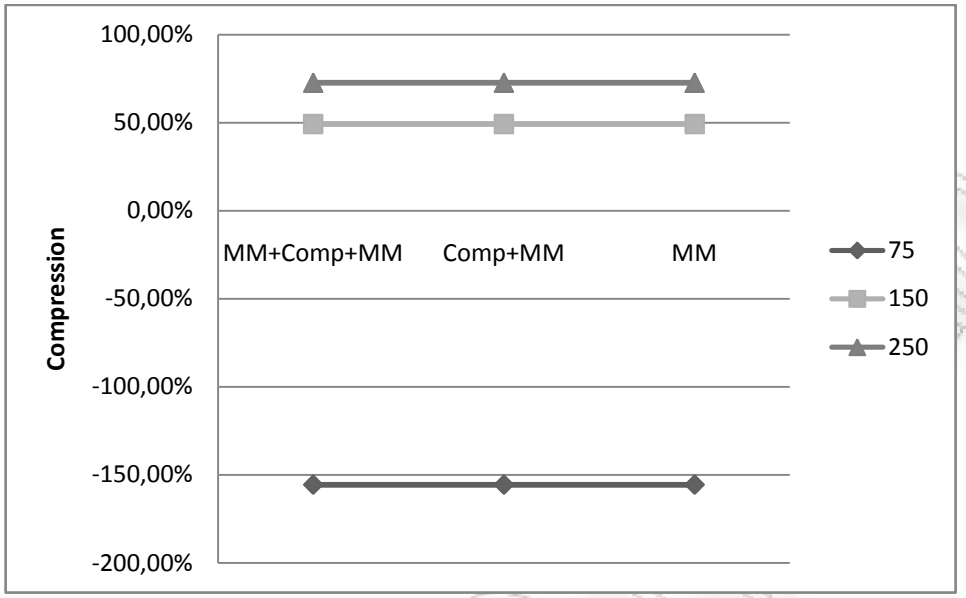
Εικόνα 29 - Απόσταση για διαφορετικές τιμές των μεταβλητών (α) *tol* και (β) *max speed div.*

Στην Εικόνα 29(α) παρατηρούμε ότι για μεγαλύτερες τιμές ανοχής τα αποτελέσματα είναι ελαφρώς χαμηλότερης ποιότητας. Στην Εικόνα 29(β) βλέπουμε ότι, ανεξάρτητα από τη μέθοδο, μικρός ή πολύ μεγάλος αριθμός σημείων δίνει χειρότερα αποτελέσματα όσον αφορά στην ποιότητα.

Συγκριτικά με τα offline δεδομένα (Εικόνα 23), παρατηρούμε ότι ο προς τα εμπρός έλεγχος βοηθάει στην επίτευξη αποτελεσμάτων καλύτερης ποιότητας για δεδομένα με μικρό ή πολύ μεγάλο αριθμό σημείων.



(α)



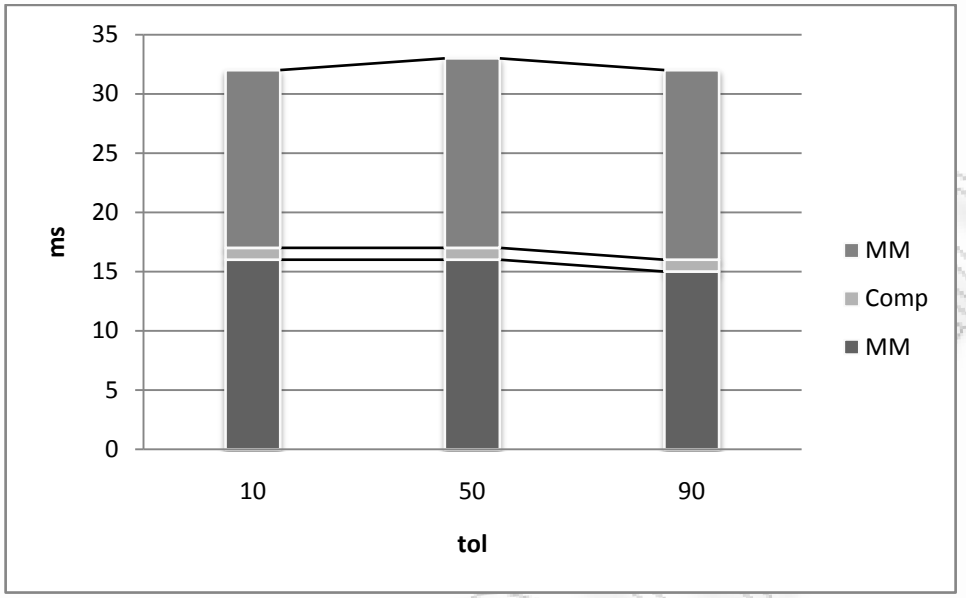
(β)

Εικόνα 30 - Συμπίεση για διαφορετικές τιμές των μεταβλητών (α) *tol* και (β) *max speed div.*

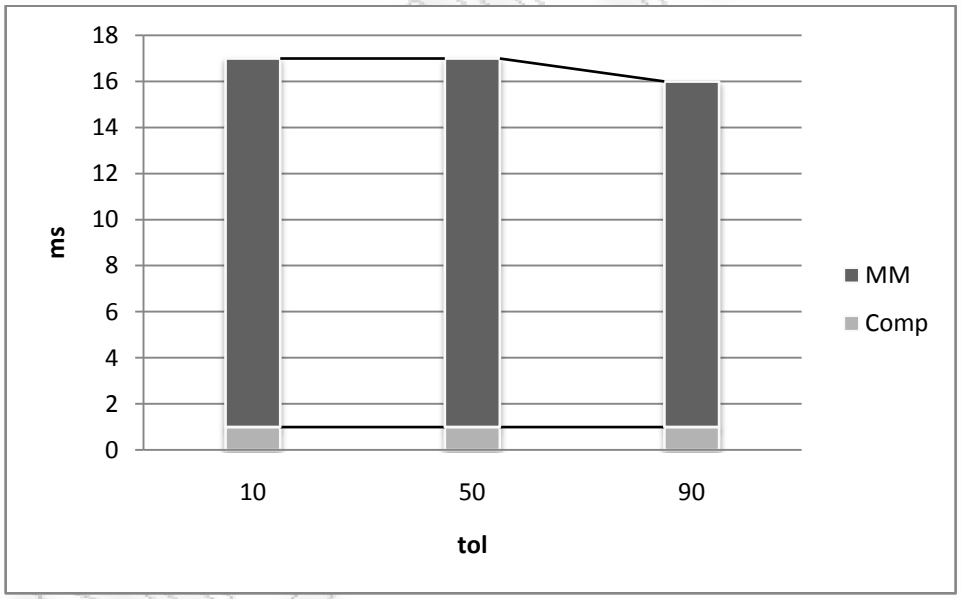
Στην Εικόνα 30(α) φαίνεται ότι μεγαλύτερη τιμή ανοχής προσφέρει μεγαλύτερο βαθμό συμπίεσης, αφού μετά από μεγαλύτερη συμπίεση χάνονται λεπτομέρειες της διαδρομής.

Στην Εικόνα 30(β) παρατηρούμε ότι όσα περισσότερα σημεία έχει η αρχική τροχιά μας τόσο καλύτερη συμπίεση επιτυγχάνεται, αφού συγκρίνουμε αποτέλεσμα τροχιάς πάνω στο δίκτυο με τροχιά σημείων στο χώρο. Στην περίπτωση που τα σημεία της τροχιάς έχουν μεγάλη απόσταση μεταξύ τους, έχουμε αρνητική συμπίεση.

Σε σχέση με τα offline δεδομένα (Εικόνα 24) τα αποτελέσματα είναι ανάλογα τόσο για τις διαφορετικές τιμές ανοχής, όσο και για τον αριθμό των σημείων.



(α)



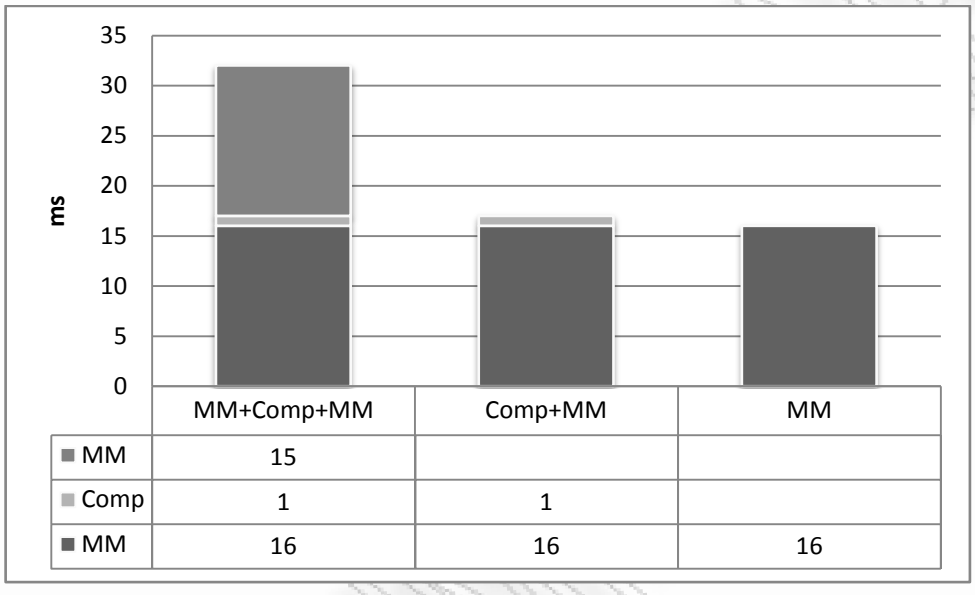
(β)

Εικόνα 31 - Χρόνος εκτέλεσης των μεθόδων (α) MM+Comp+MM και (β) Comp+MM για διαφορετικές τιμές της *tol*

Στην Εικόνα 31 παρατηρούμε ότι στην περίπτωση των online δεδομένων ο αριθμός των σημείων δεν επηρεάζει το χρόνο εκτέλεσης.

Συγκριτικά με τα offline δεδομένα (Εικόνα 25), ο χρόνος στα offline επηρεαζόταν από τις διαφορετικές τιμές της μεταβλητής *tol* σε αντίθεση με τα online δεδομένα.

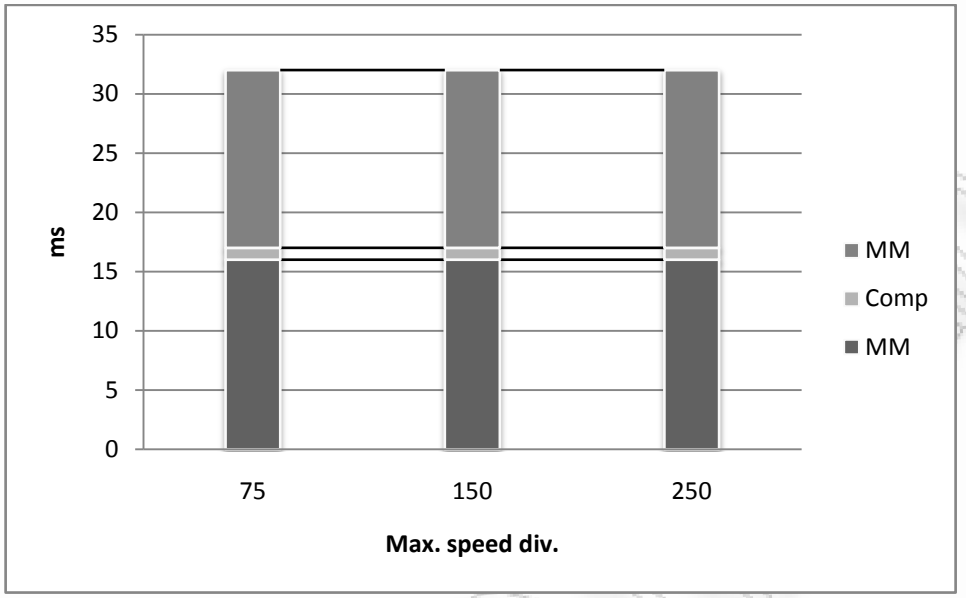
Προφανώς ο προς τα εμπρός έλεγχος ακμών είναι αυτός που καθυστερεί το αποτέλεσμα και εξαρτάται από τον αριθμό των σημείων.



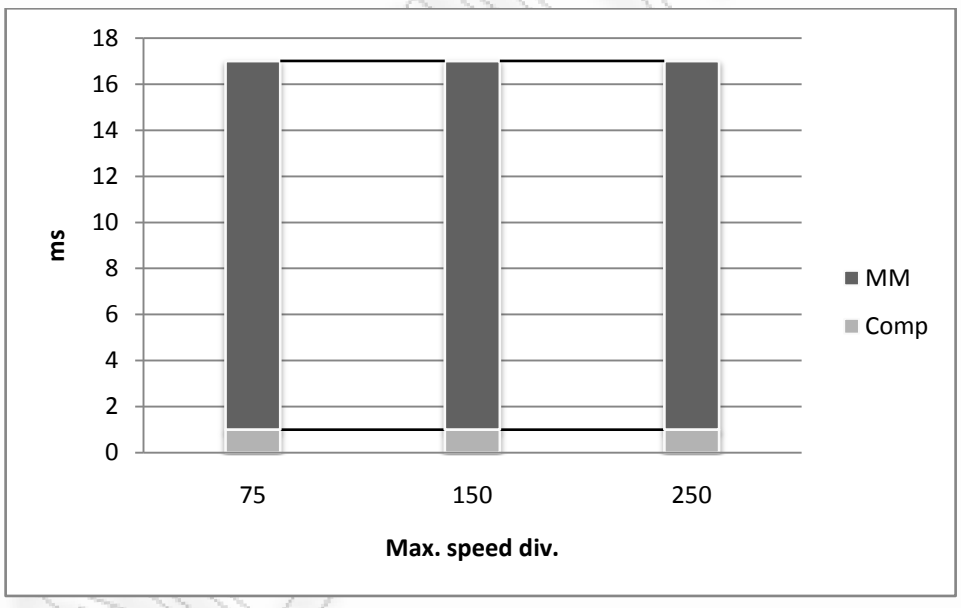
Εικόνα 32 - Χρόνος εκτέλεσης της κάθε μεθόδου

Στην Εικόνα 32 φαίνεται ο καλύτερος χρόνος εκτέλεσης των μεθόδων Comp+MM και MM σε σχέση με τη μέθοδο MM+Comp+MM, λόγω της εκτέλεσης της μεθόδου MM μόνο μία φορά από τις δύο πρώτες. Η Comp+MM δίνει τα ίδια αποτελέσματα με τη MM.

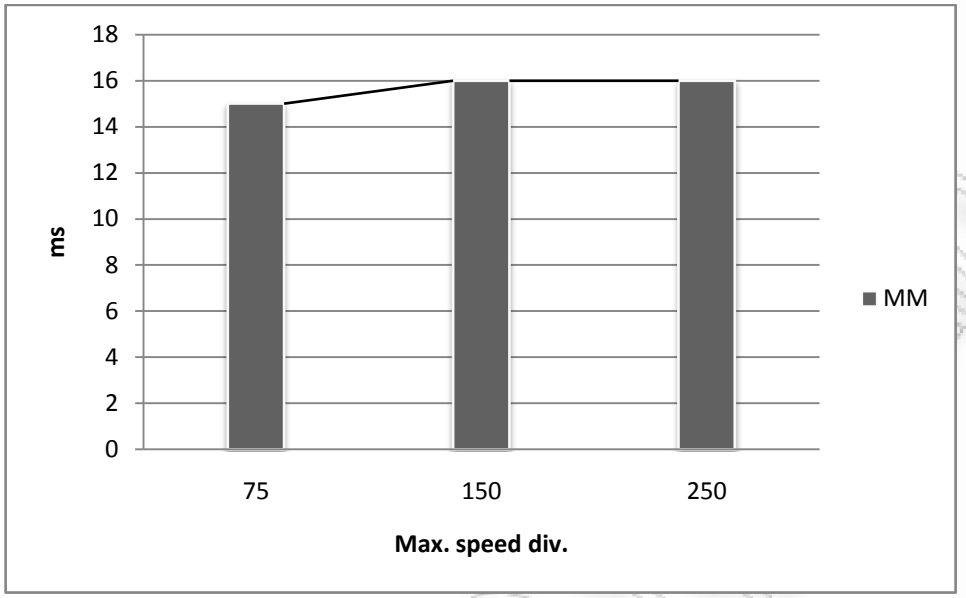
Συγκριτικά με τα offline δεδομένα (Εικόνα 27), η έλλειψη προς τα εμπρός ελέγχου καθιστά το χρόνο εκτέλεσης του Map-Matching ανεξάρτητο από τον αριθμό των σημείων. Έτσι η Comp+MM δίνει τον ίδιο χρόνο με τη MM στα online, ενώ στα offline η Comp+MM έδινε καλύτερους χρόνους από τη MM.


35
30
25
20
15
10
5
0
ms
75
150
250
Max. speed div.
MM
Comp
MM

(α)


18
16
14
12
10
8
6
4
2
0
ms
75
150
250
Max. speed div.
MM
Comp

(β)



(γ)

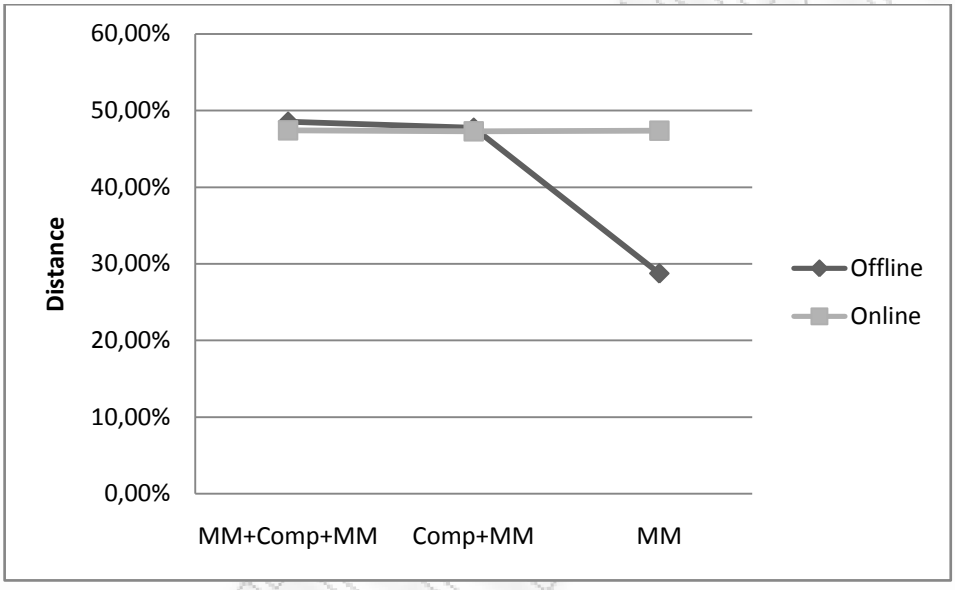
Εικόνα 33 - Χρόνος εκτέλεσης των μεθόδων (α) MM+Comp+MM και (β) Comp+MM (γ) MM για διαφορετικές τιμές της *max. speed div.*

Στην Εικόνα 33 παρατηρούμε ότι όσον αφορά στο χρόνο εκτέλεσης σε σχέση με τον αριθμό των σημείων, στα online δεδομένα δεν υπάρχει κάποια σχέση. Ο χρόνος είναι ο ίδιος σε κάθε περίπτωση.

Αντίθετα, στα offline δεδομένα (Εικόνα 28), ο αριθμός των σημείων επηρεάζει το χρόνο εκτέλεσης των αλγορίθμων MM+Comp+MM και MM.

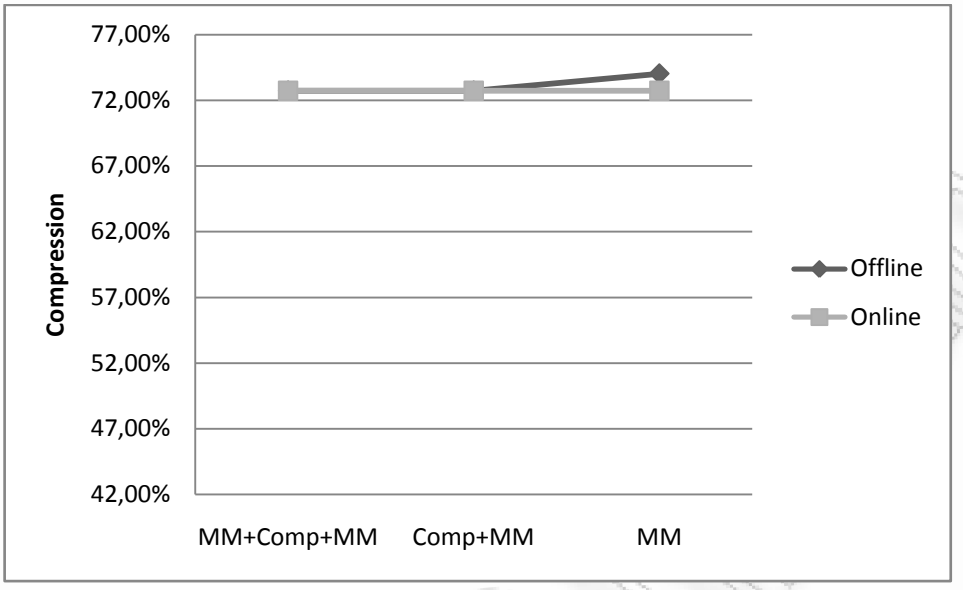
4.6.3 Σύγκριση μεταξύ online και offline υλοποιήσεων

Ακολουθούν γραφικές παραστάσεις για περαιτέρω σύγκριση της εφαρμογής των αντίστοιχων μεθόδων σε offline και online δεδομένα.



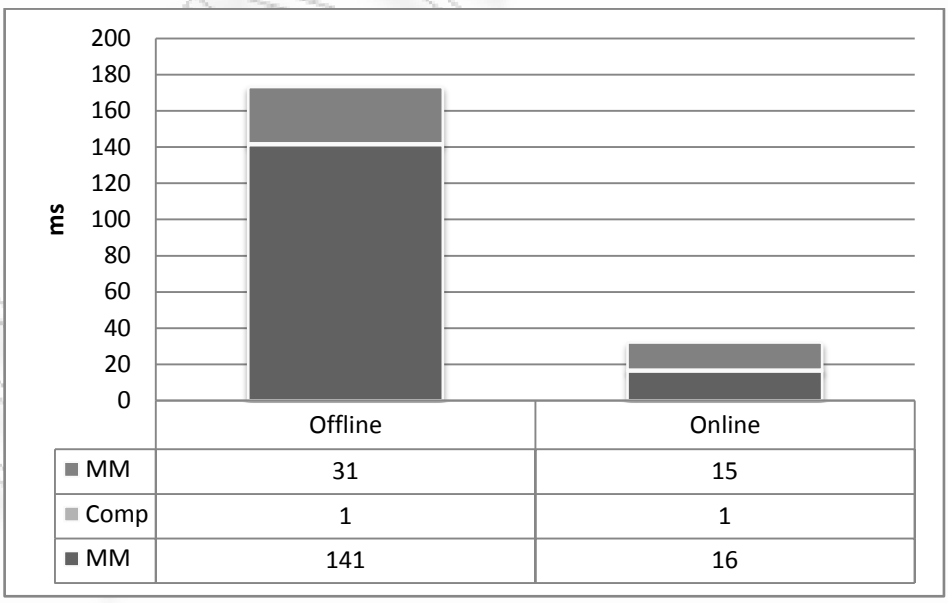
Εικόνα 34 - Απόσταση online και offline δεδομένων για κάθε μέθοδο

Στην Εικόνα 34 παρατηρούμε ότι τα αποτελέσματα των μεθόδων MM+Comp+MM και Comp+MM για offline δεδομένα διαφέρουν ελάχιστα με τις αντίστοιχες μεθόδους για online δεδομένα. Αυτό οφείλεται στο μικρό αριθμό σημείων για τον αλγόριθμο Map-Matching μετά από την εκτέλεση της συμπίεσης. Με λίγα σημεία ο αλγόριθμος παράγει φτωχά αποτελέσματα ανεξάρτητα από τον προς τα εμπρός έλεγχο ακμών. Καλύτερα αποτελέσματα δίνει η μέθοδος MM, μιας και στα offline δεδομένα ελέγχει 4 ακμές μπροστά ενώ για τα offline καμία.

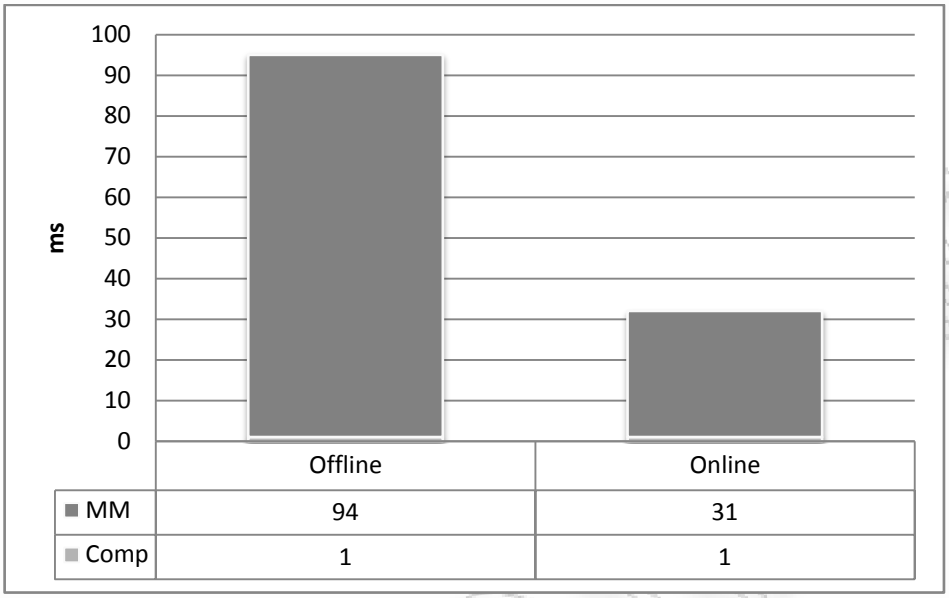


Εικόνα 35 - Συμπίεση online και offline δεδομένων για κάθε μέθοδο

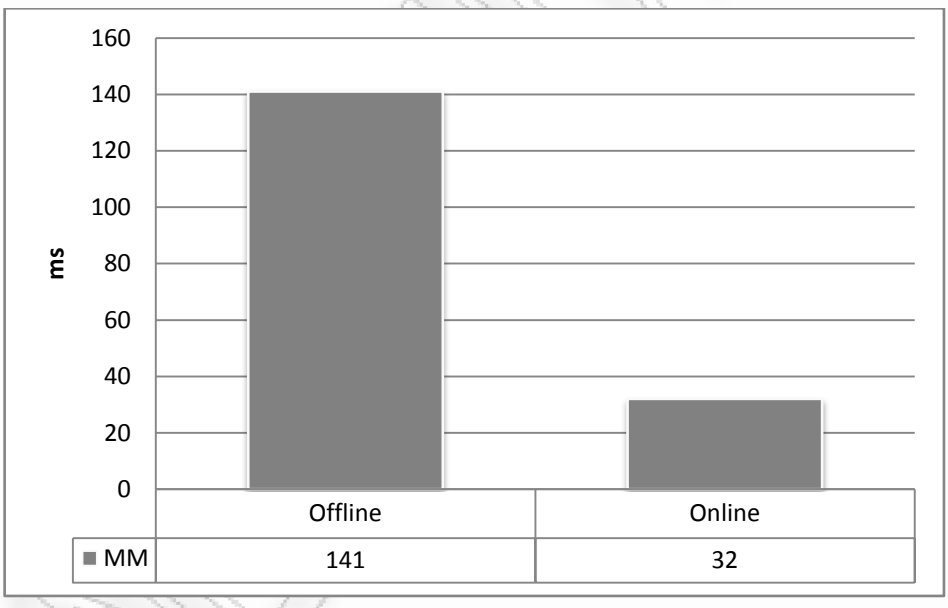
Στην Εικόνα 35 φαίνεται ότι όσον αφορά στη συμπίεση, οι διαφορές είναι πολύ μικρές για τις δύο κατηγορίες, μιας και δεν εξαρτάται από την ποιότητα του αλγορίθμου ταιριάσματος χάρτη.



(α)



(β)



(γ)

Εικόνα 36 - Χρόνος εκτέλεσης των μεθόδων (α) MM+Comp+MM, (β) Comp+MM και (γ) MM για offline και online δεδομένα

Στην Εικόνα 36 φαίνεται ότι η διαφορά μεταξύ του χρόνου εκτέλεσης των μεθόδων για offline και online δεδομένα είναι αρκετά μεγάλη, φτάνοντας μέχρι και τις 4 φορές μικρότερη υπέρ των online μεθόδων. Αυτό αποδεικνύει ότι η

εφαρμογή του προς τα εμπρός έλεγχου στο ταίριασμα χάρτη είναι αρκετά χρονοβόρα.

4.7 Σύνοψη της αξιολόγησης

Από την ανάλυση των δεδομένων, όσον αφορά στην αποτελεσματικότητα της κάθε μεθόδου, προκύπτει ότι η μέθοδος Comp+MM προσφέρει παρόμοια αποτελέσματα με τη μέθοδο MM+Comp+MM όσον αφορά στην ποιότητα και την συμπίεση. Η MM+Comp+MM, όμως, είναι τουλάχιστον 2 φορές πιο αργή από τις υπόλοιπες. Η μέθοδος Comp+MM προσφέρει πολύ καλύτερο χρόνο από τη MM, λόγω του μικρότερου αριθμού σημείων στα οποία εκτελείται η πράξη του ταιριάσματος χάρτη. Παρόλα αυτά, μαζί με την μέθοδο MM+Comp+MM προσφέρουν αποτελέσματα λιγότερο αξιόλογα απ' ότι η μέθοδος MM όσον αφορά στην ποιότητα και στην συμπίεση offline δεδομένων. Στη online δεδομένα προσφέρουν παρόμοια αποτελέσματα.

Ο λόγος που η Comp+MM προσφέρει μεγαλύτερη ταχύτητα είναι ότι η πράξη της συμπίεσης, συγκριτικά με αυτήν του ταιριάσματος χάρτη, δεν επιβαρύνει καθόλου το χρονικό αποτέλεσμα. Η ταχύτητα εξαρτάται αποκλειστικά από τον αλγόριθμο ταιριάσματος χάρτη. Την ταχύτητά του καθορίζουν ο αριθμός των σημείων και των προς τα εμπρός ακμών που θα ελεγχθούν. Η τιμή της *look* φαίνεται να είναι ανάλογη του χρόνου που χρειάζεται για την εκτέλεση του αλγορίθμου. Εξετάζοντας τα αποτελέσματα για την online εφαρμογή των μεθόδων, βλέπουμε ότι ο αριθμός των σημείων δεν επηρεάζει ιδιαίτερα τον χρόνο εκτέλεσης αν δεν υπάρχει προς τα εμπρός έλεγχος ακμών.

Όσο αφορά στην πράξη συμπίεσης, μεγαλύτερη τιμή ανοχής προσφέρει καλύτερα χρονικά αποτελέσματα, επηρεάζοντας αρνητικά την ποιότητα και το βαθμό συμπίεσης, όπως ήταν αναμενόμενο. Επίσης παρατηρούμε ότι η τιμή της μεταβλητής *look* επηρεάζει το χρόνο εκτέλεσης και την ποιότητα, αλλά όχι τον βαθμό συμπίεσης. Στην σύγκριση μεταξύ των offline και online μεθόδων παρατηρούμε ότι η διαφορά της ύπαρξης προς τα εμπρός ελέγχου ή όχι,

παρατηρείται μόνο στην περίπτωση χρήσης της μεθόδου MM, χωρίς την προηγούμενη εκτέλεση συμπίεσης.

Τελικά, η Comp+MM σε σχέση με την MM προσφέρει καλύτερους χρόνους αν χρησιμοποιούμε προς τα εμπρός έλεγχο ακμών και περίπου τους ίδιους αν όχι. Από άποψη ποιότητας, η μέθοδος Comp+MM δίνει χειρότερα αποτελέσματα σε σύγκριση με τη MM αν ελέγχουμε offline δεδομένα και περίπου τα ίδια στα online. Η ύπαρξη του προς τα εμπρός ελέγχου ακμών επηρεάζει την ποιότητα των αποτελεσμάτων μόνο στη μέθοδο MM. Όσον αφορά στη συμπίεση, όλες οι μέθοδοι έδωσαν παρόμοια αποτελέσματα.

Παρόλα αυτά, καμία μέθοδος δεν μπορεί να προσφέρει μεγάλο βαθμό συμπίεσης, μιας και η μέθοδος ταιριάσματος χάρτη είναι σχεδιασμένη για να βρίσκει την πιο απλή διαδρομή, η οποία δε σημαίνει πάντα ότι είναι και η πιο σύντομη ή με το μικρότερο αριθμό ακμών. Μπορεί να εξαλείψει λεπτομέρειες της διαδρομής μας, αλλά όχι να μας δώσει το αποτέλεσμα με τα λιγότερα σημεία.

ΚΕΦΑΛΑΙΟ 5

Σκιαγράφηση νέας μεθόδου συμπίεσης σε δίκτυο

5.1 Εισαγωγή

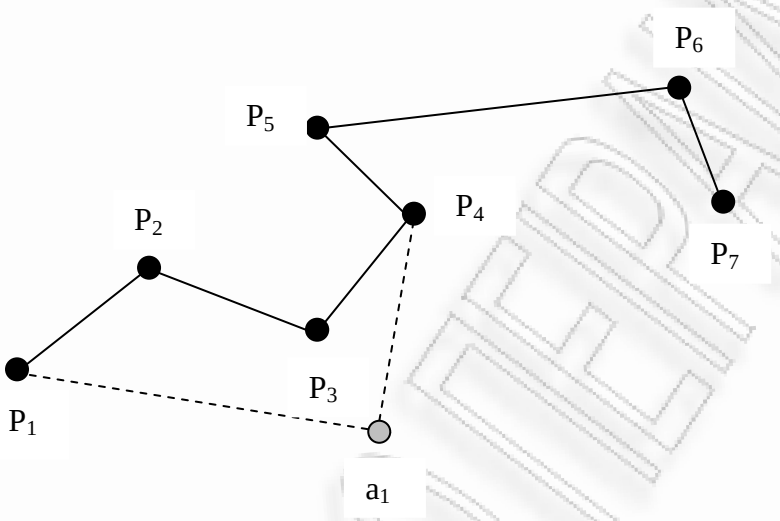
Στόχος μας είναι η επέκταση των εναλλακτικών μεθόδων του βήματος 3 με την εύρεση αλγορίθμου που θα μπορεί να συμπιέζει την τροχιά μας πάνω στο δίκτυο. Αυτό μπορεί να επιτευχθεί αν, μετά την εκτέλεση αλγορίθμου ταιριάσματος χάρτη, αλλάξουμε κάποια σημεία της πορείας, ώστε να χρειάζονται λιγότεροι κόμβοι για την περιγραφή της διαδρομής του αντικειμένου. Ένα τέτοιος αλγόριθμος θα λειτουργούσε για offline δεδομένα και θα μπορούσε να υλοποιηθεί με τη χρήση ενός αλγορίθμου shortest path, σε συγκεκριμένα σημεία της τροχιάς μας.

Θα μπορούσαμε να ορίσουμε το μέγιστο βαθμό συμπίεσης που μπορεί να επιτευχθεί. Θεωρούμε ότι το πρώτο και το τελευταίο σημείο της τροχιάς είναι πληροφορία που δεν πρέπει να χαθεί. Επίσης έχουμε τροχιά πάνω στο δίκτυο – γράφο, οπότε ως μέγιστη συμπίεση μπορούμε να ορίσουμε το shortest path μεταξύ του πρώτου και του τελευταίου σημείου της τροχιάς μας. Εφαρμογή του αλγόριθμου του shortest path σε ενδιάμεσα σημεία θα μπορούσε να μας δώσει μικρότερο βαθμό συμπίεσης, διατηρώντας, όμως, περισσότερη πληροφορία.

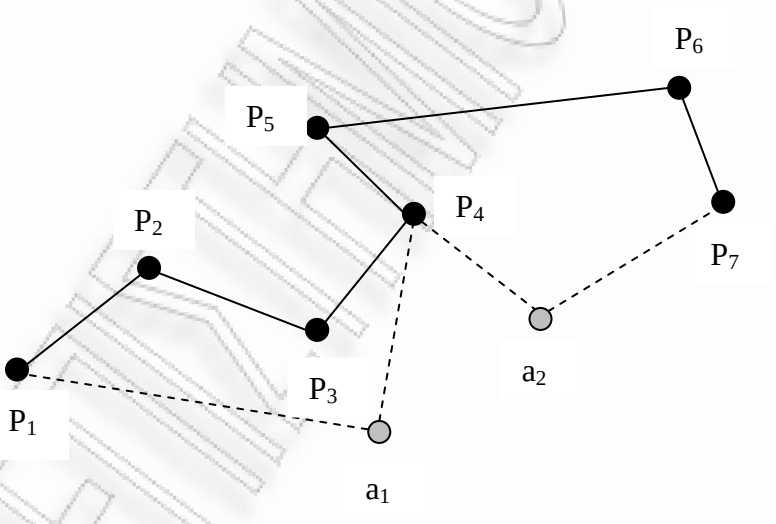
5.2 Περιγραφή αλγορίθμου

Για να εξηγήσουμε την ιδέα, δίνουμε ένα παράδειγμα στην Εικόνα 37. Έστω ότι έχουμε μία τροχιά $\{P_1, P_2, P_3, P_4, P_5, P_6, P_7\}$ και έστω ότι η μέγιστη συμπίεση που μπορούμε να πετύχουμε είναι το shortest path από το P_1 στο P_7, το οποίο είναι η διαδρομή $\{P_1, a_3, P_7\}$. Θα εξετάσουμε το πρώτο σημείο της τροχιάς μας με τα επόμενά του, μέχρι να βρεθεί σημείο όπου το shortest path από αυτό να

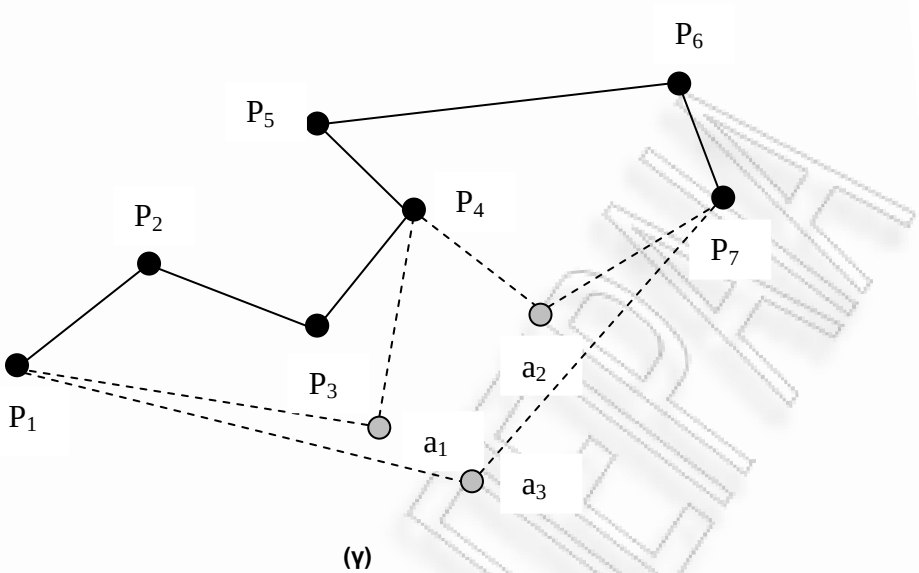
δίνει καλύτερο αποτέλεσμα από την ήδη υπάρχουσα διαδρομή, όπως φαίνεται στην Εικόνα 37(α).



(α)



(β)



(γ)

Εικόνα 37 - Παράδειγμα εκτέλεσης αλγορίθμου (α) πρώτο πέρασμα, (β) συνέχεια πρώτου περάσματος, (γ) δεύτερο πέρασμα

Από το σημείο P_1 στο P_2 προφανώς δεν υπάρχει διαδρομή που μπορεί να περιγραφεί με λιγότερα σημεία (είναι γειτονικά σημεία). Από το P_1 στο P_3 θα υπήρχε μόνο αν είχαμε απευθείας σύνδεση του P_1 με το P_3. Έστω ότι δεν υπάρχει κάτι τέτοιο και τελικά μπορούμε να μεταβούμε από το P_1 στο P_4 μέσω του a_1. Τότε η διαδρομή μας από $\{P_1, P_2, P_3, P_4\}$ θα γινόταν $\{P_1, a_1, P_4\}$, δηλαδή κατά ένα σημείο μικρότερη. Για το χρόνο στην κορυφή a_1 θα μπορούσαμε να χρησιμοποιήσουμε το χρόνο των κορυφών P_1 και P_4 και να θεωρήσουμε ότι το αντικείμενο κινείται με σταθερή ταχύτητα, δηλαδή την τεχνική που ακλουθήσαμε στην Ενότητα 3 (αλγόριθμος NodePasses). Ο αλγόριθμος, αφού βρει μικρότερη διαδρομή από το πρώτο σημείο μέχρι κάποιο άλλο, θα συνεχίζει θεωρώντας πρώτο σημείο το τελευταίο που είχε ανακαλύψει, στην περίπτωσή μας το σημείο P_4, και ελέγχοντας για μικρότερες διαδρομές από αυτό και μπροστά. Έστω ότι βρίσκει το σημείο a_2 ως καλύτερο για να περιγραφεί η διαδρομή από το P_4 στο P_7, όπως φαίνεται στην εικόνα Εικόνα 37(β) Επειδή το P_7 είναι το τελευταίο της διαδρομής μας θεωρούμε ότι το πρώτο πέρασμα του αλγορίθμου μας τελείωσε και το τελικό αποτέλεσμα είναι τα σημεία $\{P_1, a_1, P_4, a_2, P_7\}$, που είναι κατά δύο σημεία μικρότερο από την αρχική διαδρομή μας.

Αν χρειαζόμαστε μεγαλύτερο βαθμό συμπίεσης, εκτελούμε ξανά τον αλγόριθμο χωρίς να ελέγχουμε τα σημεία που προστέθηκαν στο προηγούμενο πέρασμα, δηλαδή τα σημεία P_1, P_4 και P_7, όπως φαίνεται στην Εικόνα 37(γ) Από το P_1 στο P_4 δεν υπάρχει καλύτερη διαδρομή, όπως ανακαλύψαμε προηγουμένως. Οπότε ελέγχουμε το P_1 με το P_7 και βρίσκουμε ότι υπάρχει καλύτερη διαδρομή μέσω του a_3, το οποίο είναι και η μέγιστη συμπίεση που είχαμε υπολογίσει στην αρχή. Δεν υπάρχει διαδρομή να προσφέρει μεγαλύτερο βαθμό συμπίεσης και ο αλγόριθμος σταματάει.

Η πολυπλοκότητα του αλγορίθμου είναι μεγάλη γιατί χρειάζεται η εκτέλεση του αλγορίθμου shortest-path. Παρόλα αυτά, αρκεί να εκτελεστεί ο αλγόριθμος στην αρχή για όλα τα σημεία της αρχικής μας τροχιάς και να κρατήσουμε τα αποτελέσματα σε *n* πίνακες, όπου *n* είναι ο αριθμός των σημείων της τροχιάς μας. Έτσι θα έχουμε όλη την πληροφορία που χρειάζεται, μιας και ελέγχουμε μόνο σημεία της αρχικής τροχιάς και ο αλγόριθμος του shortest path υπολογίζει ούτως η άλλως όλες τις μικρότερες διαδρομές, από ένα σημείο προς όλα τα άλλα. Με αυτό τον τρόπο δεν εκτελείται παραπάνω από μία φορά ο αλγόριθμος shortest path για κάθε σημείο.

Αναλυτικά ο προτεινόμενος αλγόριθμος MapComp φαίνεται στην Εικόνα 38.

Έστω ο πίνακας *T* με τα σημεία της τροχιάς μας πάνω στο δίκτυο και *max* ο μέγιστος βαθμός συμπίεσης που θέλουμε να πετύχουμε:

1. Υπολόγισε για όλα τα σημεία της τροχιάς *T* τα shortest paths προς όλα σημεία
2. Θέσε βαθμό συμπίεσης *c*=0
3. Όσο ο βαθμός συμπίεσης *c*<max
    a. Θέσε *i*=1
    b. Πάρε το *i*-οστό σημείο της τροχιάς *T*
    c. Για κάθε επόμενο σημείο της αρχικής τροχιάς *T* και μέχρι να τελειώσουν τα σημεία της τροχιάς *T*
        i. Αύξησε το *i* κατά 1
        ii. Αν υπάρχει συντομότερη διαδρομή από το σημείο της τροχιάς μας που επιλέχθηκε μέχρι το *i*-οστό
            1. Υπολόγισε τους χρόνους στις καινούργιες κορυφές
            2. Αντικατέστησε την διαδρομή από το επιλεγμένο σημείο μέχρι το *i*-οστό με την καινούργια και υπολόγισε τον χρόνο στις νέες κορυφές
            3. Όρισε ως επιλεγμένο σημείο το *i*-οστό
    d. Υπολόγισε την συμπίεση *c* που επιτεύχθηκε

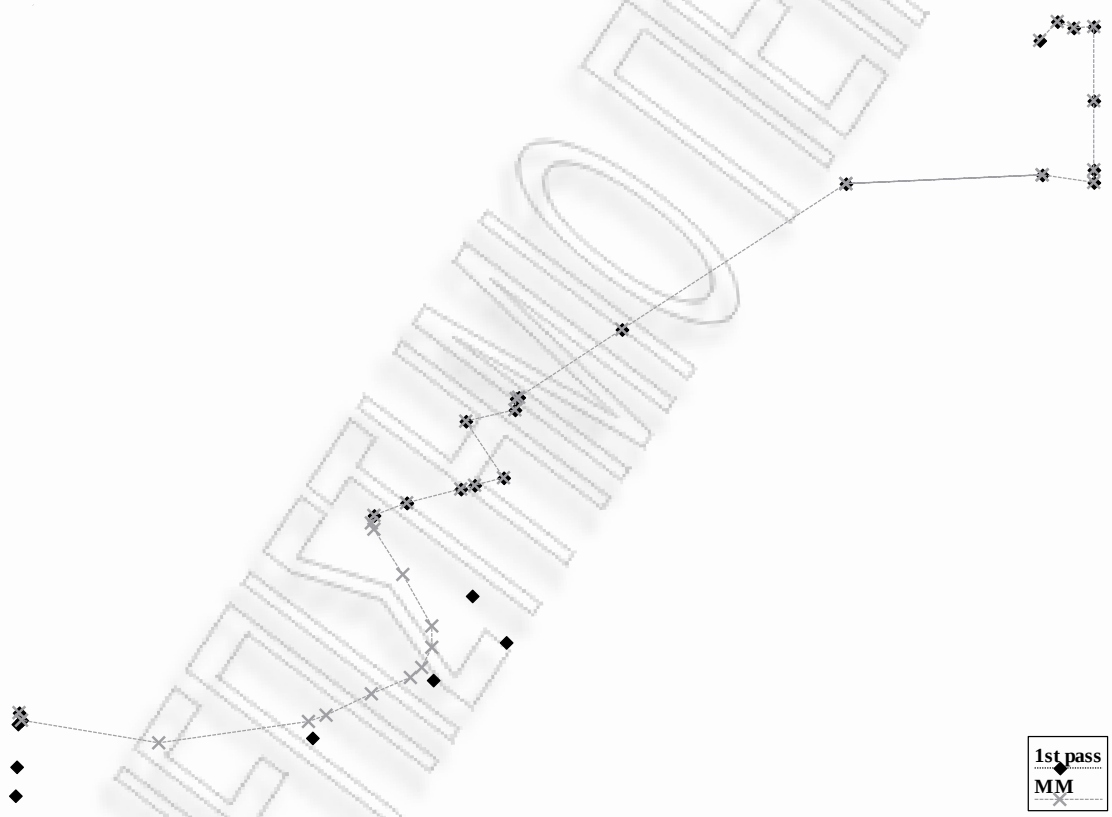
Εικόνα 38 - Προτεινόμενος αλγόριθμος συμπίεσης υπό περιορισμούς δικτύου

Ο αλγόριθμος λειτουργεί μόνο για offline δεδομένα, μιας και χρειάζεται, πριν την εκτέλεση του, τον υπολογισμό των shortest paths από τα σημεία της τροχιάς μας. Θα μπορούσαμε να εκτελούμε shortest path για κάθε νέο σημείο που θα ερχόταν σε μια online εκδοχή του, αλλά ο χρόνος πιθανότατα θα ήταν αρκετά μεγάλος για να προλάβει να εκτελεστεί ταίριασμα χάρτη και υπολογισμός shortest path, μέχρι να λάβουμε το επόμενο σημείο μας. Μια τέτοια υλοποίηση θα χρειαζόταν μικρή συχνότητα λήψης σημείων από τον GPS δέκτη μας, για να μπορεί να λειτουργεί online.

Στο Παράρτημα Β υπάρχει η υλοποίηση του αλγόριθμου σε γλώσσα Java.

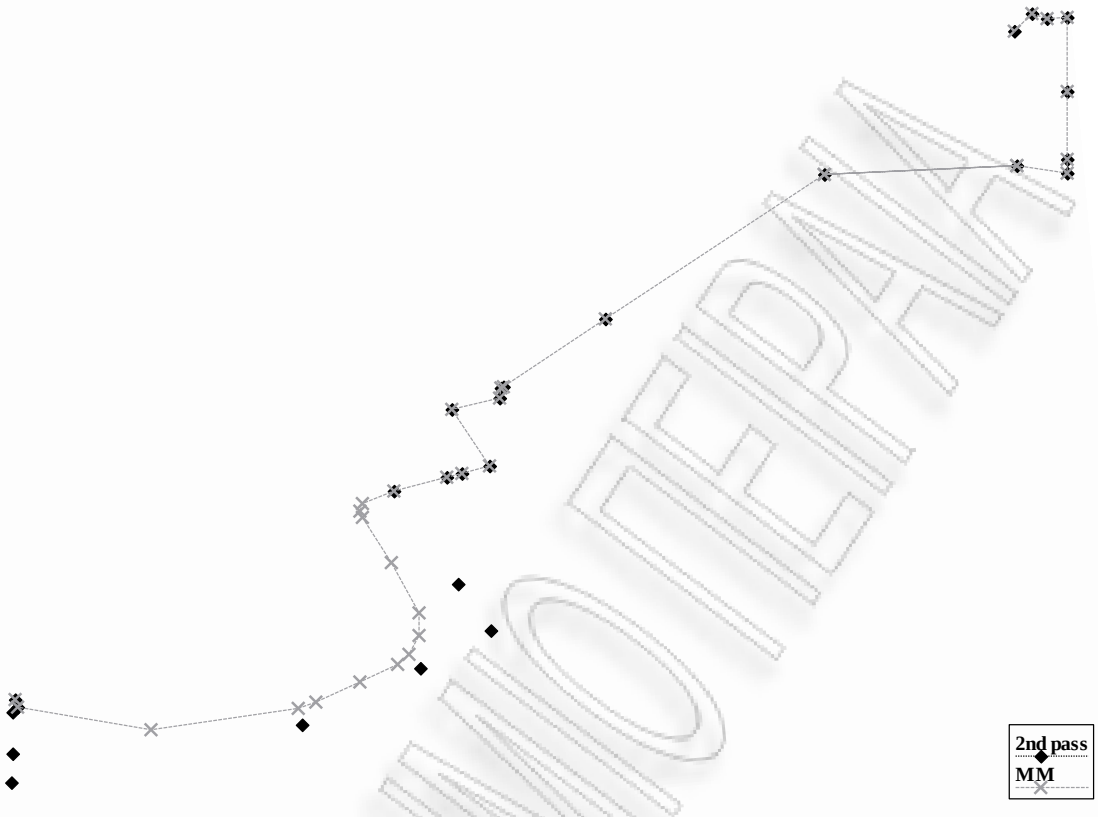
5.3 Πειραματικά αποτελέσματα – συμπεράσματα

Ο αλγόριθμος εφαρμόστηκε σε δεδομένα που παράχθηκαν από την γεννήτρια Brinkhoff με *max. speed div.* = 150. Για map-matching χρησιμοποιήθηκε ο αλγόριθμος [1] με τιμή της παραμέτρου *look*=4, ενώ για τον υπολογισμό των shortest paths χρησιμοποιήθηκε ο αλγόριθμος του Dijkstra[4]. Τα αποτελέσματα συγκρίθηκαν με την εφαρμογή του αλγορίθμου του Map-Matching.



Εικόνα 39 – Αποτέλεσμα πρώτου περάσματος του αλγορίθμου

Στην Εικόνα 39 βλέπουμε το αποτέλεσμα του πρώτου περάσματος του αλγορίθμου στα δεδομένα μας σε σχέση με την απλή εφαρμογή του Map-Matching. Παρατηρούμε ότι στην αρχή της διαδρομής ακολουθείται εντελώς διαφορετικός δρόμος, ο οποίος είναι μικρότερος σε σημεία από αυτόν που ακολουθήθηκε.



Εικόνα 40 - Αποτέλεσμα δεύτερου περάσματος του αλγορίθμου

Στην Εικόνα 40 βλέπουμε το αποτέλεσμα του δεύτερου περάσματος του αλγορίθμου στα δεδομένα μας σε σχέση με την απλή εφαρμογή του map-matching. Αν κοιτάξουμε προσεκτικά, βλέπουμε ότι στο δεύτερο πέρασμα ο αλγόριθμος έδωσε αποτέλεσμα με λιγότερα σημεία.

Όσον αφορά στα ποσοτικά αποτελέσματα, ο αλγόριθμος έδωσε για το πρώτο πέρασμα 14,7% καλύτερη συμπίεση απ' ότι ο απλός αλγόριθμος map-matching, αλλά η απόσταση ήταν 53%. Η συμπίεση σε σχέση με τα αρχικά δεδομένα ήταν 56,7%. Στο δεύτερο πέρασμα πήραμε συμπίεση που υπερτερούσε κατά 20,5% σε σχέση με τον αλγόριθμο map-matching και απόσταση 53,6%. Σε σχέση με τα αρχικά δεδομένα, η συμπίεση ήταν της τάξης του 59,7%.

Ο χρόνος που χρειάστηκε για να εκτελεστεί ο αλγόριθμος για 34 σημεία ήταν 12 δευτερόλεπτα. Αρκετά μεγάλος, αλλά χρειάστηκε να υπολογιστεί το shortest path 34 σημείων με τα υπόλοιπα 7443 που έχει ο χάρτης που χρησιμοποιήσαμε.

Παρόλα αυτά, καταφέραμε να έχουμε αρκετά καλή συμπίεση και δυνατότητα επιλογής ανάμεσα σε βαθμούς συμπίεσης, κάτι το οποίο δεν ήταν δυνατόν να επιτύχουμε με τις άλλες μεθόδους.

Όσον αφορά στον χρόνο που χρειάστηκε, ίσως να μπορούσαμε να τον μειώσουμε με την χρήση κάποιου εναλλακτικού αλγόριθμου shortest path αντί για του Dijkstra [4], ή μειώνοντας τα σημεία του χάρτη, κρατώντας μόνο αυτά που βρίσκονται σε μια προκαθορισμένη απόσταση *d* από τα σημεία της τροχιάς μας.

Για την εφαρμογή του αλγορίθμου σε online δεδομένα, με βάση τους χρόνους που καταγράφτηκαν, θα λέγαμε ότι με δειγματοληψία 0,4 sec, τιμή η οποία είναι λίγο μεγαλύτερη από την οριακή 12sec/34σημεία=0,35 sec., ο αλγόριθμος θα μπορούσε να εκτελεστεί. Ο χρόνος αυτός όμως εξαρτάται εκθετικά και από τα συνολικά σημεία του χάρτη μας. Η τιμή 0,4 sec θα ήταν ασφαλής για χάρτες με λιγότερα από 7443 σημεία. Φυσικά μεγάλο ρόλο παίζει και η υπολογιστική ισχύ του συστήματος στο οποίο εκτελούνται οι αλγόριθμοι.

ΚΕΦΑΛΑΙΟ 6

Συμπεράσματα και μελλοντικά βήματα

6.1 Εισαγωγή

Η τρέχουσα βιβλιογραφία προσφέρει κάποια εργαλεία για συμπίεση τροχιών αντικειμένων και ταιριάσματος τροχιών στον οδικό χάρτη. Κανένα όμως από αυτά δεν λύνει ξεχωριστά το πρόβλημα της συμπίεσης τροχιάς υπό τους περιορισμούς δικτύου. Η μέθοδος ταιριάσματος τροχιάς σε δίκτυο προσφέρει έναν βαθμό συμπίεσης και σε συνδυασμό με μεθόδους συμπίεσης μπορεί να το επιτύχει αυτό με μεγαλύτερη ταχύτητα. Παρόλα αυτά, η απλοποιημένη τροχιά που παράγει ο αλγόριθμος ταιριάσματος χάρτη δεν σημαίνει απαραίτητα ότι είναι και συμπιεσμένη ή ότι επιτυγχάνει τον μέγιστο βαθμό συμπίεσης. Τα πειραματικά αποτελέσματα που παρουσιάσαμε ενισχύουν αυτήν την άποψη. Γι αυτό το λόγο υπάρχει η ανάγκη για ανακάλυψη μιας νέας μεθόδου που θα πρόσφερε συμπίεση τροχιών πάνω σε δίκτυο.

6.2 Προτάσεις για περαιτέρω μελέτη

Στην παρούσα εργασία προτάθηκε μια μέθοδος συμπίεσης πάνω στο δίκτυο η οποία, αφενός επιτυγχάνει καλά αποτελέσματα συμπίεσης, αφετέρου όμως είναι αρκετά χρονοβόρα, λόγω της ανάγκης για εκτέλεση αλγορίθμου shortest path. Στόχος μιας μελλοντικής εργασίας θα μπορούσε να είναι η βελτίωση αυτής της μεθόδου, είτε χρησιμοποιώντας εναλλακτικούς αλγόριθμους shortest path, είτε μειώνοντας τα σημεία του χάρτη κρατώντας μόνο τα πιο σημαντικά.

Επίσης, θα μπορούσε να προταθεί μια λύση συμπίεσης η οποία θα αφαιρούσε από την τροχιά μας στο δίκτυο τις ακμές οι οποίες θα μπορούσαν να προβλεφτούν από τις προηγούμενες. Αναλυτικά, θα μπορούσαμε να αφαιρέσουμε μια ακμή η οποία

μπορεί να ανακαλυφτεί από την προηγούμενή της μέσω κάποιου αλγορίθμου όπως ο STTrace ή ο Thresholds, είτε να προβλεφτεί ως η μοναδική γειτονική ακμή της προηγούμενης. Επίσης θα αφαιρούσαμε συνεχόμενες ακμές μιας διαδρομής η οποία είναι το μοναδικό shortest path μεταξύ δύο σημείων και μπορεί εύκολα να υπολογιστεί από τον αντίστοιχο αλγόριθμο. Στη συγκεκριμένη περίπτωση όμως, το αποτέλεσμα δε θα ήταν τροχιά πάνω σε δίκτυο, απλώς ένα συμπιεσμένο αρχείο. Θα χρειαζόμασταν έναν αλγόριθμο απο-συμπίεσης που θα εκτελούσε την αντίστροφη διαδικασία ώστε να πάρουμε την αρχική τροχιά. Μια τέτοια λύση θα μπορούσε να είναι μη-απωλεστική όσον αφορά στη θέση που βρέθηκε το αντικείμενο, αφού οι παραπάνω αλγόριθμοι υπολογίζουν την επόμενη ακμή. Η πληροφορία του χρόνου όμως στις κορυφές δε θα μπορούσε να διατηρηθεί, απλώς να προσεγγιστεί κατά την απο-συμπίεση.

Συμπερασματικά, το ζήτημα συμπίεσης τροχιάς πάνω σε δίκτυο είναι ένα ανοικτό πρόβλημα, που δεν το καλύπτει η σύγχρονη βιβλιογραφία. Στην παρούσα έρευνα προσπαθήσαμε να αναπτύξουμε κάποιες πτυχές του και να προσφέρουμε μία βάση για περαιτέρω έρευνα στο μέλλον.

Αναφορές

[1] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On Map-Matching Vehicle Tracking Data. Proceedings of the 31st international conference on Very Large Data Bases (VLDB), 2005.

[2] T. Brinkhoff. Generating Network-Based Moving Objects. Proceedings of the 12th International Conference on Scientific and Statistical Database Management (SSDBM), 2000.

[3] T. Brinkhoff. Road network generator url: http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, Vol 1(1), pp. 269-271, 1959.

[5] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. The Canadian Cartographer Vol. 10(2), pp. 112-122, 1973.

[6] E. Frentzos, N. Pelekis, I. Ntoutsi, and Y. Theodoridis. 6th Chapter in "Mobility, Data Mining and Privacy", pp. 151-187, 2008.

[7] N. Meratnia and R. A. de By. Spatiotemporal Compression Techniques for Moving Point Objects. Proceedings of the Extending Database Technology (EDBT), 2004.

[8] D Pfoser and CS Jensen. Capturing the Uncertainty of Moving-Object Representations. Proceedings of the 6th International Symposium on Spatial Databases (SSD), 1999.

[9] M. Potamias, K. Patroumpas, and T. Sellis. Sampling Trajectory Streams with Spatiotemporal Criteria. Proceedings of the Scientific and Statistical Database Management (SSDBM), 2006.

[10] E. Tiakas, A. N. Papadopoulos, A. Nanopoulos, and Y. Manolopoulos. Trajectory Similarity Search in Spatial Networks. Proceedings of the 10th International Database Engineering and Applications Symposium (IDEAS), 2006.

Παράρτημα Α

Πίνακες πειραματικών αποτελεσμάτων

Πίνακας 2 – max. speed div.=75

Offline

Map Matching, Compression and then Map Matching

LookAhead	Thr		Total time		Compression	Distance
2	10	16	1	15	-111,11%	44,62%
4	10	31	1	31	-138,89%	50,03%
6	10	63	1	62	-111,11%	45,03%
4	10	32	1	31	-138,89%	50,03%
4	50	32	1	31	-122,22%	46,88%
4	90	31	1	31	-122,22%	46,87%

Compression and then Map Matching

LookAhead	Thr		Total time	Compression	Distance
2	10	1	16	-94,44%	40,35%
4	10	1	31	-94,44%	40,39%
6	10	1	47	-105,56%	44,71%
4	10	1	31	-94,44%	40,39%
4	50	1	32	-94,44%	40,59%
4	90	1	15	-94,44%	40,76%

Map Matching

LookAhead	Thr		Total time	Compression	Distance
2	10	16		-100,00%	42,90%
4	10	46		-88,89%	38,15%
6	10	32		-111,11%	44,51%

Πίνακας 3 - max. speed div.=75

Online

Map Matching, Compression and then Map Matching

LookAhead	Thr		Total time		Compression	Distance
0	10	16	1	15	-155,56%	47,04%
0	50	16	1	15	-177,78%	51,18%
0	90	16	1	15	-166,67%	52,72%

Compression and then Map Matching

LookAhead	Thr		Total time	Compression	Distance
0	10	1	16	-155,56%	47,05%
0	50	1	16	-155,56%	47,55%
0	90	1	16	-155,56%	48,73%

Map Matching

LookAhead	Thr	Total time	Compression	Distance
0	10	15	-155,56%	47,05%

Πίνακας 4 - max. speed div.=150

Offline

Map Matching, Compression and then Map Matching

LookAhead	Thr		Total time		Compression	Distance
2	10	47	1	16	44,78%	45,13%
4	10	62	1	31	38,81%	49,21%
6	10	94	1	31	44,78%	44,16%
4	10	47	1	47	38,81%	49,21%
4	50	78	1	31	26,87%	46,84%
4	90	78	1	31	32,84%	48,77%

Compression and then Map Matching

LookAhead	Thr		Total time	Compression	Distance
2	10	1	31	52,24%	11,72%
4	10	1	63	43,28%	46,79%
6	10	1	62	52,24%	12,53%
4	10	1	63	43,28%	46,79%
4	50	1	16	46,27%	41,73%
4	90	1	16	49,25%	35,64%

Map Matching

LookAhead	Thr	Total time	Compression	Distance
2	10	31	50,75%	11,00%
4	10	62	38,81%	49,31%
6	10	78	47,76%	36,61%

Πίνακας 5 - max. speed div.=150

Online

Map Matching, Compression and then Map Matching

LookAhead	Thr		Total time		Compression	Distance
0	10	16	1	15	49,25%	28,16%
0	50	15	1	16	49,25%	28,87%
0	90	15	1	16	49,25%	28,84%

Compression and then Map Matching

LookAhead	Thr		Total time	Compression	Distance
0	10	1	16	49,25%	28,17%
0	50	1	15	49,25%	28,76%
0	90	1	16	52,24%	13,71%

Map Matching

LookAhead	Thr		Total time	Compression	Distance
0	10	16		49,25%	28,16%

Πίνακας 6 - max.speed div.=250

Offline

Map Matching, Compression and then Map Matching

LookAhead	Thr		Total time		Compression	Distance
2	10	141	1	31	77,92%	63,98%
4	10	141	1	31	72,73%	48,54%
6	10	219	1	47	74,03%	28,64%
4	10	141	1	31	72,73%	48,54%
4	50	141	1	16	79,87%	71,80%
4	90	125	1	32	79,87%	71,35%

Compression and then Map Matching

LookAhead	Thr		Total time	Compression	Distance
2	10	1	47	72,73%	47,67%
4	10	1	94	72,73%	47,75%
6	10	1	141	72,73%	47,58%
4	10	1	94	72,73%	47,75%
4	50	1	47	72,73%	54,18%
4	90	1	31	81,82%	66,39%

Map Matching

LookAhead	Thr	Total time	Compression	Distance
2	10	78	74,03%	28,76%
4	10	141	74,03%	28,76%
6	10	203	74,03%	28,76%

Πίνακας 7 - max. speed div.=250

Online

Map Matching, Compression and then Map Matching

LookAhead	Thr		Total time		Compression	Distance
0	10	16	1	15	72,73%	47,41%
0	50	16	1	16	77,92%	71,05%
0	90	15	1	16	83,77%	52,73%

Compression and then Map Matching

LookAhead	Thr		Total time	Compression	Distance
0	10	1	16	72,73%	47,29%
0	50	1	16	72,73%	47,16%
0	90	1	15	83,12%	57,54%

Map Matching

LookAhead	Thr		Total time	Compression	Distance
0	10	16		72,73%	47,39%

Παράρτημα Β

Υλοποίηση σε java του MapComp αλγόριθμου

```
    //Vector p is the trajectory on the map, max is the maximum
comression we desire

    //and out is the output Vector

    public void MapComp(Vector p, double max, Vector out){

        final int [][] pr= new int [p.size()][G.size()];//Vector
with the shortest paths

        double c=0,maximum=0;

        int s,e,j,test=0,i;

        Vector path=new Vector();

        Vector pros=new Vector((Vector)p.clone());

        //pre-find the shortest paths

        for(i=0;i<p.size();i++)

            pr[i]=dijkstra(G,((point)p.elementAt(i)).id);

        //find the maximum possible compression

        path.add(new
point(((point)p.elementAt(0)).x,((point)p.elementAt(0)).y,((point
)p.elementAt(0)).t,((point)p.elementAt(0)).id));


findPath(G,((point)pros.firstElement()).id,((point)pros.lastEleme
nt()).id,pr[0],path,((point)pros.firstElement()).t,((point)pros.l
astElement()).t);

        maximum=(double)(p.size()-path.size())/(p.size());

        //run at least once (one pass)

        do

        {

            test=0;

            j=0;

            s=((point)pros.elementAt(0)).id;

            out.removeAllElements();

            out.add(new
point(((point)p.elementAt(0)).x,((point)p.elementAt(0)).y,((point
)p.elementAt(0)).t,((point)p.elementAt(0)).id));

            for(i=1;i<pros.size();i++) {

                path.removeAllElements();

                //check if the point is one of the originals

                for(int k=0;k<p.size();k++) {


if(((point)pros.elementAt(i)).id==((point)p.elementAt(k)).id)

                        test=1;
```

```
                }
                //if not continue to the next one
                if(s==((point)pros.elementAt(i)).id||test==0)
                    continue;
                //find the shortest path to the next point

findPath(G,s,((point)pros.elementAt(i)).id,pr[j],path,((point)pro
s.elementAt(j)).t,((point)pros.elementAt(i)).t);
                //if the shortest path has less points than the
original path replace add it to output
                if(path.size()<=(i-j)) {
                    out.addAll(path);
                    s=((point)pros.elementAt(i)).id;
                    int z=0,g=0;
                    double dist1=0,dist2=0,t1,t2;
                    //find the time at the nodes
                    for(int l=out.size()-path.size();
l<out.size();l++,g++) {
                        //if we don't know the time for that node

if(((point)out.elementAt(l)).id!=((point)pros.elementAt(j+g)).id)
{
                            z=0;
                            //left distance
                            while(((point)out.elementAt(l-
z)).t==0) {

dist1+=Math.sqrt(Math.pow(((point)out.elementAt(l-z)).x-
((point)out.elementAt(l-z-
1)).x,2)+Math.pow(((point)out.elementAt(l-z)).y-
((point)out.elementAt(l-z-1)).y,2));
                                z++;
                            }
                            t1=((point)out.elementAt(l-z)).t;
                            z=0;
                            //right distance

while(((point)out.elementAt(l+z)).t==0) {

dist2+=Math.sqrt(Math.pow(((point)out.elementAt(l+z+1)).x-
```

```
((point)out.elementAt(l+z)).x,2)+Math.pow(((point)out.elementAt(l
+z+1)).y-((point)out.elementAt(l+z)).y,2));
                                    z++;
                                }
                                t2=((point)out.elementAt(l+z)).t;

((point)out.elementAt(l)).t=(dist1*t2+dist2*t1)/(dist1+dist2);
                            } else

((point)out.elementAt(l)).t=((point)pros.elementAt(j+g)).t;
                        }
                        j=i;
                    }
                }
                //add the rest of nodes
                out.addAll(path);
                //add the time-value of the rest of the nodes
                for(int k=j;k<pros.size();k++) {
                    int l=out.size()-(pros.size()-k);

((point)out.elementAt(l)).t=((point)pros.elementAt(k)).t;
                }
                pros.removeAllElements();
                pros.addAll(out);
                //calculate the compression achieved
                c=(double)(p.size()-out.size())/(p.size());
            }
            while(c<max&&c<maximum);//repeat until we obtain the
maximum wanted or achievable compression
        }
```

Εικόνα 41- Κώδικας σε java του αλγόριθμου MapComp